1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISION^{SM} RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

If pages are missing, contact the university which granted the degree.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Automating the Analysis of Complex Physical Systems - The Virtual Foundry

by

Mehmet Naci Dai

M.A.Sc., Technical University of Nova Scotia

B.S.M.E., Bogaziçi University

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Department of Mechanical and Aerospace Engineering
Ottawa-Carleton Institute
for Mechanical and Aerospace Engineering
Carleton University
Ottawa, Ontario
Canada

August 8, 1994

©Copyright 1994

Mehmet Naci Dai

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file   Votre référence

Our file   Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-98447-3

Name MEHMET NACI DAI

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

THE HUMANITIES AND SOCIAL SCIENCES

SUBJECT TERM

0548

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0401
Literature 0294
General 0295
Classical 0297
Comparative 0298
Medieval 0316
Modern 0591
African 0305
American 0352
Asian 0355
Canadian (English) 0593
Canadian (French) 0311
English 0312
Germanic 0315
Latin American 0313
Middle Eastern 0314
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography 0578
Gerontology 0578
History 0578
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0337
United States 0585
History of Science 0398
Law 0615
Political Science 0616
General 0617
International Law and Relations 0814
Public Administration 0452
Recreation 0626
Social Work 0627
Sociology 0938
General 0631
Criminology and Penology 0628
Demography 0629
Ethnic and Racial Studies 0630
Individual and Family 0700
Studies 0344
Industrial and Labor 0709
Relations 0999
Public and Social Welfare 0453
Social Structure and Development 0537
Theory and Methods 0539
Transportation 0540
Urban and Regional Planning 0541
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0329
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0231
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0748
Atomic 0607
Electronics and Electricity 0798
Elementary Particles and High Energy 0759
Fluid and Plasma 0609
Molecular 0610
Nuclear 0752
Optics 0756
Radiation 0611
Solid State 0463
Statistics
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0670
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Carleton University
Ottawa, Canada K1S 5J7

Thesis contains black and white and/or coloured graphs/tables/photographs which when microfilmed may lose their significance. The hardcopy of the thesis is available upon request from Carleton University Library.

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,

**Automating the Analysis of Complex Physical Systems -
The Virtual Foundry**

submitted by
Mehmet Naci Dai

in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

A handwritten signature in black ink, appearing to read 'C. A. Gold', written over a horizontal line.

Thesis Supervisor

A handwritten signature in black ink, appearing to read 'Vine Thomson', written over a horizontal line.

External Examiner

A handwritten signature in black ink, appearing to read 'Robert Bell', written over a horizontal line.

Chairman,

Department of Mechanical and Aerospace Engineering

Carleton University
Ottawa, Ontario
Canada

Anne ve Babama

To my parents

To my grand parents and family

To Karen

for years of love and support

Table of Contents

Table of Contents	i
Acknowledgments	iv
Abstract	v
List of Figures	vi
List of Tables	viii
Nomenclature	ix
Chapter 1: Introduction	1
1.1 Why wear three different hats?	3
1.1.1 Casting	4
1.1.2 Basic physics	5
1.1.3 Interactions and integration	5
1.2 The interaction problem	7
1.3 Essential attributes of a software environment to support design of mechanical systems	10
1.3.1 Multiple viewpoints	11
1.3.2 Asynchronous (dynamic), inconsistent, incomplete design	12
1.3.3 Heterogeneous and distributed	12
1.3.4 Negotiated	12
1.3.5 Version management	13
1.3.6 Multiple levels of abstraction	13
1.3.7 Function	13
1.3.8 Language of communication and contract	14
1.4 A portrait of the thesis	15
1.4.1 Casting design	15
1.4.2 Virtual foundry	18
1.4.3 Basic physics and mathematical foundations of casting analysis	19
1.4.4 Agents	22
1.4.4.1 Agents for analysis	24
1.5 A road map	25
Chapter 2 : Designing a casting	27
2.1 Can we do better than just guessing?	27
2.2 Prelude: Tradition of casting design	28
2.2.1 Priorities	29
2.2.2 Methods	30
2.2.3 Are there solutions?	31
2.3 Traditional analysis software	31
2.3.1 Limitations	31
2.4 A casting analysis environment and a methodology for using it	32
2.4.1 The study of a casting during the various stages of design	36
2.4.1.1 Initial assessment of the casting without the rigging	37
2.4.1.2 The design and development of the rigging	40
2.4.1.3 Analysis during filling	44
2.5 Remarks	47
Chapter 3: Virtual Foundry - Software for casting design	48
3.1 Tools versus environments	52
3.2 The intelligent design assistant	53
3.3 Virtual foundry	54
3.3.1 Shared knowledge	55

3.3.1.1 The interface with the designer: domain knowledge	56
3.3.1.2 The interface between the assistant and analysis software	57
3.3.2 The key components of the system	58
3.4 Casting interface	61
3.4.1 Geometry	63
3.4.1.1 Rigging	64
3.5 Analysis	68
3.6 A remark on the theory of design	69
3.7 Summary	71
Chapter 4: Basic physics and mathematical foundations of casting analysis	72
4.1 The fluid flow equations	73
4.1.1 The conservation laws	74
4.1.1.1 Conservation of mass	74
4.1.1.2 Conservation of momentum	76
4.1.1.3 Conservation of energy	78
4.1.2 The incompressible fluid flow equations	78
4.1.3 Numerical solution of the incompressible Navier-Stokes equations: FEM discretization	81
4.1.3.1 Finite element spatial discretization	82
4.1.3.2 Time integration	85
4.1.3.3 Remarks	86
4.1.4 Numerical example	87
4.2 The free-surface equations	89
4.2.1 The numerical solution of free-surface equations	90
4.2.1.1 Computing the gradient of Ψ	94
4.2.1.2 Flux limiting by linear programming	95
4.2.1.2.1 Standard form:	97
4.2.2 Numerical example	98
4.3 The energy equation	100
4.3.1 Incompressible flow	101
4.3.2 Solidification modeling	101
4.3.3 The discrete form of the energy equation	102
4.3.3.1 Time stepping	105
4.3.4 Numerical experiments	105
4.4 Coupled differential equations for casting analysis	105
4.4.1 Continuum equations	106
4.4.2 Systems of equations	108
4.4.3 Navier-Stokes equations and free-surface evolution	109
4.4.4 Flow equations and thermal transport problem	111
4.4.4.1 Cold-shut problem	111
4.4.4.2 Shrinkage analysis	112
4.5 Summary	114
Chapter 5: Formalizing interactions - Multi Agent Systems for Solving Coupled Problems	115
5.1 The model of integration	118
5.2 The agent	120
5.3 The protocol for cooperation	127
5.3.1 Control	128
5.3.2 Data dependency	128
5.3.3 Consistency	129
5.3.4 Messages and handling messages in agents	131
5.3.5 Task and result sharing	132

5.4 An implementation of agents for coupled problem solving	133
5.4.1 Structures for agents	134
5.4.1.1 Variables	134
5.4.1.2 PDE solver and the message handler	136
5.4.2 Messages	138
5.4.3 Solving a transient problem: managing data and control	139
5.4.4 The agent graph	140
5.4.4.1 Agent activity and the versions of data in the graph	142
5.4.4.2 Transient analysis	143
5.4.4.3 Time-walk problem	143
5.4.4.4 Managing time	145
5.4.5 Structures for managing cooperation	147
5.5 Distributed/Parallel processing	152
5.6 Relation to other work	154
5.6.1 Object oriented programming	155
5.6.2 Distributed/Parallel processing	157
5.6.3 Artificial intelligence and multi-Agent systems	159
5.6.4 Manufacturing automation systems and languages	160
5.6.5 The OSI reference model	161
5.7 Summary	162
Chapter 6: Agent Examples	163
6.1 Example I - Autonomous agents	164
6.1.1 Agent A ₁ :	164
6.1.2 Agent A ₂ :	164
6.1.3 The coupling	165
6.1.4 The behavior of the system - solving the equations	165
6.1.4.1 The solution	166
6.2 Example II - Strongly coupled agents	168
6.2.1 Agent A ₁ :	170
6.2.2 Agent A ₂	170
6.2.3 Agent A ₃	171
6.2.4 The coupling	171
6.2.5 Managing interactions of the agents	172
6.3 Example III - Fluid flow and free surface	175
6.3.1 Fluid flow agent A _{ff}	176
6.3.2 Free surface agent A _{fs}	176
6.3.3 The coupling	176
6.3.4 Solving the equations	177
6.3.5 Numerical example	178
Chapter 7: Postscript	179
7.1 Contributions	179
7.2 Outstanding questions	180
7.3 An overview	182
7.3.1 Virtual foundry	182
7.3.2 Basic physics of casting	183
7.3.3 Software	184
Appendix A	186
References	197

Acknowledgments

The list of people who have directly and indirectly contributed to this thesis is very long. I thank to all of them. I would like to specially thank to Dr. John A. Goldak, my thesis supervisor, whose endless enthusiasm and dedication to solve real problems have taught me how rewarding and challenging research is. I would also like to thank him for his guidance and support. Dr. Edward Hughes, who has provided invaluable guidance with the mathematics and teaching me the real meaning of clarity, without him this thesis would not have been possible. Dr. Adam C. Bell for giving me the chance and letting me discover my potential. Robert Tjoa for using the software to solve "real" problems and helping with the implementation. Mike New for helping to implement the user interface and for the after hours at "Mike's Place". Kevin Haaland for his help in implementing the agents, reading the thesis and making me work late at night. Alan Knight for being so critical and fussy about software design. Dan Downey for being skeptical. Dr. Weidian Ma, for providing the initial FEM implementations for the Navier-Stokes and Energy equations. Maoshi Gu for using agents to solve welding problems and helping to improve their design. Akbar Mashaie for reading the sections on math. Guojian Ma for using agents to implement his work. Haley Industries for letting me use the "Indy" wheel. And to many others who have contributed to this thesis in different ways and whose names I failed to mention.

Abstract

The object of this thesis is to create an automated analysis capability for complex physical systems that can be integrated into software environments for design. These environments are intended to support the process of design not, to create or propose new designs. In developing these environments, the difficulties that have limited the success of traditional software environments for mechanical design are studied.

Software is developed to support the design of casting. The behavior of a casting is predicted from the underlying physics and mathematics. This system simulates a foundry for designers: they can experiment with their designs while the system provides feedback by doing the analysis. This environment is called the "*virtual*" foundry.

The physics of casting is complex. It involves different length and time scales. The partial differential equations are non-linear and dynamic. However, it has not been the lack of the physics, software tools or computer hardware that has inhibited the development of design environments for foundries.

Progress in the broader problem of integration of the islands of automation in design software has been modest. A hypothesis of this thesis is that in order to do significantly better than previous integration efforts, a formal framework is needed to automate the integration of analysis capability into design environments.

This thesis develops a distributed, graph based model that can represent coupled complex phenomenon as a network of communicating agents. Agents are a decomposition of the complex system. Graphs of agents can alter their topology and therefore the description of the solutions, as the problems evolve. This is a means of enabling a program to use high level components as primitives. The agent model is used as the framework of integration to develop the software for modelling continuum mechanics problems, and to develop the environment for the users.

The software and methodology that deals with the interactions between agents, i.e., everything except the internal structure of agents, are domain independent. It is argued that this generalization is a useful concept of a design environment that can be applied to other design domains.

List of Figures

Figure 1.1	Casting design and analysis	6
Figure 2.1	Typical rigging designs	28
Figure 2.2	A window which belongs to the geometry module displaying a gating diagram and the corresponding FEM mesh	33
Figure 2.3	Integrating analysis into casting design	34
Figure 2.4	The wheel	36
Figure 2.5(a)	Two dimensional axisymmetric analysis of the wheel casting. (i) Temperature and fraction of solid distributions at two instants of time (ii) The temperature versus time plot for several points in the casting (iii) Fluid flow induced by the volumetric changes due to solidification.....	38
Figure 2.5(b)	The temperature and fraction of solid contours of the sixty degree slice from the wheel	39
Figure 2.6	Thermal and solidification analysis for different rigging configurations of a sixty degree section of the wheel (a) The slot gate with uniform thickness is located behind the spoke (b) The slot gate with a bulging section is located near the spoke and a riser is added to the hub (c) The slot gate is moved between the spokes (d) Temperature wrt. time plot at a point near the spoke where the gate is attached to the wheel	41
Figure 2.7	The flow induced by the volumetric changes due to solidification. The velocity vectors and the regions where there is an outflow of fluid are illustrated	43
Figure 2.8	The filling simulation of two alternative runner designs for a multiple part casting. The taper in the runner is adjusted for uniform filling of the three plates	45
Figure 2.9	The filling simulation for a two part casting.....	45
Figure 2.10	The thermal-flow analysis of a cold shut problem with cylindrical runner cross-section. The evolution of temperature, fraction of solid and velocity fields are presented at earlier and later stages of filling.....	46
Figure 3.1	Role of software and experiment in building Conceptual Models.....	49
Figure 3.2	Functional Decomposition of the "Virtual Foundry"	55
Figure 3.3	Data flow through the software components of the Virtual Foundry	58
Figure 3.4	The casting interface illustrated by the snapshot of a screen displaying the rigging diagram, editor for a sprue and the casting interface. Final geometry of the wheel casting with a rigging attached can be seen in the casting interface.....	62
Figure 3.5	A functional breakdown of the casting interface	65
Figure 3.6	Gating Primitives	66
Figure 3.7	A sample rigging assembly and the constraint system	68
Figure 4.1	Fluid body Ω , with boundary $\partial\Omega$ and outward unit normal n	74
Figure 4.2	Volume Ω as it moves with the flow	75
Figure 4.3	Traction	77
Figure 4.4	$w = u + \text{grad}(p)$	80
Figure 4.5	The Lid-Driven Cavity Problem (a) Velocity vectors and flow paths (b) The x-components of the velocity vectors on a vertical line passing through the geometric center of the cavity (c) The y-	

	components of the velocity vectors on a horizontal line passing through the geometric center of the cavity.....	88
Figure 4.6	The free-surface Π	89
Figure 4.7	The stresses acting on Π	89
Figure 4.8	The neighborhood of element e	95
Figure 4.9	Approximately 180 degree rotation of a region resembling a cylinder with a rectangular slot (a) A series of VOF contours at consecutive times illustrating the evolution of VOF (b) The VOF distribution along a line cutting through the cylinder and the rectangular slot at different angles of rotation	99
Figure 4.10	Collapse of a fluid column under the influence of gravity (a) The VOF ≥ 0.5 region and the velocity vectors at about $t=2.5$ seconds (b) A comparison of the simulation results with experimental data	110
Figure 4.11	Formation of a cavity caused by the volumetric changes due to solidification. The cylindrical section loses heat on all the walls except the top. The solidification starts at the walls and propagates towards the interior. The velocity field induced by the volumetric change causes the sinking of the liquid-gas free surface	113
Figure 5.1	A transformation of conservation equations to agents.....	123
Figure 5.2	The graph analogy separating internal and external representation.....	125
Figure 5.3	Classes of variables for PDE agents	136
Figure 5.4	Components of an Agent.....	137
Figure 5.5	Message structure.....	138
Figure 5.6	Data dependency graph.....	140
Figure 5.7	An agent graph	141
Figure 5.8	Branching trees	142
Figure 5.9	Topological sort of the agent graph for a time step	144
Figure 5.10	The tree structure for agents.....	147
Figure 5.11	Messages and the message handler	149
Figure 5.12	The states of an agent. (a) The states of the agent assuming the role of a server. (b) The states of an agent assuming the role of a client.	151
Figure 6.1	The coupling between agents A_1 and A_2	165
Figure 6.2	The trajectory of the solutions of the van der Pol's equation	167
Figure 6.3	The periodic solution for $x(t)$	167
Figure 6.4	The periodic solution for $y(t)$	167
Figure 6.5	Data flow	171
Figure 6.6	Control flow	171
Figure 6.7	The coupling between agents A_{ff} and A_{fs}	177
Figure A.1	(a) The problem description and (b) The agent description	187

List of Tables

Table 5.1	Agents for continuum mechanics.....	135

Nomenclature

u	Velocity
p	Pressure
ρ	Density
T	Temperature
$\mathbb{R}^+$	Positive real numbers
$\mathbb{R}^3$	Three-dimensional Euclidean Space
D	A region in space
Ω	A sub region (domain) contained in D
$\partial\Omega$	The boundary of the region (domain) Ω
n	The unit outward normal
x	A material point in the region (domain) Ω , $x \in \Omega$
t, t_0, t_f	Time, $t \in \mathbb{R}^+$
∇, grad	The gradient operator
$\nabla \bullet, \text{div}$	The divergence operator
Δ	The Laplacian operator, i.e., $\text{div}(\nabla\psi)$, ψ is a scalar
Λu	$\frac{1}{2}(\nabla u + \nabla u^T)$. The symmetric gradient operator
σ	Stress tensor
T_p, T_{viscous}	Stress tensors due to pressure and viscous forces
I	The identity matrix
Σ	Surface separating two fluid regions
μ	Kinematic viscosity, first coefficient of viscosity
λ	Bulk viscosity, second coefficient of viscosity
U	Internal energy
KE	Kinetic energy
Q	Net rate of heat flow into a particle by its surroundings
P	The work done on a particle
ψ	The ratio of mass of one component to the total mass of the fluid
w	A vector field
P	A linear orthogonal projection operator
$\tilde{u}, \tilde{p}$	The nodal coefficients of the finite element approximations of the velocity and pressure
$\{\tilde{u}\}, \{\tilde{p}\}$	The vectors of nodal coefficients of the finite element approximations
ϕ, φ	Finite element basis functions
G	The element volume (gram) matrix
K	The diffusion matrix
$N(u)$	The advection matrix
B	The finite element divergence operator
B^T	The finite element gradient operator
A	The matrix $BG^{-1}B^T$
b	The finite element load vector
h	The tractions, pseudo-tractions

$\tilde{u}$	The normalized velocity, i.e., $\frac{\tilde{u}}{\ \tilde{u}\ }$
α	Diffusivity
γ	The streamline-upwinding free parameter
$\hat{u}^{n+1}$	The trial solution for the nodal approximation of the velocity
Q	The intermediate pressure increment
θ	An integration constant
Re	The Reynolds number
Pe	The Peclet number
Π	The free surface
R_1, R_2	The principal radii of curvature
τ	The surface tension coefficient
δ_{ij}	Kronecker delta
$\Gamma_{in}, \Gamma_{out}$	Inflow and outflow boundary
A_f	The area of a face
V_e	The volume of an element
c_e, c_f	Element and face centroid
c_i	Concentration
ℓ	The characteristic length of an element in the direction of flow
g	The gravitational constant
g, g_x, g_y, g_z	Gradient and the components of the gradient in the direction of x, y and z
$\hat{g}$	The approximation to the gradient
ϖ	enthalpy
ε	internal energy per unit mass
q, j	flux
s	entropy
κ	Thermal conductivity
ν	Dynamic viscosity
C_p	Specific heat
L	Latent heat of fusion
F_s, f_s	Volumetric fraction of solid
f_k	Volumetric fraction the component k
$\tilde{T}$	The nodal coefficients of the finite element approximations of the temperature
$\{\tilde{T}\}$	The vector of nodal coefficients for temperature
β	Volumetric shrinkage coefficient

Chapter 1: Introduction

The behavior of a design can be predicted from the underlying physics and mathematics. The primary emphasis of this thesis is to create a capability for analysis that is integrated into the design processes and environments. The software environment that is described here is a supportive system not a creative one, i.e., it will not suggest or generate new or original designs.

*Hypothesis: Software environments to support designers
(or the design process) can be constructed.*

The quest for better designs began in ancient times. With current technology, tools to aid design are based on three essential islands of knowledge and automation, mathematical and physical understanding, manufacturing technology, software and computer technology. Although, strongly related to each other, integrating these islands

into the design process has proved to be a hard problem. With the exception of electronics, few examples of fully integrated, computer-aided technology covering the complete product design cycle from the conceptual realization to manufacturing exist. The answer to the problem of creating an integrated capability for design, if the answer exist, does not lie within any of these islands. Currently, integration is largely an unexplored research area. Very few individuals are able to even read the current literature in the three fields of applied mathematics and physics, manufacturing technology, software and computer technology. This supports the argument that research in integration is immature.

Quality cannot be manufactured, unless it has been designed into the product. It can best be achieved by predicting and solving problems as early as possible in the design cycle. Obviously, any tool or methodology that enhances our capability to predict problems could be valuable. Scientifically based simulations of the physics and computational environments that are an integral part of the design process can be an important part of the answer. Simulations can support the design process by facilitating educated design decisions, which in turn can improve the functional and structural quality, reduce costs and development time. What is required to design and manufacture a quality product? How can the knowledge and understanding of mathematics, physics, engineering and software be embedded in a tool that is integrated into the design process?

Hypothesis: Software based design aids can be developed and integrated into the design process and by facilitating analysis, they will help the designer to explore the design space more effectively by experimenting with design parameters. This will help to detect defects and to optimize the design.

There is agreement among some researchers¹ that if a theory of design is developed, it will probably evolve from advances in computer assisted tools for design

---

¹ Finger & Dixon review design research in mechanical engineering [1, 2] .

processes. "Virtual Manufacturing" environments are needed that enable experiments to be carried out in software before any action takes place on the factory floor.

This thesis is about a software environment for casting design. It is, however, more than just that: The basic objective is to provide a formal system that can be used for applications other than casting design to automate the integration of analysis capability into software environments. Thus, abstract or generalized software to create and integrate design environments is sought.

1.1 Why wear three different hats?

The complexity of this thesis will strongly depend on how it is presented. On the assumption that complexity evolves from simplicity, I will attempt to find the simplest building blocks and express them in a hierarchical manner. A different hat will be worn to present each alternative perspective. Once these building blocks are outlined, it will be easier to establish the relations and see the larger patterns.

This work has three unique facets, which are quite distinct, yet strongly tied to each other:

- *Casting* - Provides a problem domain that limits the scope
- *Basic physics and mathematics*
- *Integration* - Integration of the tools into the design process and integration of diverse software to construct computational analysis tools for design

The following sections of this chapter provide a general overview of the thesis. In the context of these sections, there will be brief statements on the hypotheses of this thesis, which provide different perspectives and levels of detail for its objectives.

1.1.1 Casting

Casting is the preferred method of manufacturing for complex shaped parts. However, the costs associated with a casting can be high. It is not unusual for a foundry to spend one to five years to bring a casting into the market. The process of transforming engineering drawings of a part into a pattern and then into a finished cast product is a long, labor intensive process. Casting problems will provide concrete examples and a context for this thesis.

Casting design in the foundry industry, which dates back to 3000 BC², currently lacks a strong basis in physics and science. The manufacturing technology of castings has evolved significantly. However in many ways, the method of design has changed little since the earliest times. Experiments with test castings are carried out until a satisfactory casting is produced. Design relies more on the expertise³ of seasoned craftsman than on scientific analysis and engineering.

The study of casting design and the employment of numerical analysis tools in assessing the quality of designs will provide a context for the exposition of the physical significance of the analysis methods, their contribution to the manufacturing of castings, and aspects of software architecture. As foundries base more designs for castings on science and engineering analysis, the number of experiments with test castings should be reduced significantly. Foundries would be expected to have more control over the casting process, and produce higher quality parts at lower costs with shorter delivery times.

---

²See Simpson [3] for a brief history of the metal casting industry.

³The expertise of a designer can be also based of scientific and engineering principles. Here, the analysis of a particular design is implied.

1.1.2 Basic physics

The geometry of a casting is often complex. The physics is usually non-linear. Realistic analysis not only requires understanding and modeling a variety of physical phenomena, but also the complex interactions between them. The basic physics such as thermodynamics, mass and heat transfer, phase transformations have long been studied and are often, though not always, well understood. The formal mathematics to model most of the physics is usually known. During the last few decades many of these models have been implemented computationally, in specific industries, such as gas turbine design; this has stimulated further study into more detailed and accurate models. In many cases, these advances have not yet substantially improved the quality of products or the understanding of manufacturing processes, nor are they used by the engineer who has to deal with these problems on a day to day basis. It is one of the goals of this thesis that what is now available and accepted science to mathematicians and physicists, should be made accessible and useful technology to a practicing engineer.

1.1.3 Interactions and integration

Casting design

Consider the overall social or organizational scenario of the design of a casting.

The customer, who has designed a component, approaches a foundry to have it cast. It may be the engine block of a car, a turbine blade, or a simple bend in a pipe. The customer has detailed information specifying the function and desired behavior. Their knowledge of casting technology is usually very limited. They believe that the part should be cast; a certain alloy should be used; and it should satisfy specified performance

criteria when it is delivered. The customer is usually a complex organization such as one that designs engines or even automobiles and aircraft.

When the foundry receives this order, including the engineering drawings, another social or organizational network appears. The Foundry's Methods Engineer determines if this part can be produced at all; if the metal can be delivered to all sections, and if this process can be realized economically. Once a decision is made to manufacture the part, a gating system will be designed to deliver the metal. A new geometry is specified to account for shrinkage. The metallurgist seeks to achieve the desired material and mechanical properties specified by the customer. The methods engineer seeks to optimize the gating system, flow patterns and yield ratios. They would use whatever tools, analysis, testing, etc., that are available to them to make the best decisions they can.

Figure 1.1: Casting design and analysis

Figure 1.1, shows a schematic of the interactions among these entities. Each of the boxes (agents) in this figure can be individuals or groups. Internally, they do not have to share a common language, a common representation or common knowledge. Each has their own beliefs, views, symbols, models, assumptions, abstractions and perceptions. They each conduct their own part of the task assuming certain parameters are fixed by the rest of the world and usually cannot be changed, but sometimes, under very strict rules, changes can be negotiated. Communications with other agents takes place in terms of shared terms and protocols. Forms of communication and communication patterns are established between each pair of agents. These usually differ for each pair.

The modern "*concurrent*" engineering approach would propose to involve the foundry in the earlier stages of the component design. However, this is a hard problem. This interaction is also illustrated in Figure 1.1.

1.2 The interaction problem

Analysis for design is an important part of the overall design process in mechanical engineering. The literature on analysis methods and the validity of their predictions is vast. However, the literature on integrating these methods into the design process is remarkably sparse. Although it has been recognized since the 1970's, that the integration of analysis into the design process is of primary importance, little progress has been made as a scientific discipline (Kuhn's⁴ Terms). This thesis assumes one of the reasons for the lack of progress is that design is distributed, evolutionary, negotiated, incomplete and inconsistent. Multiple viewpoints and multiple abstraction levels are often exploited. Software engineering methods to deal with these issues are not well developed. This has hindered efforts to integrate software.

---

⁴ In "Structure of scientific revolutions", Kuhn [4].

The quest to model a design in a computational environment is as old as the history of computers. Attempts to create software environments to support design in mechanical engineering at its various stages have evolved through several phases: The early single large FORTRAN programs would provide answers to particular aspects of a problem, which with reformatting could be used as the inputs to another program. A human operator, the facilitator of the integration, would physically take the answers and prepare new input files for the next program. This is a time consuming, unreliable and inefficient method and generally it is very costly. Nevertheless, it is available and problems are solved with it. Industry continues to use such software systems; in fact, much of the software installed in industry falls into this category.

Next, there were "n" smaller programs and " n^2 -n" file formatting programs. This solution was not satisfactory either. System level support is impossible for integration at the level of files. Writing and maintaining " n^2 -n" formatting programs is also time consuming and expensive.

The third wave of automation efforts focused on database technology. The examples of this stage are the centralized relational databases of the past two decades (70's/80's). The intention was to automate the integration by providing a common representation and data access facility. A human operator would not need to perform file conversions, the data would be shared and would be rich enough to address all demands. These projects, such as IPAD [5], did not recognize the distributed, delegated and decentralized, inconsistent and incomplete nature of design. The database approach, soon ran against the wall of complexity and the rich variety of largely distributed and inconsistent representations of data. This inconsistency is one of the fundamental difficulties that makes integration a hard problem.

Although a great deal of progress has been made within each of the islands of automation, progress on the essential problem of integration or interactions between the islands of automation in design environments in software has been modest.

Hypothesis: A set of communicating, independent agents with separate name and address spaces will lead to a significant advance in the capability to develop integrated design environments.

The minimum requirement and a significant advance is to recognize and characterize the independent agents currently involved in the design process. Each agent can have its own internal representation and language, and its own way of conducting its internal affairs. Agents communicate with each other using a relatively small set of common terms and parameters. Agents are described in detail in chapter 5.

The fundamental abstraction is a group of communicating agents with a clear separation of internal behavior and external environment such that an invariant relation between the inner system and the function/goal/purpose of the agent is maintained. This philosophy is already an essential feature of many contemporary advances in software technology such as object-oriented languages, distributed algorithms, etc.

Description of an agent concept, and an implementation based on the agent model to solve coupled and distributed problems is one of the contributions of this thesis. The formal systems developed to express the internal and external affairs of these agents of automation must recognize the fundamental complexity of design and the complexity of software. An agent model captures some of the essential features of design. It is distributed not monolithic. It is delegated and decentralized. The collective activities of these agents define the behavior of the system. The configurations and connections of the agents are programmable and dynamic. The design graph and the associated schema evolve in time. For these reasons, agents have the potential to provide better design environments.

1.3 Essential attributes of a software environment to support design of mechanical systems

What are the essential attributes of a software environment to support design of artifacts or mechanical systems such as an engine or transmission housing? There are software based design systems that provide sophisticated support for analysis, e.g., landing gear design (Boeing), shoe design (Bata/Nike), hydraulic turbine design (GE), etc. However, these environments and analysis tools usually are neither designed to be reused for other problems nor do they provide support to develop systems for other applications.

With the exception of a few⁵ systems, widely accepted software environments for mechanical design are not known to the author. Why is this so? The fact that they do not exist in spite of serious efforts that have been made to create them, suggests that fundamental problems must be overcome to create them.

In the early 1970s Voelcker and Requicha [6, 7] identified computational geometry, i.e., the lack of representations and procedures for geometry as a fundamental difficulty. Much progress has been made to resolve this difficulty. This is an example of a fundamental difficulty.

This section presents a taxonomy of difficulties that limit the creation of software environments for design. For the purpose of this discussion, it is assumed that existing Computer Aided Design (CAD) tools can be integrated. It is appreciated that this is unlikely, but we consider this an accidental difficulty because the methodology of existing CAD tools can certainly be integrated.

---

⁵Piping design is a well developed area of software assisted mechanical design. There are also software environments for design and analysis of lumped parameter kinematic and dynamic mechanical systems.

1.3.1 Multiple viewpoints

All design considers at least the following viewpoints:

- Geometry
- Material properties
- Performance
- Manufacturing processes
- Cost
- Function

CAD systems are designed to support geometric modeling. Their capability of specifying material properties is usually limited to specifying a material type. Sometimes CAD systems are connected to an NC machining program. Usually the NC machining program specifies tool paths and to a limited extent does process planning.

We know of no commercial CAD systems that include reasoning about thermal-deformation processes such as casting, forging, welding, etc. We know of no CAD systems that include cost accounting⁶. We know of no CAD systems that include function. Without knowing function, a CAD system cannot evaluate a design because function defines intent.

It can be safely concluded that multiple viewpoints will be needed in next generation CAD systems. Another example of multiple viewpoints in the case of casting design was presented in section 1.2.

What is needed to support multiple viewpoints? In our opinion, multiple viewpoints is the first requirement. In this thesis, a model based on multiple agents will provide an example of how to provide this support.

---

⁶There are systems that can provide the cost of an assembly by using tabulated costs of the sub-parts. The integration of cost accounting to the design of specific parts, i.e., relating cost to function and form of a part is a harder problem. That is what we mean by cost accounting.

1.3.2 Asynchronous (dynamic), inconsistent, incomplete design

Design is the process of specifying a solution. Until the design is complete, one cannot be certain that it is valid or good. In general, it is not clear if the validity or the "goodness" of the design can be judged even after it is complete. Therefore the design environment must allow asynchronous (dynamic), inconsistent, incomplete design. This is an essential difficulty that the database schemas do not address, therefore databases are not able to support a complete design environment. It is not surprising that attempts to develop software environments for design using databases have not been successful.

1.3.3 Heterogeneous and distributed

Design is heterogeneous in that it calls upon many skills. A dress has been estimated to pass through approximately 60 job shops in the 52 weeks it requires to evolve from design to the rack in the department store. None of these shops know much about the internal activities and processes of any of the others. Each shop has an internal and external role, a private and public world. Software for design would have to support these job shops and their interactions and the evolution of the design as it flows through a sequence of job shops.

1.3.4 Negotiated

The fact design is negotiated implies that designs must be inconsistent, asynchronous, incomplete. If they were not, there would be nothing to negotiate. Negotiation is needed to resolve conflicts between multiple views.

1.3.5 Version management

The evolution of a design could be managed and tracked by a version control system. Software engineering has developed CASE tools for version control. Similar functionality could be provided for archiving design versions. These systems allow a private design to evolve in the private world of a designer. When the designer is ready, his design changes can be released. This is equivalent to the real world process of signing off a design and handing it over to one's supervisor. At that point the designer has relinquished control and the design is seen by other designers who can make decisions based on this version. After a version of the design is released, this version cannot be modified in any way. Any modification is a new version. Version control involves time management and requires temporal representations and reasoning. Such a version control system would journalize and archive the design process, which provides a basis for reviewing the design process and producing variations on the design. Ultimately it would allow the design process to be compiled.

1.3.6 Multiple levels of abstraction

This is a technique for managing complexity and design efficiency. It is support for the principle of least commitment. The sequence of starting the design on the back of an envelope and working towards detailed drawings and possible full scale prototypes is an example of multiple levels of abstraction.

1.3.7 Function

Representation for function is needed. Since the function denotes the intent of the design, it would also require building a knowledge of the domain to provide context. To

represent the function of a chair, one must represent the domain of the chair. For example, one must be able to analyze the response to a person sitting on the chair, supporting the person who is sitting on the chair, the dynamics of moving the chair, the scratching of the floor, the room and the purpose it is used for, the knowledge of esthetics and what it means to be beautiful. General representation of the "function" is unlikely. However, for specific problems it is possible to define or parameterize the function (or a design metric) from a certain point of view.

1.3.8 Language of communication and contract

Because the different parties involved in the design do not share a common language or conceptual model, communication between them is limited. Customers do not know in detail how a foundry manufactures a casting, but they have a procedure to communicate their request. They may be unaware of all possible casting defects but they have procedures to control and test quality. The information in a contract must be self sufficient. A foundry should not require information about the internal operations of its customer to know what is required and conveyed by the contract (their engineering practice, databases, notations, particular software they use).

Establishing a language of communication and contract is critical to their cooperation. This language is learned. It develops as they learn to do business together. Integration of software is similar. If two software programs were to communicate, they must either have a common language, or they must adhere to an overall standard for exchanging information, in which case they would each have to develop their own understanding and conceptual model of the standard protocol.

1.4 A portrait of the thesis

The discussions in this thesis are developed almost independently from four different viewpoints: 1) The casting designer's viewpoint. 2) Mathematics and basic physics of casting analysis. 3) Design theory and user interface. 4) Computer science and software engineering. The issues that are tied to casting design and analysis are developed from the point of view of foundrymen and provided with examples in a chapter. The mathematics and physics of casting analysis form the contents of a separate chapter. The emphasis to aid design is present throughout the thesis. However, the highest concentration of design theoretical issues are in the chapter on a virtual foundry. A formal system for modelling interactions and software engineering issues is discussed and developed with the agent model.

1.4.1 Casting design⁷

The discussions of casting design will serve a double purpose: They will demonstrate the use of numerical simulations to support casting design; and, they will provide the examples and applications that will be used to describe the underlying physics, mathematics and the software architecture. The ultimate product of this part of the thesis is a software environment for casting design and analysis, which was used to perform all of the numerical experiments.

Is it at all possible to provide a software environment that increases the predictive capability of the casting designer in a way that is attractive to the foundry industry? It has not been the lack of the physics, software tools or computer hardware that has

---

⁷The term casting design is primarily used in the context of designing the casting process to produce the desired configuration and properties. Of course, the design of the casting process affects the design of the component (the casting), i.e., fillets, etc.

prevented the development of design environments for foundries. The deficiency⁸ of most numerical simulation systems has been the weakness of their integration with the designer. The research issues in integrating the software for the design and the production of a casting are largely unknown. The intent of such a capability should be to supplement, not replace, the common sense and creative talent of the designer.

With the current state of the foundry technology, there is little or very often no analysis performed to assist in the design of a casting. Without analysis, it is extremely difficult to learn from previous designs and assess the impact on quality of design decisions. How can one use an analysis capability? A casting design methodology that is integrated with the software environment is developed to analyze cases involving different castings to demonstrate different aspects of the numerical analysis as a part of the casting design process.

Real castings naturally have inclusions, voids and other forms of imperfections that degrade quality. The acceptable quality of a casting not only depends on the form or existence of these imperfections but also on the relations of these imperfections to the function of the cast part. A general classification of the origins of defects was suggested by Beeley [8]: the casting design, the method of casting and the application of the process of manufacturing. The design is central to quality. Although it is possible to optimize the manufacturing process at the foundry floor, flaws in the design cannot be fixed at this late stage of product development. The casting designer's priorities in achieving quality are:

- *Avoid defects*
  - *Filling defects arising in pouring* - Cold laps, cold shuts, lack of fill, excessive turbulence and Reynolds numbers.
  - *Inclusions* - Slag, sand erosion, oxides
  - *Gas defects* - Negative pressures, ingestion of air into castings

---

⁸The validation of software in real design environments is an equally important task.

- *Shrinkage and porosity defects* - Defects due to volumetric contractions in solidification
- *Cold cracks* - Defects due to volumetric contractions in solid state.
- *Dimensional errors* - Deformations
- *Provide adequate metallurgical properties* - This is primarily achieving a sufficiently fine grain size through nucleation (add nucleating agents) and a high enough cooling rate, controlling micro/macro segregation.
- *Improve yield, reduce cost* - Minimizing the ratio of rigging volume to casting volume to reduce cost. Design the gating to reduce finishing costs.

Frequently, the first objective is to produce the casting within acceptable defect levels. Once the desired quality is obtained, the focus shifts to improving yield without compromising the quality of the casting. The design activities in a foundry revolve around the following themes to achieve the desired behavior: Gating design is the construction of a hydraulic circuit to deliver the molten metal to the mold cavity. Riser design adds reservoirs of feed metal so that shrinkage cavities can be minimized and made to occur in places where they can be tolerated. To achieve the designed metallurgical properties, process parameters such as the pouring temperature and rate, additives to control under/super cooling and nucleation rates, and chills and exothermics to control the cooling rate are adjusted.

A casting design environment that provides the designer with the appropriate tools and decision support mechanisms is very desirable. One solution is a scientifically based simulation of the casting process, a "virtual manufacturing" system, where experiments can be carried out in software. A prototypical casting software was developed as a part of this thesis. Bringing such software with a scientific basis to the foundry involves capturing an intricate web of interactions among many physical and social phenomena, and interpretations of the underlying physics. Copp & Zanella [9], observe:

"Engineers have come to use much of the methodology of science, but the nature of the questions differ from science as engineers seek better machines, structures, systems,

chemicals or processes. Problems of scale become important to the engineer as do problems of cost and other aspects that influence utility. While general scientific theories set the broad constraints, final designs must be informed by more detailed considerations than general descriptions of nature can provide."

One can only hope to capture a portion of this web for casting design. This software will give insight into the analysis of a small set of physical events that occur in castings and how these analyses could be used to improve quality; thermal and solidification analysis to facilitate design decisions dealing with shrinkage defects, metallurgical properties, location of hot spots and cavities, micro-porosity, micro/macro segregation, microstructure evolution, mechanical properties, etc. Flow predictions improve understanding of filling, volumetric contractions, the inclusions entrained in the fluid stream, filtering impurities and slag, lack of fill, cold shuts and cold laps, air pockets, pressure and velocity profiles, flow regimes and turbulence.

1.4.2 Virtual foundry

Learning for design with software is an alternative to learning with experiments. Software can be based on mathematical models of casting physics, and can provide formal methods for the study of castings. Experiment and software provide diverse conceptual models of the casting process. The experimental part of the spectrum is well developed: Design rules and practices based on experimental findings are widely documented. However, contributions of experimental methods to casting quality have their limits.

Hypothesis: The next order of magnitude gain in quality and cost-effectiveness in the casting industry will critically depend on the availability of software with a science based analysis capability.

The goal of the software system is to predict and analyze the behavior of a design to enable the designer to improve the quality of the casting. This thesis attempts to

satisfy the essential requirements of a software environment for casting design. The concept of an intelligent design assistant is central to our strategy. The intent is that the design assistant assumes responsibility for routine details of the design process allowing the designer to concentrate on more productive and creative tasks; acting as a junior partner of the designer in the design process.

The aid provided by the design assistant differs from the aid a traditional off-the-shelf analysis program would provide. The assistant and the designer each have their own views, beliefs and values about the casting analysis. They will autonomously make decisions in the process according to their view of the world. Compared to the engineer's high level decisions about design, the decisions made by the design assistant are very low level.

The interface between the designer and the intelligent design assistant is critical to the success of their collaboration. The environment where the designer and the assistant interact must be highly expressive and bi-directional. The system simulates a foundry for the designers: they can experiment with the effects of their design decisions while the system provides feedback in the form of analysis based on mathematical models. This environment constitutes the "*virtual*" foundry.

1.4.3 Basic physics and mathematical foundations of casting analysis

A foundation for the physical background, the underlying mathematics and their discretized forms for a numerical treatment relevant to the analysis of castings will be developed. A compact summary of the relevant physics, mathematics and the numerical methods is provided. The goal of this chapter is neither to provide an exhaustive account of all physics pertinent to casting analysis, nor to study the resulting equations with deep mathematical rigor. The discussions provide references to mathematical and numerical

methods. The presentation of the models is reinforced by examples. In some cases, the numerical methods are extended to provide a concrete example of numerical analysis for practical casting analysis and design.

The basic physics of castings include 3D multi-phase fluid-flow and free-boundary evolution; flow through porous media; heat transfer and solidification; and porosity development. The three fundamental nonlinear partial differential equations in the mathematical model are fluid flow, free surface evolution and the energy equation.

Viscous, incompressible fluid flow is modelled with the Navier-Stokes equation. The velocities in the molten metal are low enough to treat the air as incompressible and the details in the flow of air in the mold cavity are not crucial to the foundry engineer. Efficient methods ([10], [11], [12]) are available to solve this problem. The following time-dependent nonlinear system is considered:

$$\begin{aligned}\rho \dot{u} + u \bullet \nabla u - \mu \Delta u + \nabla p &= f \\ \operatorname{div}(u) &= 0\end{aligned}$$

where u is the velocity, p is the pressure, ρ is the density, μ is the dynamic viscosity. The density and viscosity vary in time and space. f is the body force. The differential equation is not completely defined without the proper boundary conditions and initial values, but they are ignored for the moment to avoid detail. Fluid flow in sand can be modelled using a simplified model, such as potential flow (D'Arcy model), or with more sophisticated continuum equations for flow through porous media such as those of Beckermann et. al. [13].

The free surface problem is modelled by the evolution of a density field for the fraction of liquid. The relevant hyperbolic partial differential equation is:

$$\psi_t + u \bullet \nabla \psi = 0$$

where u is the velocity, ψ is the fraction of liquid (VOF) function.

The thermal-transport problem with solidification kinetics is modeled with the following energy equation with solidification models implemented in a Eulerian reference frame.

$$\dot{h} + \kappa \Delta T + C_p(u \cdot \nabla T) - L\dot{f}_s = s$$

where u is the velocity, h is the specific enthalpy density, C_p is the specific heat, κ is the thermal conductivity, L is the latent heat of fusion, f_s is fraction of solid per unit volume. The specific heat, latent heat and thermal conductivity vary in space and time. Solidification modelling is an active research area; models of varying detail and sophistication are available for a variety of alloys. ([14], [15], [16]).

Fluid flow due to shrinkage may be used to predict porosity. Pores are more likely to form at the liquid-solid front where the highest negative pressures and highest levels of dissolved gases exist. If volumetric shrinkage during solidification due to the change in density between liquid and solid metal is $\alpha = \frac{\hat{\rho}_l - \hat{\rho}_s}{\bar{\rho}}$, then the volumetric strain rate due to shrinkage will be $\alpha \dot{f}_s$, where f_s is the fraction of solid as above. The rate of formation of pores $\frac{\rho_l}{\bar{\rho}} \dot{\epsilon}_{pore}$ also contributes to the volumetric strain rate. This volumetric strain rate, which equals the divergence of the velocity, modifies the incompressibility condition in the Navier-Stokes equations, and can be added to the governing equations as a distributed sink or source term.

Coupling equations, such as those given above, can raise subtle issues. For example with increasing fluid velocity, the effect of the diffusive term can become negligible in the energy equation, and it changes from a parabolic equation to a hyperbolic equation. The numerical solution techniques must be capable of dealing with the changing form of these equations.

1.4.4 Agents

The physics of casting, even in the simplified form presented in this thesis, impose challenges in several disguises; the interactions of the physical phenomena are complex, and the complexity will increase as more detailed modelling of physics is needed. Events occur at different length scales⁹, ranging from atomic size to the scale of the geometry of the casting, and at different time scales, ranging from fractions of seconds to hours. The system is non-linear and dynamic. The nature of the physics, the associated models and their solutions evolve in time. Can these models be formalized, generalized and applied to other problems? Can they be managed? Can they be solved? The proposed solutions to these problems could make it easier to deal with more complex physics problems. On the other hand, if we do not recognize their limitations, they will limit the success of the overall system even if the sub problems of the physics were successfully solved. It will be argued that the design of the software is at least as important as the correct modelling of the physical problem. Not surprisingly, these problems are not specific to basic physics; the general problem of interactions and integration have similar character.

Hypothesis: There are multiple agents with different viewpoints. Agents delegate certain responsibilities to other agents as they perform their own tasks, exchanging information in the process.

As a design evolves, the schema and the participants will vary. The agents and their interactions form the components (nodes) of a graph based representation. Communication paths are the edges of the graph. In this graph, there are multiple viewpoints; each agent has its own perception of reality and truth. There are multiple abstraction levels; the parameterization, goals and functions are hierarchical and are different at different levels of design detail. Each agent seeks to separate its internal

---

⁹See Glimm [17], for an excellent survey of these issues.

representation and behavior from the external interactions with other agents. An agent may encapsulate a lower level graph to hide complexity.

What are the minimal requirements for a formal environment that deals with the problem of interactions? It should be capable of representing the graph of interactions. There should be capabilities to deal with multiple viewpoints and multiple levels of abstractions. The function and the form of components of the graph should be clearly specified, separated and structured in a manner that would make it possible for them to cooperate in a software environment. A heterogeneous environment of agents, where there is freedom of internal representations and implementations is crucial. The formal system should allow incomplete and inconsistent specifications of agents. There should be standardized structures for protocols of communications among the agents.

The essential difficulty is properly decomposing a coupled problem and formalizing the subproblems and their interactions. A meta-programming approach will be described where solvers for coupled systems of PDEs can be expressed as programs of atomic operations. In this case, a typical primitive agent is a software component modeling an aspect of the physics, such as the Navier-Stokes equation. These programs not only describe the interactions of the subproblems, but also can be designed to adapt¹⁰ to the requirements of a dynamic problem during execution. Ultimately, this dynamic network of programs should behave as a single program with continuous behavior. A decomposition of the structure into general coupled systems of equations is fundamental to this thesis. The model for the coupled problem is a set of abstract agents (subproblems) that cooperate by sending messages over an abstract network (the coupled problem). New agents may be introduced into this network, or existing agents can be removed. An agent can be replaced. New connections may be established or existing

---

¹⁰The system is dynamic. However, the questions such as how it evolves, and how its activities are controlled do not have clear answers. In some cases, it is a human expert who makes the decisions. In other cases, models based on some parameters, such as the Reynolds or Peclet numbers, can be used to control the evolution of the system.

connections can be removed. The structure of the network is dynamic, its function is continuous. The mechanisms for adding and removing agents to solve new problems encourages software reuse. Agents can be fault tolerant. An agent which fails can be re-instantiated, replaced or modified dynamically. Agents are modular. Modularity reduces complexity and provides the capability to deal separately with internal and external behavior. Instrumenting the natural chunking of complex problems provides a new technology and new understanding.

Agents can represent a formal system and a natural decomposition of the problem into subproblems to deal with large complex phenomena. Formalized relations among the agents allow programmable connections and configurations of clusters of agents. As the connections and configurations of the agents change in a controlled manner, the program represented by this network also changes.

The agent model provides support for a framework for distributed and parallel processing naturally since the concept of agents are based on similar principles.

1.4.4.1 Agents for analysis

When approached at the level of agents, the difficulties identified in the section that deals with the basic physics of castings share common problems with the overall design process. Multiple physical phenomena interact and the graph associated with it may change in time. Each physical phenomenon can be interpreted as a viewpoint. The multiple length and time scales are similar to multiple abstraction levels problem. A problem is parameterized and understood in a quite different manner at each abstraction level, yet these levels are intricately entwined. They exchange information, i.e., data flows.

Obviously, this is a very restricted view of casting physics and the related models. Many important phenomena are not included. Despite its narrow focus, this capability would still be a significant advance for the scientific understanding of the casting process. As more detailed knowledge of physics becomes available, it should be added to the collection of scientifically based design aids available to the engineer. The framework based on the agent model supports this extendibility.

Do agents solve the problem of interactions? The answer is definitely no. However, in our belief it is a forward step since agents identify and attempt to remove many of the accidental difficulties of design and integration. By definition, they can not address the essential difficulties. In fact, they will introduce new problems and new accidental difficulties.

1.5 A road map

The reader of this thesis is likely an expert in casting, theory of design (for mechanical engineering), applied mathematics or computer science. As an expert in one of these areas, the reader is encouraged to primarily read the chapters relevant to their expertise. It is inherent in the design of a complex system that no one person is familiar with all aspects. The author does not claim to be an expert in these fields, but he claims to have a sufficient understanding of the areas.

In the remainder of this thesis, there are chapters on casting design, continuum mechanics of the basic physics and numerical methods, agents and their design and analysis implications. Each chapter will deal with a topic in its own context, and whenever possible, will use examples from castings. If the reader's primary interest is casting, then chapters 2 and 3 are most important. If the reader's primary interest is design theory and methodology for mechanical engineering, then chapter 3 and parts of chapter 5

are of most interest. If the reader's primary interest is computational mechanics or mathematics then chapter 4 would be of most interest. If the reader's primary interest is computer science, then chapters 5 and possibly chapter 3 would be of most interest. Chapter 6 provides examples for chapter 5 on agents. The reader can read the two chapters nonlinearly, referring to the examples for concrete demonstrations or to build an intuitive sense before reading abstract concepts. Chapter 7 provides a synopsis of the thesis and summary of contributions. The readers are advised to read those chapters that best suit their viewpoint and to read the chapters in the order that suit their interests. To a large degree, each chapter can be read independently.

*"An invaluable general rule which I recommend
to all those studying running and gating systems
is 'If in doubt visualize water' ..."*
J. Campbell, Castings, 1991

Chapter 2 : Designing a casting

2.1 Can we do better than just guessing?

Providing the foundry industry with the technology for "virtual manufacturing" can increase their predictive capability and can in time transform the art of designing a casting into a science. The research issues in integrating the software for the design and the production of a casting are largely unknown. The intent of such a capability should be to supplement, not replace, the common sense and creative talent of the designer. The deficiency of most numerical simulation systems has been the weakness of their integration with the designer.

Is it at all possible to provide a software environment that increases the predictive capability of the casting designer in a way that is attractive to the foundry industry? The hypothesis of this thesis is that it is possible. The rest of this chapter attempts to illustrate the software environment with examples.

This chapter serves a double purpose: By examples, it demonstrates the use of numerical simulations to support casting design; and, it provides the examples and applications, that will be used to describe the underlying physics, mathematics and the software architecture in subsequent chapters. A casting analysis environment that was developed for this research was used to perform the numerical experiments.

2.2 Prelude: Tradition of casting design

Foundrymen have long sought a methodology for the design of castings, ([18], [19], [20]). Generally design starts with correctly orienting the part in the mold and "properly" feeding the metal to the cast part. In orienting the part, designers try to eliminate or reduce the number of cores (isolated shapes inserted in the mold to cast internal cavities, etc.). They seek an optimum use of the space available in the mold box. Designing the metal feeding system is known as the gating or rigging design, (see Figure 2.1).

Figure 2.1: Typical rigging designs

Rigging is the hydraulic circuit which delivers the molten metal to the cavity of the cast part. The design of the gating must satisfy a wide range of constraints such as preventing damage to the mold and to the quality of the liquid metal during the filling of the cavity and utilizing the gating to accommodate fluid flow due to volume changes in the cooling process. Once the gating can satisfactorily deliver metal to all sections of the casting, the gating is optimized to increase yield, and to prevent unacceptable defects.

An important part of the design is risering, i.e. designing the risers. Although it is sometimes recognized as a part of the gating design, the function of a riser is different. The primary function of the gating is to deliver liquid metal to all parts of the casting and to screen slag (inclusions, oxides, etc.). Risers, on the other hand, are designed to prevent and reduce shrinkage and porosity defects. Risers and the gating system are removed during clean up, they are not salable items. Reducing the amount of metal used in gating and risering system can significantly increase the casting yield and reduce the overall cost of the casting.

There is no universally accepted system of gating principles. The interactions among design variables are complex due to the wide variety of design constraints and physical events that take place. With the current state of the foundry technology, there is little or very often no analysis performed to assist in the design of a casting. Without analysis, it is extremely difficult to learn from previous designs and assess the impact on quality of design decisions. To a large extent, every new design involves a completely new and expensive learning process.

2.2.1 Priorities

The defects that must be controlled to achieve quality can be categorized as:

- *Filling defects:* Cold laps, cold shuts, excessive turbulence.

- *Inclusion defects:* Slag, sand erosion, oxides.
- *Gas defects:* Negative pressures, ingestion of air into castings, surface gas defects.
- *Shrinkage/Porosity defects:* Defects due to volumetric contractions in solidification.
- *Cold cracks:* Defects due to volumetric contractions in the solid state.
- *Dimensional errors:* Deformations.

If the gating and riser design is satisfactory, the casting designer's attention will be shifted to improving and optimizing the design for metallurgical quality and reduced cost. Providing adequate metallurgical properties is primarily achieved by controlling the grain size and nucleation rate, (adding nucleating agents and controlling the cooling rate with chills), and controlling micro/macro segregation. Improving yield to reduce cost is achieved by minimizing the ratio of rigging volume to casting volume to reduce scrap metal. Designing the gating system to reduce the cost of removing the rigging from the casting is also a part of the optimization effort.

2.2.2 Methods

Foundrymen have produced sound castings by relying on the knowledge and the intuition accumulated over generations. The methods to predict behavior and the quality of the casting have evolved over 5000 years. There are published guidelines of gating design principles ([18], [19], [21]), which provide case studies and simple physics to aid the designer. Good and bad gating practices are documented. There are some scientific studies of why some designs are better and of some events that take place during casting, (Campbell [22]). Despite the available knowledge, the capability of predicting or even observing events in the closed mold box is largely unavailable to the designer. Current predictive technology is remarkably similar to that of the early times and relies heavily on design experiments, i.e., design via trial and error with sample castings.

2.2.3 Are there solutions?

A scientific basis and tools to integrate the basis into the designer's environment would significantly advance the predictive capability of the foundry industry. Existing software technology and understanding of physics of casting has the capabilities needed to create such a design environment. However, it has not been the lack of the physics, software tools or computer hardware that has prevented the development of design environments for foundries. Why have proper analysis tools not been offered to casting designers? And, why have the tools that are available today not been adopted by the designers?

2.3 Traditional analysis software

There are many physical phenomena in producing a casting. Realistic modeling of the underlying physics of casting events is fundamental to providing a scientific basis for casting design. It is possible to deal with some of these phenomena with the current level of understanding; solidification modeling to study the macroscopic thermal evolution in the casting is well developed, ([14], [16]). Models to study the formation of pores, while still in early stages of research, are valuable in identifying and solving risering and gating problems, ([23]). Fluid flow analysis is one of the more advanced areas of numerical modeling; it can address design concerns such as fluidity (cold shuts and remelting), inadequate filling and good flow regimes require careful study of the fluid flow in castings, ([24], [13]).

2.3.1 Limitations

The challenge in providing a predictive capability to the casting designer is not the lack of good mathematical models or the understanding of the basic physics. Good models

have been available during most of this century. Software for thermal, fluid flow, stress, etc. analysis is readily available. In spite of their availability, the use of this technology in the foundry industry has been very rare and limited.

There are fundamental problems: The analysis programs are complex and gaining proficiency is dependent on the users' understanding of the mathematics and the models used. Casting designers are rarely experts in heat transfer, Navier-Stokes equations or numerical algorithms used in their solutions. Translating casting problems to the form needed as input to run these programs is not an easy task and is very often prohibitively expensive for industry. These programs do not use a language familiar to the casting designers. Attempts to integrate them into the working environment of the designer have not been successful to date.

A more technical difficulty is that, like most real problems, casting physics involves the interactions of a multitude of physical phenomena. It is very rare that a packaged analysis system addresses these interactions; for example a traditional fluid flow analysis program is not easily integrated with a stress analysis program to solve for stresses in the solid regions of a cooling casting. This is not a physics problem, it is mainly a software engineering problem. It is typical of the difficult software engineering issues which must be recognized and addressed.

2.4 A casting analysis environment and a methodology for using it

A casting analysis environment has been developed. The environment is modular which means that new or improved analysis capabilities can be added as they are developed. Coupled analyses involving interactions between several physical phenomena can be

performed. A geometry module is provided for rapid experimentation with the rigging design ([25], [26]), (see Figure 2.2).

Figure 2.2: A window which belongs to the geometry module displaying a gating diagram and the corresponding FEM mesh

Parameterized features such as sprues, gates and risers can be instantiated and attached interactively to a casting and their dimensions can be manipulated. A mesh for the rigging is automatically created from the designer's specifications. The current analysis capability consists of Finite Element Method (FEM) based thermal analysis, solidification analysis, fluid flow analysis and free surface modules. Visualization software is included for presentation of the results. The designer is provided with a graphical user interface. The process of translating a casting problem into a problem that can be solved by FEM analysis is completely hidden from the user.

Although the benefits of having an analysis capability are clear the task of integrating the analysis into the design of a casting has not been addressed. As the designer goes through different phases of design, each phase has unique needs and reasons for analysis. The comprehension of the results also varies; the interpretations are more qualitative at the early stages of design. The quantitative outcomes of the analysis become more important as more details of the geometry and the process are specified. How can one use an analysis capability? There are probably as many answers as there

are designs. Instead of attacking the general question, adopting a style of proof of existence by example style, cases involving different castings will be used to demonstrate different aspects of the numerical analysis as a part of the casting design process.

This casting analysis environment is not an expert system that would propose alternative or new designs. It is a software to experiment and analyze alternative designs. The environment is intended as an "apprentice" designer who can help the casting designer by assuming responsibility for doing analysis. A prescriptive methodology for

Figure 2.3: Integrating analysis into casting design

using the analysis environment is adopted. The methodology is limited by the current analysis capability, however, it is obvious that extending the analysis capability would only make it more useful. This casting analysis environment attempts to incorporate the analysis capability into traditional phases of the casting design. This methodology is

highlighted in Figure 2.3. A typical design and analysis process could consist of the following stages:

- **Orient the part, identify solidification patterns:** It is generally accepted that the rigging should be designed to promote progressive solidification, and that enough metal should be supplied to all sections of the casting as it solidifies. One rule is to locate the gates in such a way that the solidification starts away from them and propagates towards the gates allowing them to stay open and feed the casting as long as possible during cooling. This implies that thermal and solidification analysis on the cast part should be carried out fairly early during the design, even before the rigging and risers are attached. The first task is to identify and to assess the suitability of gating and risering locations.
- **Select gate and riser locations:** After a gating and feeding setup is determined a more detailed analysis of the casting system is necessary: Thermal and shrinkage analysis could validate whether or not the rigging is functioning as designed. At this stage the rigging designer is also concerned with the effects of the rigging on the metallurgical quality. The relative cooling rates of different parts of the casting should be studied. To control cooling rates, chills may be inserted into the mold, or chemical agents may also be added to the liquid metal. If possible, it can be desirable to use chills to reduce the number of risers and gates.
- **Adjust the rigging for better fluid flow:** As consistent patterns emerge in the rigging design, the runners and gates can be reviewed and relocated to deliver the liquid metal to the casting with the least amount of damage to the mold and metal. Gating ratios are adjusted to control flow rates, gates are shaped and oriented with respect to the runner to stop the early metal from getting into the casting and eroding the mold, and vents are added to ease the flow and remove trapped air pockets. Tapers and fillets in the rigging may be reviewed for smoother flow. Fluid flow analysis and filling simulation will provide insight into these events. Problems related to lack of fill and cold shuts should also be analyzed. Coupled thermal and flow analysis also provides information on how energy is transferred during the filling.
- **Investigate the effects of the mold:** The mold has significant effects on the cooling rates and the metallurgical quality of castings. Final details of the rigging should be analyzed including the mold. Thermal and solidification analysis and modeling of the contact between the mold and the casting will reveal effects of mold, superheating, chills and insulators.

The intent in suggesting these stages was to establish a context to present the predictive capability to the designer. These design stages are not expected to be unique

or complete. The state of the art in predictive technology has a significant effect on the content and structure of the design stages and this methodology is expected to evolve as new knowledge and capability is added to a design environment. The design stages can overlap. Different design phases may be repeated many times. However, the cost of analysis and design iterations increase as the design becomes more detailed and more accurate parameters are needed. Faster, less accurate and simpler analysis tools are justified in the earlier iterations. However, during the later stages of design, more sophisticated and accurate analysis is necessary.

2.4.1 The study of a casting during the various stages of design

The design of a wheel casting is studied. The wheel was cast from a magnesium alloy using sand casting technology. The analysis was done in parallel with the foundry as they performed their own research with sample castings. Some designs were not cast by the foundry but investigated only with numerical simulations. Other analyses were guided by data and designs from the foundry and assessed qualitatively with respect to its findings.

Figure 2.4: The wheel

The design starts with reasoning with the geometrical considerations: orientation of the part, location of the parting surface, etc. The analysis requires that a mesh is provided. Mesh creation is currently the most expensive and time consuming part of numerical analysis. The development time for meshes ranges from weeks to months depending on the complexity of the part. Orienting the casting by aligning the vertical axis of the mold box with the center axis of the wheel eliminates the

use of cores, and simplifies the pattern and the removal of the casting from the mold. There are studies on automated reasoning about the geometry to aid the selection of parting planes and natural flow channels ([27], [28]); but they will not be discussed. The results of thermal and solidification analyses are important at the early stages. The orientation of the part and the locations of parting surfaces will affect how and where the runners, gates and risers are placed. Design errors at this stage may make the difference between success and failure of the casting and the foundry.

2.4.1.1 Initial assessment of the casting without the rigging

The thermal and solidification analysis of a section of the wheel including one of the spokes with no rigging attached shows the thermal behavior of the geometry, (see Figure 2.5). Because a rigging is not attached, the results are largely qualitative. The analysis with a mold would change the cooling profile, but not using a mold is expected to provide a good trial design and often does not alter the qualitative outcome of the analysis for most cases. The similarities between the 2D-axisymmetric results (Figure 2.5(a)), and the 3D slice from the wheel (Figure 2.5(b)) demonstrate that there can be effective shortcuts to further simplify analysis early in the design process.

The solidification time map indicates that the spoke near the center hub is the last place to solidify, (locations 2 & 3 on Figure 2.5(a) {ii}). Solidification proceeds from the walls of the wheel towards the hub. Analysis of fluid flow due to volumetric contractions, (Figure 2.5(a) {iii}), shows flow from the center of the wheel towards the wall. The spoke and the center hub are potential locations for feeding problems because they are the last regions to solidify in the casting. The solidification time map also suggests that the top and bottom edges of the rim have isolated liquid metal that suggest of a higher risk of defects (points 6 and 8), however, this problem is more apparent in the spoke near the center hub (point 3).

Figure 2.5(a): Two dimensional axisymmetric analysis of the wheel casting.
 (i) Temperature and fraction of solid distributions at two instants of time (ii) The temperature versus time plot for several points in the casting (iii) Fluid flow induced by the volumetric changes due to solidification

Figure 2.5(b): The temperature and fraction of solid contours of the sixty degree slice from the wheel

A series of data sets can be extracted from the early analysis to guide the rigging design. Shrinkage induced fluid flow and isolated hot spots are the type of information that suggest possible locations for risering. The information required to alter shrinkage and solidification patterns can also be deduced from thermal and solidification maps, (i.e., the chill locations). Hot spots on the walls of the casting are suitable places to attach risers and gates because they will stay liquid for the longest periods of time.

2.4.1.2 The design and development of the rigging

The initial thermal and the solidification analysis of the casting guides the design of a trial rigging by providing information on critical regions in the casting that must be fed to avoid defects. Usually the rigging is designed to drive solidification from the body of the casting towards the rigging. The gates and risers are placed and sized in a manner to keep them from solidifying before the casting. Usually, there will be more than one design. Alternative designs should be tested and their relative advantages compared. Eventually the final form of the design will appear as a blend and compromise of several rigging configurations.

Potential problem areas were identified in the hub and the spoke. A series of alternative rigging designs were analyzed: In the first design, a slot gate aligned with the spoke is attached. Although this type of gating significantly modifies the thermal behavior on the walls of the wheel closer to the gate, the solidification was still driven from the walls towards the center of the wheel, (see Figure 2.6(a)). Increasing the width of the gate near the spoke improves the fluidity of the gate but it is not sufficient to feed the casting because the gate still solidifies before the casting or the vertical gate. In the next design, a riser is attached to the center hub to compensate for the lack of liquid metal around that region, Figure 2.6(b). In the third alternative, the gate is attached between the

Figure 2.6: Thermal and solidification analysis for different rigging configurations of a sixty degree section of the wheel (a) The slot gate with uniform thickness is located behind the spoke (b) The slot gate with a bulging section is located near the spoke and a riser is added to the hub (c) The slot gate is moved between the spokes (d) Temperature wrt. time plot at a point near the spoke where the gate is attached to the wheel

spokes, Figure 2.6(c). Investigating the temperature versus time evolution at a point where the gate was attached to the casting for these three cases indicated that increasing the width of the gate had the effect of keeping the gate from freezing shut early but changing the location of the gate with respect to the spokes did not significantly alter the time at which the gate freezes shut, Figure 2.6(d). This may indicate that the designer is free to place gates around the casting with respect to the spokes to experiment with different runner schemes, chills for faster cooling, etc.

The risers (feeders) are designed to remain liquid longer than the casting, and to provide the necessary liquid metal to feed the solidifying parts of the casting where avoiding defects is important. It is also desired that the hot spots be located in the risers, not in the casting. In Figure 2.7, the solidification and the flow induced by volumetric changes in the wheel are analyzed: The riser remains at a higher temperature. The flow induced by the shrinkage is directed from the riser towards the casting and from the walls closer to the spoke towards the edges of the rim. The isolated regions where the flow is not supplemented with liquid metal are candidates for porosity and void defects.

Analysis of the rigging system without considering the effects of filling and mold obviously introduces errors; overheated mold, transfer of thermal energy with fluid flow, solidification at the walls of the casting during filling are some of the important effects that are ignored. However, these simplifications can be justified early in the process when the rigging is still being designed to avoid defects such as shrinkage, in order to try and experiment with more designs rapidly. It is certainly true that a more detailed analysis including the effects of the mold and filling will predict more realistic behavior of the casting. Such analysis should be done at later stages in the design process.

Figure 2.7: The flow induced by the volumetric changes due to solidification. The velocity vectors and the regions where there is an outflow of fluid are illustrated

2.4.1.3 Analysis during filling

The defects incurred during the filling of a casting were itemized previously. There are several factors that cause these defects. The commonly encountered ones are: surface turbulence in filling, improper trapping and screening of slag and inclusions, unexpected fluidity problems such as cold shuts or cold laps (the freezing and remelting of liquid metal).

Analysis focusing on the effects of filling may also be approached at different levels of sophistication. The simple case is the fluid flow analysis without considering the effects of solidification or thermal evolution. This analysis is a valid one when the filling rates are high and the cavity fills before the metal solidifies. The velocity distribution, trapped air pockets, filling sequence, surface behavior, etc. are some of the typical results that may be assessed from this analysis. The primary design response would be to adjust tapers, fillets, chokes, vents, etc., to remedy these problems. In Figure 2.8, the filling of a multi plate bottom-gated casting with two different runner configurations is demonstrated. The purpose of the taper in the runner is to compensate for the inertial effects of the fluid flow in the runner and allow uniform filling of the plates. Note the early filling of the plate furthest away from the sprue in the casting with the flat runner. This problem is remedied with a taper in the runner in the next design. Figure 2.9, displays the filling of a casting and the velocity profile at a section. Note the back flow from the runner extension. This behavior is undesired because the initial liquid metal is lower in quality. The designer usually attempts to contain it in the runner extension. In this case the extension fails to fulfill this objective.

Slower filling rates or lower pouring temperatures close to the melting point of the metal justify more detailed studies. The analysis described above will fail to predict the

Figure 2.8: The filling simulation of two alternative runner designs for a multiple part casting. The taper in the runner is adjusted for uniform filling of the three plates

Figure 2.9: The filling simulation for a two part casting

Figure 2.10: The thermal-flow analysis of a cold shut problem with cylindrical runner cross-section. The evolution of temperature, fraction of solid and velocity fields are presented at earlier and later stages of filling

interactions of fluid flow, solidification and thermal behavior of the metal. According to Campbell [22], the defects due to residual flow patterns and solidification during filling can be mistaken as shrinkage problems, and often futile attempts are made to fix them with feeders, etc., but they can be cured by modifying the locations of the ingates, their size or the filling rate. Cold shuts and laps are another type of problems due to lowered fluidity of the metal that occurs when the filling rate or pouring temperature is low. Coupling the fluid flow analysis with thermal analysis and solidification predicts the development of fraction of solid, the evolution of velocity and temperature field, which are demonstrated on a longitudinal cross-section of a cylindrical piece, Figure 2.10.

2.5 Remarks

The capability to analyze a casting and its behavior, even when limited, is an important step forward. At this stage of development, the analysis capability is not sufficiently advanced to replace the trial and error with sample castings. The primary difficulties lie in managing, reasoning and discretizing the geometry, i.e., automatic meshing, and doing analysis within a period of time acceptable to the industry. The analysis requirements and methods are different at various stages of design and there should be a suite of analysis tools which can adapt to these needs.

Most physical phenomena described in this chapter involve interaction among two or more complex events. To be useful, the tools provided to industry must be capable of coupled analysis.

When a sufficiently advanced capability is developed, it is expected that foundries will have to develop methods and strategies to integrate this capability into their design and production process.

Chapter 3: Virtual Foundry - Software for casting design

Conceptual models abstract the essential details of a complex real system and assist individuals to deal with reality. A car is conceptualized very differently by the mechanic, the sales person and the driver. One constructs a conceptual model of reality through learning. Conceptual models are archived in the beliefs, values and traditions of a culture. In casting, physical experiments have been the traditional method of learning how to design. The designer's conceptual model is built from this knowledge. The difficulties arising in casting design are often consequences of flaws in conceptual models, and the time, effort and cost of learning with physical experiments. The designer is unlikely to solve a problem if the real cause (or a relatively good model) of a defect is not understood. The alchemists tried to change lead into gold for hundreds of years. Their attempts were futile because of a flawed conceptual model of the relationship between mercury and gold. Constructing deep conceptual models such as the structure of atoms poses essential difficulties.

If the conceptual model is a good one, i.e., the model is consistent with reality, the remaining complications in design are accidental. Tools, methods, and strategies are often available to deal with accidental difficulties (although, they often introduce new difficulties). The essential difficulty is understanding the real physics. Accidental difficulties arise from the methods and instruments of design and learning.

Learning for design with software is an alternative to learning with experiments. Software can be based on mathematical models of casting physics, and can provide formal methods for the study of castings. It is only natural that the conceptual models developed using software tools are different than those developed from experiment. These are two very different views of the casting reality. In Figure 3.1, the role of software and experiment in building conceptual structures is illustrated.

Figure 3.1 Role of software and experiment in building Conceptual Models

Experiment and software provide diverse conceptual models of the casting process. The evolution of temperature, pressure and velocity distribution in a casting, and filling

sequences are some of the outputs that can be easily provided with analysis software. Although, in a physical experiment thermocouples and probes can be inserted into the casting to acquire some local information on temperatures and velocity, the experiment provides a quite different type of information. In an experiment, it is not usually possible to gain direct knowledge of what happens inside the casting during the event. Post-experiment analysis and inspection can be carried out to correlate the effects of changes in process parameters to the changes observed. The results of experiment are susceptible to the interpretation and experience of the designer.

It is not our intent to argue against experiment, or to compare the advantages of software and experiment. Almost all of our knowledge of physics owes a great debt of experiment. In an ideal environment, they complement and support each other.

Experiment is usually more expensive than a simulation. On the other hand, in an adequate software environment, a simulation can be cheaper and faster. The results can be interpreted with less ambiguity. However, software may have bugs, inappropriate mathematical models and may provide incorrect answers. The software may not be friendly to the designers and may not be adequate to be used routinely. The goal of the software system is to predict and analyze the behavior of a design to enable the designer to improve the quality of the casting.

The experimental part of the spectrum is well developed: Design rules and practices based on experimental findings are widely documented. However, contributions of experimental methods to casting quality have their limits. It is a hypothesis of this work that the next order of magnitude gain in quality and cost-effectiveness in the casting industry will critically depend on the availability of software with a science based analysis capability.

It is a premise of this thesis that a software environment for casting should have the following essential properties:

- *Software for Casting* - The software is designed for casting. It has an understanding and representation of the domain; the governing natural laws, materials, methods, the language, defects and problems.
- *Software for Design and Interpretation-Experiment* - The primary goal of the software is to improve understanding of the casting. It complements experiment and other types of aids available to the designer. It provides insight into the natural laws governing the design. It facilitates design of experiments in software. It provides methods to build, classify and catalog models. It is faster and cheaper than real experiments.
- *Software for Designer* - The software is intended for use by the designer. The designer is not necessarily an expert on mathematics, physics or numerical methods. The software does not impose its own language on the designer. It provides assistance in a language that is familiar to the designer. It does not hinder the designer's productivity, but improves it.
- *Software for Analysis* - From a mathematical perspective, the analysis capability is a system of coupled partial differential equations that represent the underlying conservation laws. The software is to be capable of solving large, detailed and complicated systems of differential equations both accurately and quickly. It understands different numerical problems and can utilize alternative methods and solutions.

It is a long term project to develop such a system. An incremental development of this environment can still be very useful to a designer. The designer can compensate for the limitations of the software while benefiting from the assistance available.

This thesis attempts to satisfy the essential requirements of this system in an environment. The concept of an intelligent design assistant is central to our strategy. A prototype of such an assistant will be described. Inevitably, this implementation is not

complete. However, we are convinced that this is a significant step toward the goal of improved aids for design.

3.1 Tools versus environments

Before continuing with the discussion of the system, the distinction between tools and environments for assisting design in our context is clarified.

A tool is a passive aid. Its context and goal are solely defined by its user. It is incapable of any autonomous activity. It has no local or global intentions or goals. In this respect, a hammer is a tool. It does not know that it can be used to drive a nail. It neither has knowledge of a nail, nor does it have a goal to drive a nail into a wall. A hammer can be used to drive nails or it can be used as a paper weight. Its purpose is defined by the user. All of its activities are intentionally initiated by external agents.

An environment is quite different. It has a context and well defined interface to its external world. Its internal structure and observable behavior are clearly separated. It is capable of autonomous activity. It has local goals and intentions. It interacts with its external environment: it receives and provides information. In our terms, a taxi-cab is an environment. Its purpose is clear, it will transport its customers from one place to another as specified. It follows a route of its choice, makes local decisions to arrive at the destination. It may have local goals different from that of the customers, it may stretch the route to earn a higher fare. A flight simulator is also an environment. The models used by the flight simulator to mimic an airplane do not concern the pilot. The pilot interacts with the controls of the simulated airplane to fly the plane, while the flight simulator attempts behave like a real airplane in response to these inputs. A computer that plays chess is an environment.

3.2 The intelligent design assistant

The implementation of the intelligent design assistant is an attempt to provide an integrated¹ software environment that contains some of the properties of the software listed above.

The intelligent design assistant is a junior partner of the designer in the design process. It is "*intelligent*" because it can collaborate with the designer by making some decisions. The intent is that the design assistant assumes responsibility for routine details of the design process allowing the designer to concentrate on more productive and creative tasks.

The decisions made by the design assistant and the designers are at different levels of abstraction. In a typical casting design, such as those previously described in Chapter 2, the designer makes the critical and the creative decisions about the shape, size, orientations of the rigging and communicates them to the design assistant. The assistant translates the designer's description into a form that can be used by the analysis programs, does the analysis, reports results and any problems that may be caused by the designer's decisions. The assistant and the designer each have their own views, beliefs and values about the casting analysis. They will autonomously make decisions in the process according to their view of the world. Compared to the engineer's high level decisions about design, the decisions made by the design assistant are very low level such as the type of analysis to use (e.g., incompressible flow versus potential flow), the level of refinement in the computational grid, tolerances, solution and integration methods, etc.

---

¹ Integration is an overloaded term. Its meaning is not clear in phrases such as Computer-Integrated-Manufacturing. Here, it means a high level software environment which transparently manages a library of primitive software tools to provide cooperative solutions to a design problem.

The aid provided by the design assistant differs from the aid a traditional off-the-shelf analysis program would provide. Generally, in such programs designers must explicitly specify the type of equation to solve, control all aspects of the analysis including translation of the proposed design into initial values and boundary conditions, material properties, preparing a mesh, selecting numerical methods, tolerances, time step sizes, etc., without any help from the analysis software.

3.3 Virtual foundry

The interface between the designer and the intelligent assistant is critical to the success of their collaboration. Shared knowledge of domain specific information and a common language are fundamental to their interaction. The environment where the designer and the assistant interact must be highly expressive and bi-directional. The system simulates a foundry for the designers: they can experiment with the effects of their design decisions while the system provides feedback in the form of analysis based on mathematical models. This environment constitutes the "*virtual*" foundry.

The philosophy of the virtual foundry is similar to that of the Programmer's Apprentice, Rich [29]. The overall software is intended to be an environment where an intelligent assistant cooperates with casting designers, providing them with capability to do analysis. Figure 3.2 illustrates the interactions of the casting designer and the design assistant in the framework of the virtual foundry.

Shared domain specific knowledge captures the required semantic knowledge of the physics of the casting process, dependencies between functional requirements and specification, the design specification and the resulting artifact. Shared domain specific knowledge is the basis of a common language used to interface and facilitate the interactions between the casting designer and the assistant. The language captures

episodic and semantic information about the current geometry, rigging design, materials, type of process (sand-casting, die-casting, etc.), process parameters (pouring temperature, pouring rate, etc.).

Figure 3.2: Functional Decomposition of the "Virtual Foundry"

3.3.1 Shared knowledge

The designers maintain their own abstractions, beliefs and values about the process. The intelligent Analysis Assistant's task is to transform symbols representing these abstractions into mathematical statements embedded in formal systems that can be contracted to analysis tools. It must manage the interaction between the analysis and the designer. The designers do not need a deep understanding of the laws of nature to manipulate them. Similarly, they do not need a deep understanding of analysis and the equations solved to use analysis in design.

3.3.1.1 The interface with the designer: domain knowledge

The casting designers have their own abstractions and beliefs of the events that take place during the pouring of a casting and how their decisions affect the behavior of the process or product. These beliefs and values are first learned from a teacher (master to apprentice), based on their observations of the process and then extended by their own experience accumulated over time. Nevertheless, they are built upon a common tradition of casting design and rules.

A scientifically based simulation of the casting on the other hand, has a quite different abstraction of the casting process and evolved from a different tradition or culture. The process is viewed as a mathematical system. An approximate solution of this mathematical system is obtained after translating the real problem into a mathematical one with initial and boundary conditions and a mesh interpreting the physical problem as closely as possible.

These views differ significantly. The casting designers' abstractions, even if they were sometimes incorrect, allow them to deal with the complexity of the process and help them to produce castings. The mathematical abstractions, even if they produced the most realistic and accurate simulation of the process, may still be ineffective if they can not be understood and integrated by the designers into their routine design activity. The majority of existing analysis systems make little or no attempt to serve the needs and abstractions of the designer. The mathematical abstractions and numerical details are usually exposed in a raw form that is hostile and unacceptable for the designers. Becoming experts in the mathematics of the analysis software and the numerical methods would add undesired complexity to the designers' already very complex task.

It would be a rather extensive, probably futile, research project to attempt to completely map the semantics of casting design onto a formal knowledge structure. The

best parameterization of the human-software interface is an open issue. The construction of representations of dependencies, specifications and requirements of design is not a well understood process. There have been attempts to develop rule-based systems, see Hill [28]. Good rules are hard to find. This thesis makes no attempt to provide an expert system that can propose designs. Considering the general unavailability of analysis capability to casting designers, this thesis makes the hypothesis that it is more important to provide designers with good analysis tools first. The extent of the knowledge representation built into the virtual foundry was limited by this hypothesis.

A casting interface that uses as much of the designer's language as possible to describe the design problem will be presented in a following section.

3.3.1.2 The interface between the assistant and analysis software

The library of analysis tools is the core of the intelligent design assistant. The assistant must translate the casting information into a form that can be analyzed. Once the analysis is complete, the results must be translated into a form that can be interpreted by the casting designer.

The interaction between the assistant and analysis software is facilitated by the knowledge of the underlying physics of castings and the systems of coupled partial differential equations, state variables, initial values, boundary conditions, etc. Also, the casting geometry must be prepared for analysis by creating a mesh for the system.

A formal description of the analysis requirements of the castings is embedded into a representation of the governing coupled equations. The virtual foundry has on-line knowledge to construct this representation from the designer's specification. The analysis library is capable of interpreting the coupled problem description and providing software

for its analysis. A detailed description of the implementation of a novel system for analysis is the subject of the next chapter.

3.3.2 The key components of the system

The intelligent design assistant is an integrated collection of software tools. It would be difficult, if not impossible, to develop and maintain the virtual foundry as a monolithic program that is capable of all conceivable tasks. Flexibility and adaptability of the design assistant are important factors in its design. The pluggable components of the virtual foundry provide the means needed to deal with a variety of changing analysis problems encountered throughout the casting design process.

Figure 3.3: Data flow through the software components of the Virtual Foundry

Figure 3.3 shows the data flow between the designer and the design assistant. The key components of the design assistant are illustrated within the frames. The user interactions are strictly limited to casting knowledge (interface) and results components. The casting interface provides an environment where the problem is specified in a language native to the designer. The implications of the design decisions are reported back in the form of results from analysis. Other components are isolated or hidden from

the user. Given a specification for a casting design, these components make decisions for automatically planning and configuring the analysis for each problem.

To summarize, the key components are:

- ***Casting Interface:*** This uses the language of the designer and it is the primary interface with the designer. The casting specific design knowledge must be parameterized in a language native to the casting designer. There are important sub tasks such as the user interface and reasoning with geometry. The user's interactions with the system start and end here. The amount of information from the analysis capability exposed to the designer is very critical. It is not in the interest of the designer to know all the details of an analysis.
- ***Visualization of Results:*** The analysis of a casting problem may generate large amounts of data which are not directly usable by the designer. The designers must have feedback in a format from which they can make decisions. Designers quite often interpret the results of an analysis as "interesting" but "indecisive²". Even though the information that they seek is there, it may not be discernible to the designer unless the results are in a useful format. Many visualization tools and methods are available and none is suitable for everything. This component should be adaptable and replaceable depending on the requirements of the user.
- ***Coupled Problem Description:*** This is the second language of the design assistant. The designer's description of the problem must be converted into a form that can be used by analysis programs. This is basically a mapping between two parametric spaces: the description of the casting problem and the descriptions of the mathematical problem. The casting problem must be parameterized and unambiguously translated into a form useful for numerical analysis. This

---

²This claim is based on personal experience; analysis results were presented to a foundry in the early stages of a design.

component partitions the casting problem into a set of conservation equations and produces formats suitable for the analysis programs. It must also provide a representation of the coupling between the equations to model the interactions.

- ***Library of Analysis Tools:*** The formal coupled problem description is a plan provided to the analysis library to specify requirements for an analysis task. This library is a flexible manufacturing system for building analysis engines. The library contains a set of analysis modules and other planning agents that arbitrate and facilitate the communications between modules.
- ***Custom Analysis Tool:*** A customized analysis tool is a set of cooperating analysis agents with a common goal to solve the coupled systems of equations. It can modify its behavior and components at run-time by monitoring the changes in the description of the coupled problem. The custom analysis tool produces the set of results that are presented to the designer.

In the next section, the Casting Interface and its components will be described in detail. Encapsulating the intelligent design assistant by structuring the dependency of its software components and introducing real boundaries between their internal and external behaviors provides a methodology where the complexity of an individual component is deferred and can be dealt with in isolation. Any component can be replaced by another component as long as the new component satisfies the external interface requirements. This a strategy of least commitment.

This methodology for integrating analysis tools to create a design environment can be further generalized and extended to application areas other than casting. This

technique will be reconstructed independently of casting and developed as a general tool in the next chapter.

3.4 Casting interface

The casting interface uses a language that is natural and simple to the casting designer. This language expresses the entities, relationships, processes and activities of casting as defined by the casting industry. Internally it maintains a mapping to translate this information into a form that can be directly contracted to analysis programs.

The casting interface is not merely a format conversion utility that translates the terminology of the designers into input files for analysis programs. It is also the front door to the environment where casting designs can be prototyped rapidly, analysis can be carried out and results can be evaluated, see Figure 3.4.

In Figure 3.5, the casting design and analysis is illustrated from the different viewpoints and abstraction levels involved. Different groups of individuals are concerned with different aspects of the casting. Casting design requires negotiation to resolve conflicts. It is a distributed and delegated process: The schema of design is dynamic and it evolves with the design. Incomplete and inconsistent descriptions arise frequently. The casting design environment must tolerate them.

Associated with a view point is a set of beliefs, values and a conceptual structure from a biased perspective. An abstraction level is the level of granularity at which these beliefs, values and conceptual structures are maintained. From a software perspective, a view point can be a particular parameterization of a concept. Then, the abstraction level would be the hierarchical level of detail at which these parameters are maintained. The

The intelligent design assistant provides the parametric space and a configuration capability that captures the episodic³ knowledge of a casting design. The casting interface is the part of this virtual foundry environment that communicates with the designer. Only a part of the casting design knowledge, both semantic and episodic is captured by the casting interface. For the design assistant, the primary function of the casting interface is to extract just enough information about a design to do an analysis. Therefore, the parameterization of the casting process is biased and not general. It must be expressive enough to capture sufficient information about a casting to perform an analysis.

In Figure 3.5, the functional breakdown of the casting interface is presented. The parameters are extracted from a natural set of variables that are natural to the casting designers. A comprehensive guide to the implementation is provided in the user's reference manual to the Casting Interface, New & Tjoa [26, 31]. The interface is not a script that requires the designers to follow a fixed sequence. The interface allows the designers to follow their priorities. The designers can specify geometry, material, casting process in any order. They can edit any specification at any time.

3.4.1 Geometry

An important component in the casting user interface is a capability to express and reason with the casting and the rigging geometry. The geometry of the casting is usually fixed; the dimensions of the final product are specified by the customer. In designing the pattern, the foundry will modify the dimensions slightly to allow for shrinkage and machining. The rigging is modified repeatedly until the desired casting quality is achieved.

---

³ See Sowa [30], Episodic vs. Semantic memory.

The capability to design a rigging around the casting is critical to the casting interface. The rigging is the most dynamic part of the overall geometry. Much of the design activity and decisions are focused on changes to the rigging. As the rigging is modified, the computational mesh should also be changed. It is well known that creating a mesh for analysis is a long and costly process. A meshing capability is critical to the overall usefulness of the design assistant. Without a good meshing facility, the software may easily be impractical.

The initial stage of dealing with geometry is building the gating system around the casting. This process is known as the rigging design. The user manipulates the orientation of the part, location and the size of the gating system's components.

The geometry of the rigging components and their configuration have significant impact on the quality and cost of the casting. It is difficult for a casting designer to predict how a change in the rigging design will affect the behavior of the castings. A series of changes to the rigging system that are tested with an experimental casting are called "go rounds". Frequently, the experimental casting does not clearly show the effects of the changes, so more tests are carried out. The geometry module in the casting interface application attempts to provide a tool to create and modify the rigging components rapidly, so that an analysis can be carried out to examine their effect.

3.4.1.1 Rigging

Rigging design is changed quite often during an iterative design process. A primary difficulty is associated with managing the complex 3D geometry and creating a mesh for further numerical analysis. Currently, the cost of numerical analysis is dominated by the cost of manual mesh creation. The time required to prepare a mesh can range from a few days to several months depending on the complexity of the casting. The rigging design

module reduces the time required to create meshes to a fraction of the time required by the manual methods.

Figure 3.5: A functional breakdown of the casting interface

Rigging design has attracted the interest of researchers such as Dixon [32] as a geometric problem. Hill [28], studied rigging design from the viewpoint of creating expert systems. The traditional design of gating and the components of rigging systems are best described by Karsay [18].

Figure 3.6: Gating Primitives

A unique software module was developed for rigging design. The application consists of a rigging design module and a constraint based language that is used to develop the rigging components and generic assembly systems. A detailed description and the design of the system can be found in Dai & Goldak [33]. The rigging system is viewed as a geometric assembly. It is decomposed into a set of standard parts that form the building blocks of an assembly system. Foundrymen use such a decomposition to define features such as pouring basins, sprues, sprue basins, runners, gates, risers and geometric variations of them, Figure 3.6. The pieces have fairly simple geometric shapes. They are easily parameterized with a few variables such as length, diameter, type, etc. They usually have circular or rectangular cross sections. The problem with this

type of assembly is managing the dimensional and topological integrity of the whole rigging system when the parameters of individual pieces are changed.

The assembly of components is represented by a graph where the rigging components are the nodes and edges of the graph are the constraints associated with the assembly. To support rapid prototyping and implementation of parametric rigging components, a constraint based novel programming paradigm was developed, [33]. The system is embedded in an object oriented language, Smalltalk [34]. Constraints were used in the specification of the components and the management of the assembly.

In Figure 3.7, a rigging assembly graph is illustrated along with partitions from the underlying constraint graph and the automatically generated mesh. The constraints are automatically built in the background as the user assembles a rigging. These constraints maintain the geometric and topological integrity of the rigging as the parameter such as diameters, widths, heights, etc. are modified. It is often trivial to build large constraint graphs this way that can prove too hard for a human to manage and understand, see Dai [35] for an example. Higher level structures such as multiple levels of abstractions and multiple viewpoints were developed to manage the inherent complexity of the graph topology, see Dai [33].

The original constraint language and the DeltaBlue constraint satisfaction algorithm were due to Borning et. al [36]. Constraints certainly improve the expressive power by providing a mechanism for declarative programming and reduce development times. On the other hand, constraints have on occasion introduced undesired complexity either by causing unexpected behavior or by leading to complex relations. Constraints are particularly effective on a certain class of problems such as rigging design where partitioning based on either function or form can be done and the dependency of the parameters and the order they affect each other can not be determined beforehand. Also it is trivial to specify a set of constraints that has no solution, e.g., $x < 0$ and $x > 0$.

Figure 3.7: A sample rigging assembly and the constraint system

3.5 Analysis

Analysis of castings, like many other phenomena, involves solving systems of coupled partial differential equations (PDE). The essential difficulty is properly decomposing a coupled problem and formalizing the subproblems and their interactions. The solver should be defined at a very high level of granularity to manage the cost and time of software development.

Standard parts, procedures and methods are very often used as a paradigm to hierarchically build a new concept from more primitive parts using uniform methods that are well defined and understood (e.g., circuit design; piping systems; dynamic mechanical networks of masses, springs, dashpots; application of the Finite Element Method as a standard tool to analyze stresses.) In the next chapter, a meta-programming approach will be described where solvers for PDEs are standard components. Solvers for coupled systems of equations can be expressed in terms of programs using this library of standard analysis agents as computational units. These programs can be made to adapt to the requirements of a dynamic problem at runtime. Ultimately, this dynamic network of agents behaves as a single program with continuous behavior.

3.6 A remark on the theory of design

The design process is distributed, evolutionary, negotiated, incomplete and inconsistent. Multiple viewpoints, Struss [37], and multiple abstraction levels are often used to manage complexity. There have been attempts to provide general theories of design and computer aided tools based on several models. However, there is no widely accepted theory and some doubt there will ever be a general theory of design. Computer based design assistants in specific applications areas such as castings, are in our view, a way to work towards a better understanding of issues in aiding designers and improving the design process.

In terms⁴ defined by Brooks [38] for software engineering design, the essential difficulties of design can not be avoided. Design is complex; the dependencies of requirements, specifications, and design decisions are not simple or trivial. Design is distributed and dynamic; there are multiple viewpoints and abstraction levels that

---

⁴ Brooks' classification of the essential difficulties in software engineering design are due to complexity, invisibility, changability and conformability.

constantly evolve and change. It can not be visualized; the relations are so complex that it is not possible to display them in a visualizable space. It is not conformable; there are no uniform natural laws that govern all areas of design.

Accidental difficulties can be removed. For example, CAD removed the accidental difficulty of modifying engineering drawings. The design assistant attempts to remove some of the accidental complexity in designing castings.

Finger & Dixon [1, 2] categorize research areas in their review of theory and methodology of mechanical design. They agree that computer-based models provide the best opportunity to improve theories of design and increase their acceptance. In Finger & Dixon, parametric design and configuration design are identified as the areas that emphasize the importance of the relations between design variables and performance parameters, use of features, and decomposition for knowledge organization.

In parametric-routine design, which is one of the themes in this research, design variables are known. Routine-design is a process that takes place in a setting where certain beliefs and practices constrain the design, the designer is not permitted to modify certain variables and rules, which in effect are treated as laws. Often the designer is asked to work within a strict set of rules and standard components. Artifacts of the routine-design process are often, though not always, modifications of previous designs (variant design). Modifications are limited to a few parameters and in some cases the parameters may only be selected from a finite set. The design of a rigging system is a good example of routine design. Piping design that conforms to an ASME code is also a familiar example.

3.7 Summary

In this chapter, a software environment to support casting design was presented. The main concept of this environment is an Intelligent Design Assistant. The design assistant is an autonomous entity that is capable of making decisions about low level activities in the design process such as setting up and administrating details of analysis. The design assistant provides help to the casting designer by doing analysis that frees him to concentrate on creative higher level aspect of the design.

The Virtual Foundry is an environment where the casting designer and the intelligent design assistant interact. The key software components of the virtual foundry are the design assistant and an analysis library. The interactions between the designer and the assistant are based on the shared domain-specific knowledge of castings. This is embodied in the casting interface in a language native to the casting designer.

A component of the Casting Interface deals with the geometry. A novel application for rigging design was described.

Analysis capability is in the core of the design assistant. A formal representation of the underlying physics and mathematics must be used to describe the casting design as an analysis problem.

Chapter 4: Basic physics and mathematical foundations of casting analysis

In this chapter a foundation for the physical background, the underlying mathematics and their discretized forms for a numerical treatment relevant to the analysis of castings will be developed. A compact summary of the relevant physics, mathematics and the numerical methods is provided. The goal of the chapter is neither to provide an exhaustive account of all physics pertinent to casting analysis, nor to study the resulting equations with deep mathematical rigor. These topics are widely covered in the literature.

In some cases, the numerical methods are extended to provide a concrete example of numerical analysis for practical casting analysis and design. Detailed accounts of the physics involved in castings are found in references such as Campbell[22], Beeley[8], ASM Handbooks[39, 40]. References to detailed and rigorous studies of the recent

mathematics and numerical work are provided within sections. In the presentation of the subject, castings analyses and efficient numerical schemes are primary concerns.

The chapter is divided into sections examining the Fluid Flow, Free Surface, Energy equations and how they are coupled. Emphasis is given to deriving continuum equations; the detailed microscopic effects of phenomena such as diffusion or dissipation are ignored. Each section develops the continuum equations and detailed discrete forms of these equations. Numerical algorithms and their effectiveness are presented by sample problems. A separate section is dedicated to the study of subtleties in coupling these equations.

4.1 The fluid flow equations

This section will develop the basic equations of fluid mechanics. The physical laws governing fluid flow are written in terms of a set of variables; these physical quantities form the state space.

A set of generally accepted physical quantities in fluid flow is $\{u, p, \rho, T\}$, where u, p, ρ, T are the velocity, pressure, density and temperature respectively. The definitions of these variables are not arbitrary. More precisely, these variables are functions of points in space (i.e., the Euclidean Space $\mathbb{R}^3$), and time. Furthermore, in order to be solutions of differential equations such as the Navier-Stokes equation, these field variables must belong to an appropriate function space, i.e. Sobolev Function Space¹.

The macroscopic behavior of a fluid is considered. The smallest volume or particle is assumed to be significantly larger in size than the atoms. Accepting existence

---

¹ Chung [41], provides a brief summary, see pages 10-31. More formal presentation of the subject can be found in advanced mathematical texts.

of the density function is one of the fundamental assumptions of continuum mechanics. The fact that ρ is in the state space is the continuum assumption. This state space is obviously not appropriate to describe the physical behavior of a fluid at the atomistic level, however this is not a problem because it is below the scale of detail the casting analysis requires.

4.1.1 The conservation laws

The equations governing the fluid flow can be studied under a more general heading of conservation laws. Although this level of abstraction is appropriate from a mathematical standpoint, a more focused approach, which emphasizes the study of these equations from a fluid flow perspective is adopted to study the equations in this section. For the interested readers, Majda [42], provides a study of the conservation laws pertinent to fluid mechanics from a mathematical perspective. Classical mathematical treatment of the fluid mechanics equations may be found in references such as [43], [44], [45]. There are also many physical interpretations such as [46].

The equations of fluid flow are derived from three conservation laws: The conservation of mass, the conservation of momentum and the conservation of energy.

4.1.1.1 Conservation of mass

The assumption is that mass can neither be created nor destroyed. This can be stated mathematically in more than one way. Given a region D , Ω is a sub region (domain) contained in D , i.e. $\Omega \subset D$. Let $\partial\Omega$ be the boundary of the region Ω , n

Figure 4.1: Fluid body Ω , with boundary $\partial\Omega$ and outward unit normal n

denotes the unit outward normal, see Figure 4.1. The domain Ω is fixed in space, i.e. it does not change with time. Using an Eulerian frame (x, t) , in the Euclidean space $\mathbb{R}^3$, where $x \in \Omega$ and time $t \in \mathbb{R}^+$, the rate of change of mass in this volume can be expressed as: $\frac{d}{dt}m(\Omega, t) = \frac{d}{dt} \int_{\Omega} \rho(x, t) dV = \int_{\Omega} \frac{\partial \rho}{\partial t}(x, t) dV$. Given a velocity field $u(x, t)$, let $(\rho u \cdot n)$ be the mass flow rate across the boundary, then, the conservation of mass is:

$$\begin{aligned} \frac{\partial}{\partial t} \int_{\Omega} \rho(x, t) dV &= \int_{\Omega} \frac{\partial \rho}{\partial t}(x, t) dV = - \int_{\partial \Omega} (\rho u \cdot n) dA = \int_{\Omega} \text{div}(\rho u) dV \\ \int_{\Omega} \left\{ \frac{\partial \rho}{\partial t} - \text{div}(\rho u) \right\} dV &= 0 \end{aligned} \quad (4.1)$$

The equation reads as: the rate of change of mass in the volume Ω equals the net mass flow across the boundary for any arbitrary region $\Omega \subseteq D$.

A Lagrangian frame $\Omega(t) \subset D$, such that $x(t) = x(t_0) + u(t_0)(t - t_0)$, $\forall x(t) \in \Omega(t)$, may also be used to derive the mass conservation equation, see Figure 4.2. Following a material point in $\Omega(t_0)$, its volume may change but the mass should be conserved, $\frac{D}{Dt}(\rho dV) = 0$, where $\frac{D}{Dt} = \frac{\partial}{\partial t} + u \cdot \nabla$, is the material derivative.

Figure 4.2: Volume Ω as it moves with the flow

The rate of deformation may be stated² as $\frac{D}{Dt}(dx) = (\nabla u)dx$, where dx is a vector that belongs to $\Omega(t)$, i.e. x is a material point, t is fixed. The gradient of the velocity at a material point can be decomposed into a symmetric (stretching tensor), and an anti symmetric part (rotation/spin tensor). Stretching is also called rate of deformation.

²Derivations can found in references such as [47].

$$\nabla u = \text{Stretching} + \text{Spin} ,$$

$$\text{Stretching} = \frac{1}{2}(\nabla u + \nabla u^T), \text{ Spin} = \frac{1}{2}(\nabla u - \nabla u^T)$$

The first scalar invariant of the rate of deformation tensor (dilatation), is the rate of change of volume per unit volume, i.e. $\frac{1}{dV} \frac{D}{Dt}(dV) = \text{trace}(\text{Rate of Deformation}) = \text{div}(u)$.

Expanding the statement of mass conservation in a Lagrangian frame and replacing the term for the rate of change of volume with the above term, we arrive at another form of the same conservation equation:

$$\begin{aligned} \frac{D}{Dt}(\rho dV) &= dV \frac{D\rho}{Dt} + \rho \frac{D(dV)}{Dt} = 0 \\ \frac{D\rho}{Dt} + \rho \text{div}(u) &= 0 \quad (4.2) \\ \frac{\partial \rho}{\partial t} + u \cdot \nabla \rho + \rho \text{div}(u) &= 0 \end{aligned}$$

Equations 4.1 and 4.2 are equivalent and provide integral and differential forms of the conservation of mass equation.

4.1.1.2 Conservation of momentum

The principle of conservation of linear momentum, also known as Newton's second law, states that the total force (surface and body forces) acting on any fixed body of material is equal to the rate of change of linear momentum.

$$\frac{D}{Dt} \int_{\Omega(t)} (\rho u) dV = \int_{\partial \Omega} s dA + \int_{\Omega(t)} (\rho b) dV$$

where b are the forces acting on the body at a distance, such as gravity or electromagnetic forces, and s is the stress vector. Furthermore, using the assumptions of simple fluids, the stress vector can be decomposed into natural pieces with physical significance, i.e. the stress tensor $\sigma = T_p + T_{viscous} = p I + T_{viscous}$, where p is scalar pressure and $T_{viscous}$, which is

sometimes called the viscous stress tensor. Therefore the force exerted on the body is $s = (pI + T_{viscous}) \cdot n$. More precisely, if Ω_A is one fluid region, Ω_B another, separated by the surface Σ , the force exerted by Ω_B on Ω_A at a point is $s = \sigma \cdot n$, where σ and n are a linear transformation and the normal vector defined at that point respectively, see Figure 4.3. There may be forces parallel to n in $T_{viscous}$. Assuming that the force s is a linear function of n , $T_{viscous}$ depends linearly on velocity gradients, is invariant under rigid body rotations and is symmetric (This result is known as Cauchy's Theorem), $T_{viscous}$ can be rewritten as:

Figure 4.3

$$T_{viscous} = \lambda \operatorname{div}(u)I + \underbrace{2\mu \left(\frac{1}{2}(\nabla u + \nabla u^T) \right)}_{\text{deviatoric stress}}$$

μ is often called the first coefficient of viscosity or kinematic viscosity, to which the dissipative forces are attributed. λ is the second coefficient of viscosity, it relates the rate of changes in volume to viscous mean normal stresses. λ is also known as the coefficient of bulk viscosity.

Inserting these into the integral form of the conservation of linear momentum equation, applying the divergence theorem and a certain amount of reorganization, it can be shown that the conservation of linear momentum can be now written as:

$$\rho \left[\frac{\partial u}{\partial t} + (u \cdot \nabla) u \right] = -\nabla p + (\lambda + \mu) \operatorname{div}(u) + \mu \operatorname{div}(\nabla u) \quad (4.3)$$

This equation is more popularly known as the Navier-Stokes equation.

4.1.1.3 Conservation of energy

Let the kinetic energy of a differential volume at time t to be denoted as KE . Let U denote its internal energy, Q the net rate of heat flow into the particle by its surroundings and W the rate at which work is done on the particle by external forces. Then, the conservation of energy can be written as:

$$\frac{D}{Dt}(U + KE) = W + Q \quad (4.4)$$

Further insight into the energy equation can be gained by introducing thermodynamic principles. To avoid a generalized study and diversion from the topic of main interest, the study of the energy equation will be delayed until after the incompressible flow conditions are introduced.

4.1.2 The incompressible fluid flow equations

A class of fluids, such as water, mercury and other liquid metals have the property that their density is almost invariant over a wide range pressures. Therefore the simplifying assumption $\frac{D\rho}{Dt} = 0$, is usually made in the study of their behavior. This class of equations are called *Incompressible* due to the fact that the density remains the same at all time regardless of the stresses applied. In our context, a flow is incompressible if all changes in pressure cause no change in density. However, this does not imply that the density is also constant in space. The density can change due to changes in temperature, composition, or phases such as liquid to air. If the density is constant, the flow is incompressible; but this is only a sufficient condition, not a necessary one.

The principle of incompressibility is applicable to mixtures of two or more individually homogenous incompressible fluids. The previous equations of fluid flow

can be extended to inhomogeneous fluids, such as those that are of interest in casting analysis. Imagine a mixture where each differential volume dV is composed of fractions of two fluids with concentration ψ , the ratio of mass of one component to the total mass of the fluid. If we assume that the changes in the distribution of concentration is only due to macroscopic motion of the fluid or mechanical mixing, i.e. with the velocity field u , and not by any other mechanism such as molecular transfer of components (diffusion), and that the thermal conduction/reaction and internal friction between the two components are ignored, there would be no dissipation of energy due to the mixing of two fluids. Using averaged density $\langle \rho \rangle = \psi \rho_a + (1 - \psi) \rho_l$, where $\psi = \frac{m_a}{m_{total}}$, the equation of continuity for the mixture can be written as:

$$\frac{\partial \langle \rho \rangle}{\partial t} + u \cdot \nabla \langle \rho \rangle + \langle \rho \rangle \text{div}(u) = 0$$

The continuum mechanics of mixtures will be discussed in more detail later. In the following discussions, unless necessary, the use of the notation $\langle \rangle$ is ignored for averaged quantities, the meaning of the notation should be clear from the context. Obviously averaged density can also be applied for mixtures of more than two fluids.

For incompressible flows we know that the $\frac{D\rho}{Dt} = 0$, inserting this term in the previous expression of the conservation of mass, the equation is reduced³ to:

$$\begin{aligned} \rho \text{div}(u) &= 0 \\ \text{div}(u) &= 0 \end{aligned} \quad (4.5)$$

Applying equation (4.5) to (4.3), and using material averaged quantities for a continuum of a fluid mixture, the Navier-Stokes equations for incompressible flow are obtained:

$$\rho \left[\frac{\partial u}{\partial t} + (u \cdot \nabla) u \right] = -\nabla p + \mu \text{div}(\nabla u) \quad (4.6)$$

³So far, incompressibility is based on the assumption that density is independent of pressure. If there are rapid temperature changes then the equation (4.5) is not valid. It must be replaced by a more detailed expression.

Since the density is invariant, the pressure can not be explicitly determined from density within the incompressible flow formulation. However, it is determined using the incompressibility condition (4.5). The Helmholtz-Hodge decomposition theorem is presented without a proof⁴: It implies that any smooth vector field w on Ω can be uniquely decomposed into $w = u + \text{grad } p$, where $\text{div}(u)=0$, and $u \cdot n = 0$ on $\partial\Omega$, and p is a scalar field determined only up to an additive constant. This theorem provides an important result for numerical algorithms; the conservation of mass is enforced by projecting the $\text{grad } p$ on any vector field w , to obtain a velocity field u which is divergence free, see the Figure 4.4. It is also known that the solution to this problem exists and is unique for $(p+c)$, c is an arbitrary constant.

Figure 4.4: $w = u + \text{grad}(p)$

More specifically, if P defines a linear orthogonal projection operator $P(w)=u$ such that $P(u)=P^2(w)=u$, $P(\text{grad } p) = 0$, then applying P to both sides of (4.6), the pressure can be eliminated from the Navier-Stokes equation for incompressible flow. Then, the pressure is usually recovered using the gradient of velocity terms in the pressureless Navier-Stokes equation.

Also, the incompressible flow implies that the only mechanism for converting the inflow of energy to internal energy is through viscous dissipation. Therefore, the only two mechanisms which energy interactions will take place is kinetic energy and viscous dissipation:

$$\frac{d}{dt} KE \leq 0,$$

$$\frac{d}{dt} KE = \frac{d}{dt} \frac{1}{2} \int_{\Omega} \rho \|u\|^2 dV = \frac{1}{2} \int_{\Omega} \rho \frac{D\|u\|^2}{Dt} dV = \int_{\Omega} (-u \cdot \nabla p + \mu u \cdot \text{div}(\nabla u)) dV$$

⁴See. Chorin & Marsden [43], page 36, for a proof.

2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISION^{SM} RESOLUTION TARGETS

$$\text{but } \operatorname{div}(u) = 0, u \cdot \nabla p = 0, u \cdot n = 0$$

$$\frac{d}{dt} KE = -\mu \int_{\Omega} \|\nabla u\|^2 dV + \int_{\partial\Omega} u \cdot \nabla u dA, \text{ where } \|\nabla u\|^2 = \nabla u \cdot \nabla u$$

if $u = 0$ at the boundary $\partial\Omega$, then

$$\frac{d}{dt} KE = -\mu \int_{\Omega} \|\nabla u\|^2 dV \leq 0$$

In summary, the conservation laws for the incompressible fluid flow and the associated boundary conditions, such as $u=0$ at the walls (no-slip), $u \cdot n = 0$, the Dirichlet boundary conditions $\partial\Omega_D$, and the higher order boundary conditions due to the second order derivatives in the Navier-Stokes equation, the Neumann boundary conditions $\partial\Omega_N$, will be written as:

$$\begin{aligned} \rho \dot{u} - \mu \Delta u + (\nabla u)u + \nabla p &= f & \text{in } \Omega &\subset \mathbb{R}^1 \\ \operatorname{div}(u) &= 0 & \text{in } \Omega & \\ u &= g & \text{in } \partial\Omega_D & \\ (-pI + 2\mu \Lambda u)n &= h & \text{on } \partial\Omega_N & \\ \Lambda u &= \frac{1}{2}(\nabla u + \nabla u^T) & & \\ \partial\Omega &= \partial\Omega_D \cup \partial\Omega_N & & \\ u &= u_0 \text{ at } t = 0, \text{ where } \operatorname{div}(u_0) = 0 & & \end{aligned} \quad (4.7)$$

4.1.3 Numerical solution of the incompressible Navier-Stokes equations: FEM discretization

There are a variety of highly efficient numerical algorithms for the solution of incompressible Navier-Stokes equations. The references given here are certainly not exhaustive but they are representative of the type of solutions available today. Chorin[48-50], among others, presented finite difference models as early as 1960s. References such as [49], introduced the $\operatorname{div}(u)=0$ condition as a constraint which became standard practice. The finite element method appears at a latter stage, Chung [41], Gallagher [51]. Modern approaches like space-time formulations and extensions of the

previous efforts may be found in references such as Brooks[10], Gresho[11, 52-55], Jiang[12], Hansbo[56, 57], Hughes[58], Hughes[59], Shakib[60], Pironneau [61], etc. There are alternative models such as spectral methods, or the velocity-vorticity formulation, which is more suitable for certain problems and the pressure does not appear in the formulation directly, [62].

4.1.3.1 Finite element spatial discretization

The finite element method which will be described here is essentially that of Gresho and Chan[54, 55]. Using Einstein's summation notation, i.e. $s = \sum_{i=1}^n a_i x_i$ is denoted by $a_i x_i$, the partial derivative of a with respect to a reference frame i , by $\partial a / \partial x_i = a_{,i}$, and using a Cartesian coordinate frame the Navier-Stokes equation is written as:

$$\rho u_{i,t} + \rho u_j u_{i,j} + (p \delta_{ij})_{,j} + \mu(u_{i,j} + u_{j,i})_{,j} = 0 \quad (a)$$

$$u_{i,i} = 0 \quad (b) \quad (4.8)$$

$$i, j = 1, 2, 3 \text{ in 3D}$$

The finite element approximations for velocity and pressure are, $u = \sum_{k=1}^N \tilde{u}_k \phi_k$, $p = \sum_{k=1}^M \tilde{p}_k \varphi_k$ respectively. ϕ and φ are basis functions for velocity and pressure. $\tilde{u}$ and $\tilde{p}$ are the nodal coefficients of the finite element approximations of the velocity and pressure fields. $\{\tilde{u}\}$, and $\{\tilde{p}\}$ will denote the associated vectors. The pressure basis functions are generally one degree lower than the velocity basis functions for reasons which may be found in Fortin [63]. The finite element spatial discretization (weak form), of the Navier-Stokes equations is performed via the Galerkin method. Multiplying Equation 4.8(a) by any of the velocity basis functions $k=1, \dots, N$ and integrating over the domain :

$$\int_{\Omega} \{ \rho u_{i,j} + \rho u_j u_{i,j} + (p \delta_{ij})_{,j} + \mu (u_{i,j} + u_{j,i})_{,j} \} \phi_k = 0$$

$$\int_{\Omega} \{ \rho u_{i,j} + \rho u_j u_{i,j} \} \phi_k + \int_{\Omega} \{ (p \delta_{ij} \phi_k)_{,j} + \mu [\phi_k (u_{i,j} + u_{j,i})]_{,j} \} - \int_{\Omega} \left\{ p \delta_{ij} \phi_{k,j} + \mu' \left(\frac{\partial u_i}{\partial j} + \frac{\partial u_j}{\partial i} \right) \phi_{k,j} \right\} = 0$$

Applying the divergence theorem to the second integral of the last equation the boundary integral terms, the boundary terms are derived:

$$\int_{\Omega} \{ (p \delta_{ij} \phi_k)_{,j} + \mu [(u_{i,j} + u_{j,i}) \phi_k]_{,j} \} = \int_{\partial \Omega} (p \delta_{ij}) \phi_k \cdot n_j + \int_{\partial \Omega} \mu (u_{i,j} + u_{j,i}) \phi_k \cdot n_j$$

Using the Galerkin method to compute a weak form of the continuity equation, we arrive at $\text{div}(u) = 0$ condition as:

$$\int_{\Omega} u_{i,i} \phi_k = 0$$

where $\phi_k, k=1, \dots, M$ are the basis functions spanning the space of pressure.

Expanding the weak form of the equations (4.8) using the finite element approximations for the velocity and the pressure fields, and arranging the equations, the following equation will be obtained:

$$G\{\tilde{u}\} + \{K + N(\tilde{u})\}\{\tilde{u}\} + B^T\{\tilde{p}\} = b \quad (a)$$

$$B\{\tilde{u}\} = 0 \quad (b)$$

initial conditions:

$$\tilde{u}(t_o) = \{\tilde{u}_o\}, \quad B\{\tilde{u}_o\} = 0 \quad (c) \quad (4.9)$$

boundary conditions:

$$\{\tilde{u}\} = g \quad \text{on } \partial \Omega_D \quad (d)$$

$$(-\tilde{p}I + 2\mu \Lambda\{\tilde{u}\}) \cdot n \quad \text{on } \partial \Omega_N$$

where:

$$G^s = \int_{\Omega} \rho \phi(x) \phi^T(x) dV, \quad G = \begin{bmatrix} G^s & 0 & 0 \\ 0 & G^s & 0 \\ 0 & 0 & G^s \end{bmatrix},$$

$$K^S = \int_{\Omega} \nabla \phi(x)^T \mu \nabla \phi(x) dV, K = \begin{bmatrix} K^S & 0 & 0 \\ 0 & K^S & 0 \\ 0 & 0 & K^S \end{bmatrix}, \quad (4.10)$$

$$N(\tilde{u})^s = \int_{\Omega} \rho \phi(x)^T \tilde{u}^T \nabla \phi(x) dV, N(\tilde{u}) = \begin{bmatrix} N(\tilde{u})^s & 0 & 0 \\ 0 & N(\tilde{u})^s & 0 \\ 0 & 0 & N(\tilde{u})^s \end{bmatrix} \quad (4.10)$$

$$B^T = \int_{\Omega} R^T \phi(x) dV$$

$$b = \int_{\partial\Omega} h_{3 \times 3} \begin{bmatrix} \phi(x) \\ \phi(x) \\ \phi(x) \end{bmatrix}_{3 \times N} dA$$

and,

$$\phi = [\phi_1, \dots, \phi_N]_{1 \times N}, \varphi = [\varphi_1, \dots, \varphi_M]_{1 \times M}$$

$$\nabla \phi = \begin{bmatrix} \partial \phi / \partial x \\ \partial \phi / \partial y \\ \partial \phi / \partial z \end{bmatrix}_{3 \times N}, R_{1 \times 3N} = [1, 0, 0, 0, 1, 0, 0, 0, 1] \begin{bmatrix} \nabla \phi & 0 & 0 \\ 0 & \nabla \phi & 0 \\ 0 & 0 & \nabla \phi \end{bmatrix}$$

$$\text{either } h = (-\tilde{p}I + \mu \nabla \tilde{u}) \cdot n \quad (\text{pseudo-tractions})$$

$$\text{or } h = (-\tilde{p}I + 2\mu \Lambda \tilde{u}) \cdot n \quad (\text{traction})$$

For a detailed description of the traction boundary conditions see Gresho[54]. The matrices are: The element volume matrix G , the diffusion matrix K , the advection matrix $N(u)$, the divergence operator B , and the gradient operator B^T . The Galerkin method leads to discretized approximations of the advection terms $N(u)$, which may introduce "artificial diffusion" and oscillations, an unphysical form of the dissipation of energy.

Brooks & Hughes [10], provides a detailed study of the advection term. In what they described as the streamline-upwind concept, a method was developed for the advection diffusion equation and several alternatives are discussed for picking a free parameter (γ). With their modification, the diffusion matrix is described as (4.11).

$$K(u) = \int_{\Omega} \nabla \phi(x)^T \alpha(\tilde{u}) \nabla \phi(x) dV, \quad (4.11)$$

$$\text{where } \alpha(\tilde{u}) = \mu I + \gamma \tilde{u} \tilde{u}^T, \quad \tilde{u} = \frac{\tilde{u}}{\|\tilde{u}\|}$$

4.1.3.2 Time integration

The solution of the equation (4.9) for u and p requires further discussion. Gresho[54], provides an interpretation of the equations as projections. In equation 4.9(a), the $\{\tilde{u}\}$ is generally neither divergence-free nor curl-free. Velocity and the pressure equations are usually solved sequentially, and using projections such as those defined by Chorin [50], pressures are obtained from the statement of acceleration, the solutions of u are then projected onto a divergence-free space. In order to solve the equation (4.9) with an explicit method, the pressure and the velocity equations are usually decoupled. Chorin[48, 49] provides a mathematical background and proofs, (consequences of the Helmholtz-Hodge decomposition). The $\text{div}(u)=0$ condition can be satisfied more or less "automatically" by a re-organization of the equation (4.9); by the time derivative of equation 4.9(b), $B\{\dot{\tilde{u}}\} = 0$ is obtained. Since $B\{\tilde{u}\} = 0$ should always be true, moving all the velocity terms to the right hand side of the Equation 4.9(a) and multiplying both sides with BG^{-1} and canceling the terms with $B\{\tilde{u}\} = 0$, the following equation for pressure will be obtained:

$$(BG^{-1}B^T)\{\tilde{p}\} = A\{\tilde{p}\} = BG^{-1}[b - K\{\tilde{u}\} - N(\tilde{u})\{\tilde{u}\}] \quad (4.12)$$

The matrix A is sometimes called the discretized Laplacian operator and the resulting pressure equation is called the Pressure-Poisson Equation. There are differences in the literature on the execution of the projections and the time integration schemes. A review of the methods used by different researchers is provided in [54]. The time integration method that has been used in this research is, (see Hughes [59]):

$$1.) (G + \Delta t \theta K) \hat{u}^{n+1} = Gu^n + \Delta t(b^n - (1 - \theta)Ku^n - N(u^n)u^n - B^T p^n) \quad , \text{ solve for } \hat{u}^{n+1}$$

$$2.) AQ = B\hat{u}^{n+1} \quad , \text{ solve for } Q$$

$$3.) u^{n+1} = \hat{u}^{n+1} - G^{-1}B^T Q, \text{ update velocity}$$

$$4.) p^{n+1} = p^n + \frac{Q}{\Delta t}, \text{ update pressure}$$

Advection term is integrated via forward Euler

$\theta = 0$ is forward Euler, $\theta = 1$ backward Euler, $\theta = 0.5$ Trapezoidal

$\hat{u}$ is an intermediate velocity(acceleration), Q is an intermediate pressure

u , p , Δt are velocity, pressure and time increment respectively

(u and p are actually $\{\tilde{u}\}$ and $\{\tilde{p}\}$, but this notation is ignored here)

4.1.3.3 Remarks

There are simplified and more cost effective versions of the equations using lumped G matrix, etc. Hughes [59], Gresho [11, 53, 55].

From the computational point of view, the pressure equations requires most of the computer effort. There is some room for reducing computation cost such as computing the A matrix once and storing it. However, if the Navier-Stokes equation is coupled to other equations, such as the free-surface evolution, the matrix must be formed and inverted at every time step.

Another complication are the spurious pressure modes. The pressure p solved here can be replaced by any other pressure which is $p + \text{constant}$. If the matrix A is singular, then any solution for p that is in the null space of A is a spurious pressure mode. Sani [64, 65], provides a study of the cause and cure(?) of this problem. A more recent study by Phillips [66], also provides further insight into this interesting topic.

4.1.4 Numerical example

The lid driven cavity problem is solved frequently as a benchmark to assess the correctness, and quality of the numerical solutions of the incompressible Navier-Stokes equation. Ghia [67], Jiang [12], Gresho [53] are some of the references that provide detailed results for a range of Reynolds numbers (100-50000).

The problem consists of a square shaped domain with no-slip boundary conditions applied at all the walls except the lid (usually the top face of the cavity). A non-zero value for tangential velocity is prescribed on the lid. The nature of the boundary conditions also has been studied in detail and they range from a constant tangential velocity, which is also called "*leaky flow*" due to prescribed velocity at the corners, to parabolic distributions of velocity with maximum magnitude at mid edge and zero at the corners.

The units were arbitrary. The problem shown in Figure 4.5 has unit dimension along the bottom and side of the cavity. The finite element mesh was 30x30 quadratic elements and the elements near the boundary are finer than the elements in the interior. The no-slip boundary conditions were applied at all the walls except the lid where a Dirichlet velocity boundary condition with a magnitude of 1 was applied. The viscosity was set to 0.0025 to attain a Reynolds number of 400. The streamline-upwinding term (4.11), was suppressed for some of the experiments to study its effect. A very rudimentary turbulence model was also tried by introducing additional damping scaled by the mesh Reynolds number above a certain threshold value ($Re=10$).

The numerical experiments agreed very well with the results published by Ghia [67], see Figure 4.5 . The streamline-upwinding term introduced additional damping, the maximum velocity attained in this case was lower than all the other cases. The

(a)

(b)

(c)

Figure 4.5: The Lid-Driven Cavity Problem (a) Velocity vectors and flow paths (b) The x-components of the velocity vectors on a vertical line passing through the geometric center of the cavity (c) The y-components of the velocity vectors on a horizontal line passing through the geometric center of the cavity

artificial damping for turbulence modelling also dampened the velocity field, it was not as pronounced as it was in the case for streamline-upwinding.

4.2 The free-surface equations

The physics of the boundary between liquid metal and the air and motion of the liquid metal are of special interest for filling analysis in castings. The boundary between the liquid metal and air will be called the free-surface. For fluids that are in motion the interface is not in thermodynamic and mechanical equilibrium. However, assuming that local departure from equilibrium is not too large we can study the transport of momentum across the surface and the evolution of the surface. The effect that any departure from equilibrium has is to tend to make the velocity and temperature across the surface uniform. This in effect causes the transport of heat or momentum, Batchelor [68].

It will be assumed that both fluids are incompressible and Navier-Stokes equations govern the motion in each fluid. A small volume of this fluid system is denoted by Ω , and $\partial\Omega$ is the boundary of the volume. Ω and $\partial\Omega$ are fixed in space. The free-surface will be denoted by Π , see Figure 4.6.

Figure 4.6: The free-surface Π

Figure 4.7: The stresses acting on Π

The velocity u (normal and tangential) is continuous across the boundary separating the two fluids. The normal component of the velocity is continuous for kinematic reasons, i.e. unless a rupture occurs, the boundary remains a material surface for both fluids. To

study the flux of momentum across the surface the relation between the stresses on the two sides of the boundary are established, see Figure 4.7. The balance of forces can be written as (also see Landau [44]):

$$\sigma_l \cdot n - \sigma_g \cdot n = \tau \left(\frac{1}{R_1} + \frac{1}{R_2} \right)$$

where τ is the surface tension coefficient, and

$$\sigma = p\delta_{ij} + \mu(\nabla u + \nabla u^T),$$

R_1 and R_2 are principal Radii of Curvature

This equation reads as: the jump in tractions across the surface is entirely balanced by the effect of surface tension. In studying the boundary conditions, some observations can be made; Batchelor [68], argues that if the viscosity and the density of the gas is much smaller than those of liquid, it can be observed from the Navier-Stokes equation that the pressure and frictional stress variations in the gas will diminish with density and viscosity. Therefore, $p_o\delta_{ij}$, where p_o is uniform gas pressure, can be taken as a viable approximation to stresses in the gas. Campbell [22], points out that the surface tension effects are critical for narrow cross-sectioned casting problems and can be safely ignored in larger sections. If these simplifications are made, and the flow in air is not of interest then the tractions across the surface may be assumed to be continuous and the expression reduces to:

$$\left[p_l\delta_{ij} + \mu(\nabla u + \nabla u^T) \right] \cdot n = p_o\delta_{ij} \cdot n$$

4.2.1 The numerical solution of free-surface equations

The numerical method used to solve the partial differential equations governing the motion of the fluids (Navier-Stokes equation) was described earlier. The methods described before are valid in regions where the solutions are smooth; in the neighborhood

of the free-surface, unless proper techniques are employed, the numerical methods described before will have difficulty in modelling realistic physics.

There are a number of methods to deal with this difficulty; an easy way to deal with a discontinuity such as the free-surface is to smear it using techniques such as artificial dissipation or viscosity. Another method is to explicitly represent the free-surface as an internal boundary, and impose the proper boundary conditions in the underlying PDE. Hyman [69], classifies these methods as artificial dissipation and interface tracking methods. Although very easy to implement, artificial dissipation methods are not found to be very effective when the two fluids are radically different. The variations of the interface tracking methods include surface tracking methods where the free-surface is represented by a set of lines in 2D and surfaces in 3D and advected with the flow [70-72].

The surface tracking methods are somewhat more complex to implement due to the detailed geometry of the interface which must be maintained continuously. Volume tracking methods are a good compromise from the complexity of the surface tracking without sacrificing the sharpness of the interface. They have been found at least as accurate for smoothly varying multiple interfaces. Examples of the volume tracking method are the marker and cell MAC algorithms [73], where each cell is either of the two fluids and the markers are advected. However, the MAC method was found to be overly diffusive causing a smeared interface. A variation of the MAC algorithm is the fractional marker volume, or the volume of fluid VOF, where the surface is defined by the fractional volume of each material (ranges from 0 to 1) in each cell, [74], [75], [76]. Alternatively, there are moving mesh methods that adaptively modify the computational domain to optimally track the interface, [77], [78]. There are also other interesting implementations and variations of these methods such as [79-84].

A variation of volume tracking approach is used in this research, [85]. The free-surface Π is represented as a level set of a scalar field $\psi = \psi(x, t)$, $x \in \Omega$, $\psi \in [0, 1]$, and the position of the free-surface at a given time t is $\Pi(t) = \{x \mid \psi(x, t) = c\}$, c is a fixed value. There are continuum methods presented in the literature that also include the effect of the surface tension for volume tracking methods, [86]. However, if the effects of surface tension are ignored, then the evolution of the scalar quantity ψ can be modelled by passively convecting the scalar field. The resulting conservation law is:

$$\begin{aligned} \dot{\psi} + \text{div}(\psi u) &= \dot{\psi} + \psi \text{div}(u) + u \cdot \nabla \psi = 0 \\ \text{if } \text{div}(u) &= 0 \text{ (incompressible flow), then} \\ \dot{\psi} + u \cdot \nabla \psi &= 0 \end{aligned} \quad (4.13)$$

Assuming that the velocity u is given, and defined on all of Ω in a time interval $t \in [0, T]$, an initial value problem for the hyperbolic system (4.13) can be defined:

$$\begin{aligned} &\dot{\psi} + u \cdot \nabla \psi = 0 \quad \text{on } \Omega \times [0, T] \\ \text{initial values} & \quad \psi(x, 0) = \psi_o(x) \quad \text{on } \Omega \\ \text{and boundary conditions} & \quad \psi(x, t) = q(x, t) \quad \text{for } t \in [0, T] \text{ and, } x \in \Gamma_{\text{in}}(t) \\ \text{where,} & \quad \psi_o, q \text{ are given functions} \\ & \quad \Gamma_{\text{in}} = \{x \in \Gamma \mid u \cdot n < 0\}, \text{ inflow boundary} \\ & \quad \Gamma_{\text{out}} = \{x \in \Gamma \mid u \cdot n > 0\}, \text{ outflow boundary} \\ & \quad \Gamma_{\text{solid}} = \{x \in \Gamma \mid u \cdot n = 0\}, \text{ solid boundary} \end{aligned} \quad (4.14)$$

The critical issue in solving (4.14) is to reduce the amount of numerical smoothing. A second-order accurate method similar to van Leer's [87] is developed⁵ instead of the more diffusive, first order accurate methods such as [74]. Assuming that the scalar function ψ is piecewise linear on each computational element and represented by the value of ψ at the centroid of the element, ψ_c , and the local gradient of ψ , $\nabla \psi_c$. In other words,

⁵Also see Hughes [85] for the detailed model.

$\psi_e = \psi(c_e) = \frac{1}{V_e} \int_{\Omega_e} \psi d\Omega_e$, where e is the element, V is the volume, c is centroid of the element. Using this definition, the ψ field in an element is defined as:

$$\psi(x) = \psi_e + \nabla \psi \cdot (x - c_e) \quad (4.15)$$

where x is the distance from the centroid of the element.

After integrating (4.14) over an element and dividing by the volume:

$$\dot{\psi}_e + \frac{1}{V_e} \int_{\partial\Omega_e} \psi(x) u \cdot n = 0 \quad (4.16)$$

Discretizing (4.16) for an element e with the set of faces $\text{Faces}[e]$, where A is the area of a face f , and integrating over the faces by the one point rule, an approximate boundary integral is obtained:

$$\dot{\psi}_e + \frac{1}{V_e} \sum_{f \in \text{Faces}[e]} \psi_f (u_f \cdot n_f) A_f = 0 \quad (4.17)$$

u_f is the velocity at the centroid of the face. The value of ψ_f is obtained by evaluating (4.15) at the centroid of a face. Next (4.17) is integrated with respect to time, in an interval $t \in [t^n, t^{n+1}]$. Before deriving the time stepping equation, the choice of the element which is used to evaluate ψ_f requires a little more attention. The choice of the element depends on the direction of flow: if u_f points out of the element e , then the element is the donor, and it is used to calculate ψ_f . However, if u_f point into the element e , then the element is an acceptor, and the neighboring element is the donor, so the neighbor is used to calculate ψ_f :

$$\psi_f(t) = \psi_{donor}^n + \nabla \psi_{donor} \cdot (c_f - u_f(t - t^n) - c_{donor}) \quad (4.18)$$

Now integrating (4.17), substituting (4.18):

$$\begin{aligned}
 V_e(\psi_e^{n+1} - \psi_e^n) + \sum_{f \in \text{Faces}[e]} \int_{t^n}^{t^{n+1}} A_f \psi_f(t) (u_f \cdot n_f) dt &= 0 \\
 V_e(\psi_e^{n+1} - \psi_e^n) + \sum_{f \in \text{Faces}[e]} \int_{t^n}^{t^{n+1}} A_f (\psi_{\text{donor}}^n + \nabla \psi_{\text{donor}} \cdot (c_f - (t - t^n) - c_{\text{donor}})) (u_f \cdot n_f) dt &= 0 \quad (4.19) \\
 V_e(\psi_e^{n+1} - \psi_e^n) + \Delta t \sum_{f \in \text{Faces}[e]} A_f (\psi_{\text{donor}}^n + \nabla \psi_{\text{donor}} \cdot (c_f - c_{\text{donor}} - \frac{\Delta t u_f}{2})) (u_f \cdot n_f) &= 0
 \end{aligned}$$

Each summand in (4.19) represents the flux through a face. If element e is a boundary element and the u_f is pointing into e , then the summand is replaced by the prescribed boundary condition, the time dependent flux $q(t)$. The time-stepping equation (4.19) is explicit, so the Courant No, $\frac{\|u\| \Delta t}{\ell}$, sets an upper bound on the time step size, $\frac{\|u\| \Delta t}{\ell} < 1$. ℓ is the characteristic element size in the direction of flow.

4.2.1.1 Computing the gradient of ψ

To complete the discrete form of the VOF equations an approximation to the gradients of the scalar field ψ is required. Traditionally, regular grids and finite-difference approximation is employed in dealing with the free-surface equations, [74, 81, 87]. Regular grids impose great difficulties in dealing with realistic geometries. On the other hand there are no full finite-element, unstructured grid implementations of second-order accurate schemes such as van Leer's.

A method for computing the $\nabla \psi$ on unstructured finite element meshes is introduced. The method first computes the initial local approximations to the $\nabla \psi$ at the centroid of each element. These gradients are known to be very oscillatory in regions where ψ changes rapidly. To suppress the oscillations, the gradients computed in the first step are adjusted to obtain a monotonically increasing or decreasing $\nabla \psi$. This process is also known as flux limiting. Alternatively, the $\nabla \psi$ can be obtained by the

finite element method by $\nabla \psi = G^{-1} B^T \psi$, where G and B were defined in equation (4.10). The first approach is employed. Using $\nabla \psi_e = g_e = [g_x, g_y, g_z]$, the gradient ψ_e in an element, the equation (4.15), and denoting the subset of the neighboring elements of this element which are adjacent to it by faces by n_i , where $i = \{1, \dots, M\}$, where M is the number of neighboring elements, the initial approximation to the gradient is computed by:

$$\begin{aligned} \psi_{n_i} &= \psi_e + (c_{n_i} - c_e) \bullet g \\ \begin{bmatrix} \psi_{n1} - \psi_e \\ \dots \\ \psi_{nM} - \psi_e \end{bmatrix}_{M \times 1} &= \begin{bmatrix} c_{n1} - c_e \\ \dots \\ c_{nM} - c_e \end{bmatrix}_{M \times 3} \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix}_{3 \times 1} \\ [\Delta \psi] &= [\Delta x][g] \\ [[\Delta x]^T [\Delta x]]_{3 \times 3} [g]_{3 \times 1} &= [[\Delta x]^T [\Delta \psi]]_{3 \times 1} \\ \Delta x, \Delta \psi &\text{ are known. Solve for } g \end{aligned} \quad (4.20)$$

Figure 4.8: The neighborhood of element e

4.2.1.2 Flux limiting by linear programming

The method of calculating the gradient g of the VOF function ψ as indicated before requires adjusting the gradients to insure monotonicity. Given c_e the centroid of element e , the flux limiting condition can be expressed as

$$\psi_{n_i} \leq \psi_{f_i} \leq \psi_e \quad \text{or} \quad \psi_{n_i} \geq \psi_{f_i} \geq \psi_e \quad (4.21)$$

where f_i indicates the i^{th} face of e and n_i is the element neighboring e by the i^{th} face. If we take the first inequality, we can re-write (4.21) by substituting (4.15) as

$$\psi_{n_i} \leq \psi_e + g_e \bullet (x_{f_i} - c_e) \leq \psi_e \quad (4.22a)$$

$$(\psi_{n_i} - \psi_e) \leq g_e \bullet (x_{f_i} - c_e) \leq 0 \quad (4.22b)$$

If we replace the terms in (4.22) with their absolute values, (4.22) applies for both cases in (4.21).

$$|(\psi_{n_i} - \psi_e)| \geq |g_e \bullet (x_{f_i} - c_e)| \quad (4.24)$$

Inequality (4.24) has to be true for all faces of e in order to satisfy the flux limiting condition.

Given $\hat{g}$, the approximation to the gradient, if (4.24) is not satisfied for all faces in the neighborhood of element e , one way to solve the gradient problem is to formulate it as a Linear Programming (Simplex) problem, [88], [89]. The following algorithm describes such a method:

1. Check if $\hat{g}$ satisfies (4.24) for all faces. If (4.24) is satisfied we are done.
2. Check if ψ_e is a local minimum or a maximum, if it is then we are done, set $g_e = 0$.
3. Formulate a linear programming problem (as explained later).
4. Solve for g_e , if there is a solution we are done.
5. If step (4) fails to resolve a vector g_e , set $g_e = 0$.

Assume $\hat{g}$ is the local approximation of the gradient, using the Euclidean metric, $L^1: \|g - \hat{g}\|_1 = \sum |g_i - \hat{g}_i| = \theta(g) = z$, where θ is the objective function written in terms of the gradient $\hat{g}$, and z is a scalar variable. The problem will be transformed into a linear programming example as follows; let us define a new variable $X = (g_x, g_y, g_z, z)$, and a mapping $z = F(X)$.

The set of inequalities (4.24) for all faces of e , will define a convex region in 2 or 3 space. The Euclidean metric, as stated above, defines the objective function (an inverted pyramid for 3D). To satisfy the flux limiting, we would like to minimize z in the L^1 norm. We can linearize $z \geq \theta(g)$ forming 4 new inequalities in 2D, 6 new inequalities

in 3D. The resulting system of linear equations will pose a valid linear programming problem as formally described below:

Independent variables:

$$X = (\tilde{g}'_x, \tilde{g}'_y, \tilde{g}'_z, \tilde{g}''_x, \tilde{g}''_y, \tilde{g}''_z, z), \text{ where } g_i = \tilde{g}'_i - \tilde{g}''_i \quad (4.25)$$

$$z \geq 0, \quad \tilde{g}'_i \geq 0, \quad \tilde{g}''_i \geq 0, \quad i = x, y, z$$

Maximize the objective Function:

$$F(X) = -z \quad (4.26)$$

subject to constraints (4.24) for each face of element e . The constraint (4.24) is linearized as:

$$g_e \bullet (x_f - c_e) \leq (\psi_{n_i} - \psi_e), \quad \forall n_i \text{ where } \psi_{n_i} \geq \psi_e$$

or (4.27)

$$-g_e \bullet (x_f - c_e) \leq -(\psi_{n_i} - \psi_e), \quad \forall n_i \text{ where } \psi_{n_i} < \psi_e$$

and constraints coming from the linearized L^1 metric, which will add 4 more constraints in 2D, and 8 more constraints in 3D:

$$\sum_{i=x,y,z} \pm(\tilde{g}_i - \hat{g}_i) \leq z \quad (4.28)$$

4.2.1.2.1 Standard form:

The algorithm implemented required that the LP described above be reduced to the standard form (sometimes known as the normal restricted form).

The constraints given in (4.27) and (4.28) are all in the form $f(X) \leq b_i$, $b_i \geq 0$. These constraints are reduced to the restricted form as $f(X) + s_i = b_i$, $b_i \geq 0$ and $s_i \geq 0$, where s_i are called "slack" variables". When (4.27) and (4.28) are transformed as described (4.29) is obtained:

$$\begin{aligned} g_e \bullet (x_f - c_e) + s_i &= (\psi_{n_i} - \psi_e), \quad \forall n_i \text{ where } \psi_{n_i} \geq \psi_e \\ -g_e \bullet (x_f - c_e) + s_i &= -(\psi_{n_i} - \psi_e), \quad \forall n_i \text{ where } \psi_{n_i} < \psi_e \end{aligned} \quad (4.29)$$

and

$$\sum_{i=x,y,z} \pm(\tilde{g}_i - \hat{g}_i) + s_i = z, \quad \text{where } s_i \text{ are slack variables}$$

4.2.2 Numerical example

The rotating cylinder problem is often used in the literature to demonstrate the extent to which the numerical method preserves a sharp interface, conserves mass and deals with artificial diffusion, Staniforth [80], Hansbo [56]. The problem consists of a prescribed constant angular velocity field centered at the origin of a square domain. The initial VOF value is prescribed such that the region over which VOF is set to 1 resembles a circular shape with a rectangular slot removed. The computational grid used in this test has 100x100 uniformly sized elements. The cylinder was rotated 180 degrees. The peak VOF value of 1.0 was preserved during the rotation. Despite some numerical diffusion, the interface did preserve its general shape quite well, see Figure 4.9.

(a)

(b)

Figure 4.9: Approximately 180 degree rotation of a region resembling a cylinder with a rectangular slot (a) A series of VOF contours at consecutive times illustrating the evolution of VOF (b) The VOF distribution along a line cutting through the cylinder and the rectangular slot at different angles of rotation

4.3 The energy equation

To model the thermal conduction problem, one more equation is added to the previously mentioned conservation equations of mass and momentum in fluids: the conservation of entropy⁶. However, in a viscous fluid this equation will not hold since the energy dissipation process is irreversible. Nevertheless, the conservation of energy holds; it is generally assumed that for such fluids, the changes in kinetic and internal energy of a fixed volume is balanced by the work done by the pressure in this volume and the energy dissipated by the viscous forces. Also, if the temperature distribution is not uniform in a volume, the flux of energy due to conduction should also be considered. Denoting the thermal conductivity by κ , the conductive flux is given by: $q_{\text{conductive}} = \kappa \nabla T$.

Landau & Lifshitz [44], provide a general form of the energy equation as a balance equation:

$$\underbrace{\frac{\partial}{\partial t} \left(\frac{1}{2} \rho u^2 + \rho \epsilon \right)}_{\text{rate of change of energy per unit volume}} = - \underbrace{\text{div} \left(\rho u \left(\frac{1}{2} u^2 + \varpi \right) - \underbrace{\underbrace{v \cdot \sigma}_{\text{energy flux due to internal friction}} - \underbrace{\kappa \nabla T}_{\text{energy flux due to thermal conduction}}}_{\text{total energy flux density}} \right)}_{\text{total energy flux density}} \quad (4.30)$$

where ϵ is internal energy per unit mass, ϖ is enthalpy. By substituting $\partial \rho / \partial t$ from the continuity equation (4.2), $\partial u / \partial t$ from the Navier-Stokes equation (4.3), and using the thermodynamic relations $d\epsilon = Tds - pdV = Tds + (p/\rho^2)d\rho$, where s is entropy per unit mass, (4.30) is re-written as:

$$\rho T \left(\frac{\partial s}{\partial t} + u \cdot \nabla s \right) = \text{div}(\kappa \nabla T) + \sigma \cdot \nabla u \quad (4.31)$$

⁶It is usually assumed that entropy is strictly increasing. Entropy is conserved only in an isentropic process. Here, isentropy is the result of the assumption of ideal fluid. In an ideal fluid, it is assumed that there is no heat exchange between different parts of the fluid. The flow is adiabatic. In adiabatic flow of an ideal fluid, the entropy of any fluid particle remains constant (see [44] page 3-4). Prigogine [90], provides a deep insight to irreversibility and to the concepts of conservation of energy and entropy.

4.3.1 Incompressible flow

Equation (4.31) is greatly simplified in the case of incompressible flows. Landau & Lifshitz [44], argue that if the fluid velocity is small compared to the speed of sound the variation of density with respect to the variations of pressure due to the motion of the liquid is negligibly small. The critical assumption of incompressibility is that density does not vary with changes in pressure. The density on the other hand may vary significantly due to non-uniform temperature. Therefore considering pressure constant, following substitutions are made in equation (4.31):

$$\begin{aligned}\frac{\partial s}{\partial t} &= \left(\frac{\partial s}{\partial T} \right)_p \frac{\partial T}{\partial t}, \nabla s = \left(\frac{\partial s}{\partial T} \right)_p \nabla T \\ C_p &= T \left(\frac{\partial s}{\partial T} \right)_p \rightarrow T \frac{\partial s}{\partial t} = C_p \frac{\partial T}{\partial t} \\ \frac{\partial T}{\partial t} + u \cdot \nabla T &= \frac{\kappa}{\rho C_p} \text{div}(\nabla T) + \frac{\mu}{2\rho C_p} \|\nabla u\|^2 \quad (\text{assuming } \sigma = \mu \nabla u)\end{aligned}\tag{4.32}$$

The flux of energy due to viscous forces can interpreted as a source of energy and added to the external sources if present, then the final form of the energy equation is:

$$\frac{\partial T}{\partial t} + u \cdot \nabla T = \frac{\kappa}{\rho C_p} \text{div}(\nabla T) + Q \tag{4.33}$$

4.3.2 Solidification modeling

The effect of solidification, the solid-liquid phase transformation, significantly influences the energy balance in a system, therefore it must be accounted for in the energy equation. The scope of this thesis is very restricted in its study of the details of solidification

modelling. Theory and applications can be studied directly from references such as Kurz & Fisher [91], Flemings [92].

Introducing a new scalar quantity F_s , the fraction of solid in a solid-liquid volume, the rate at which heat is released by solid-liquid phase transformations is $L \frac{\partial F_s}{\partial t}$, where L is the latent heat of fusion. The evolution equation for the F_s is usually derived from the binary phase diagrams for alloys and pure materials using methods such as the "Lever Rule", or "Scheil's Equation", etc., using temperature or enthalpy as a parameter. There are a variety of models with various levels of attention to detail, see [14-16, 91, 92]. The selection of the method is sensitive to the particular material being studied. With this addition the energy equation now reads as:

$$\frac{\partial T}{\partial t} + u \cdot \nabla T = \frac{\kappa}{\rho C_p} \text{div}(\nabla T) + \frac{L}{\rho C_p} \frac{\partial F_s}{\partial t} + Q \quad (4.34)$$

4.3.3 The discrete form of the energy equation

The Galerkin finite element method is used to discretize the energy equation. The energy equation is stated as an advection-diffusion problem with the associated initial and boundary conditions. Using the index notation as before, the equation is obtained as:

$$T_{,t} + u_i T_{,i} - \alpha T_{,ii} - \frac{L}{\rho C_p} [F_s]_{,t} - Q = 0, \quad \text{in } \Omega$$

initial values

$$T(x, 0) = T_o, \quad \text{at time} = 0$$

boundary conditions

$$T(x, t) = \bar{T}, \quad \text{on } \partial\Omega_{\text{dirichlet}}$$

and,

$$q \cdot n \quad \text{on } \partial\Omega_{\text{neumann}}$$

$$q = (uT - h^{\text{diffusive}})$$

$$h^{\text{diffusive}} = \alpha \nabla T$$

where

$$\begin{aligned}
 \partial\Omega_{neumann} \cup \partial\Omega_{dirichlet} &= \partial\Omega \quad (\partial\Omega_{neumann} \cap \partial\Omega_{dirichlet} = \emptyset) \\
 \partial\Omega_{inlet} \cup \partial\Omega_{outlet} &\subset \partial\Omega_{neumann} \\
 q^{inlet} \cdot n, \quad &\text{where } (u \cdot n) < 0 \quad \text{is } \partial\Omega_{inlet} \\
 q^{outlet} \cdot n, \quad &\text{where } (u \cdot n) \geq 0 \quad \text{is } \partial\Omega_{outlet} \\
 \alpha = \text{thermal diffusivity} &= \frac{\kappa}{\rho C_p}
 \end{aligned} \tag{4.35}$$

Using the finite element approximation for temperature is $T = \sum_{k=1}^N \tilde{T}_k \phi_k$, where $\tilde{T}$ are the coefficients of finite element temperature field. ϕ_k are the temperature basis functions. $\{\tilde{T}\}$ will denote the nodal coefficients vector. The weak form of the equation (4.35) is derived by the Galerkin method as follows:

$$\begin{aligned}
 \int_{\Omega} \left[T_{,i} + u_i T_{,i} - \alpha T_{,ii} - \frac{L}{\rho C_p} [F_s]_{,i} - Q \right] \phi_k &= 0 \\
 \int_{\Omega} T_{,i} \phi_k + \int_{\Omega} u_i T_{,i} \phi_k - \int_{\Omega} \alpha T_{,ii} \phi_k + \int_{\Omega} \left(\frac{L}{\rho C_p} [F_s]_{,i} - Q \right) \phi_k &= 0
 \end{aligned} \tag{4.36}$$

rewriting the second and the third integral as:

$$\begin{aligned}
 \int_{\Omega} \alpha T_{,ii} \phi_k &= \int_{\Omega} \alpha (T_{,i} \phi_k)_{,i} - \int_{\Omega} \alpha T_{,i} \phi_{k,i} \\
 &= \int_{\partial\Omega} \alpha T_{,i} n_i \phi_k - \int_{\Omega} \alpha T_{,i} \phi_{k,i} \quad (\text{using divergence thrm.}) \\
 \int_{\Omega} u_i T_{,i} \phi_k &= \int_{\Omega} u_i (T \phi_k)_{,i} - \int_{\Omega} u_i T \phi_{k,i} \\
 &= \int_{\partial\Omega} u_i n_i T \phi_k - \int_{\Omega} u_i T \phi_{k,i} \quad (\text{using divergence thrm.})
 \end{aligned}$$

the final equations including the necessary boundary and volume integrals are obtained:

$$\int_{\Omega} T_{,i} \phi_k + \int_{\partial\Omega} (u \cdot n) T \phi_k - \int_{\Omega} u_i T \phi_{k,i} - \int_{\partial\Omega} \alpha T_{,i} n_i \phi_k + \int_{\Omega} \alpha T_{,i} \phi_{k,i} - \int_{\Omega} \left(\frac{L}{\rho C_p} [F_s]_{,i} - Q \right) \phi_k = 0 \tag{4.37}$$

Substituting the approximate temperatures, $T = \sum_{k=1}^N \tilde{T}_k \phi_k$:

$$\begin{aligned} \left[\int_{\Omega} \phi_i \phi_k \right] \tilde{T}_{k,i} + \left[\int_{\partial\Omega} \phi_i \phi_k (u \cdot n) \right] \tilde{T}_k - \left[\int_{\Omega} \phi_i u_i \phi_{k,i} \right] \tilde{T}_k - \int_{\partial\Omega} h^{diffusive} \phi_k + \left[\int_{\Omega} \alpha \phi_{l,i} \phi_{k,i} \right] \tilde{T}_k \\ - \int_{\Omega} \left(\frac{L}{\rho C_p} [F_s]_{k,i} - Q \right) \phi_k = 0 \end{aligned} \quad (4.38)$$

Using the same matrix definitions as in (4.10), with correct material properties as coefficients, (i.e. thermal diffusivity for the K matrix instead of viscosity), the energy equation is written as (change in notation: T is substituted for $\{\tilde{T}\}$):

$$G^S \dot{T} + K^S T - N^S(U)T + f(T) = S \quad (4.39)$$

where:

$$f(T) = \left[\int_{\partial\Omega} \phi_i \phi_k (u \cdot n) \right] \tilde{T}_k - \int_{\partial\Omega} h^{diffusive} \phi_k - \int_{\Omega} \left(\frac{L}{\rho C_p} [F_s]_{k,i} - Q \right) \phi_k$$

The streamline-upwinding model can also be used in the advective-diffusive formulation for the energy equation. As velocity is increased, or the diffusivity is decreased, the Peclet Number, $Pe = \frac{uh}{2\alpha}$ (h is the characteristic grid size), plays an important role in changing the character of the energy equation from a parabolic equation (diffusion dominant), to a hyperbolic equation (advection dominant). The streamline upwinding schemes, as mentioned before, provide a method of dealing with the parabolic to hyperbolic transition.

$$K(u) = \int_{\Omega} \nabla \phi(x)^T \gamma(u) \nabla \phi(x) dV,$$

$$\text{where } \gamma(u) = \alpha I + \gamma \underline{u} \underline{u}^T, \quad \underline{u} = \frac{u}{\|u\|}$$

4.3.3.1 Time stepping

Similar to the time discretization used for the Navier-Stokes equation, the equation (4.39) can be discretized in time by:

$$(G^s + \Delta t \theta K^s) T^{n+1} = G^s T^n + \Delta t (S^n - f(T^n) - (1 - \theta) K^s T^n + N^s(u^n) T^n) \quad , \text{ solve for } T^{n+1}$$

The advection term is integrated with a forward Euler scheme

$\theta = 0$ is forward Euler, $\theta = 1$ is backward Euler, $\theta = 0.5$ is the Trapezoidal rule

4.3.4 Numerical experiments

A series of 1D advection-diffusion problems with dirichlet and Neumann-Dirichlet boundary conditions were checked against analytical results. They were found to agree well. The interested reader should also refer to Tjoa [93], who has performed a series of numerical experiments to validate the discrete form of the energy equation and solidification modelling with respect to the results obtained from experimental data.

4.4 Coupled differential equations for casting analysis

In casting analysis, the physics of melting and solidification, evolution of the free-surface, flow fields and the thermal field present an intricately coupled, complex and non-linear system of differential equations. Very few analytical solutions are known to exist for very restricted, usually one-dimensional models. This stimulated the development of multi-dimensional numerical procedures. The coupled equations may be classified at two different levels: 1) In casting analysis, most of the problems involve multiphase, multicomponent systems. These phases and components interact and they are strongly coupled. 2) Coupling at the level of state variables (i.e. u , p , T , ρ), and systems of equations. The relations of the state variables at this level of coupling are usually

decomposed with regard to some simplifying assumptions and grouped under physically meaningful expressions such as the Navier-Stokes equation, energy equation, free-surface equation, etc. Careful examination of the physics may identify the weaker couplings between these equations and may allow the equations to be decoupled.

Glimm [70] also indicates the vitality of coupling physical phenomena at different length and time scales; there are about 16 length scales between the atomic diameter (10^{-10} m) to the diameter of the earth (10^6 m) that define a region of our practical interest. Most theories become intractable when they require coupling between two adjacent scales. Glimm points to the fact that an increase in computational resources is not sufficient to deal with this problem, new methods and theories are needed.

4.4.1 Continuum equations

Modeling multiphase/multicomponent⁷ systems is of paramount importance to casting analysis. There are two generally accepted procedures to deal with such systems, [94, 95], [96, 97]: 1) Each phase occupies its own subdomain and is modelled with the corresponding conservation equations and coupled to others by proper boundary conditions applied at their interface. 2) The other method is the continuum approach where each differential volume can be occupied by any, all or combinations of the components or the phases; the need for separate conservation equations and the need for the boundary conditions at the interface, and therefore the interface tracking problem, are eliminated. However, the problem is now to estimate the constitutive properties of the composite material. This is known as the homogenization process.

---

⁷In thermodynamics a phase is a homogenous region, i.e. the three phases of a material: solid, liquid, gas. A component usually indicates an independent variable of a composition, i.e. Hydrogen (H) in Water (H₂O).

The implementation of multi-domain methods presents a difficulty in explicitly representing and tracking the interface, however it will preserve the sharpness of the discontinuity. The single-domain models are known to diffuse the interface. Although this effect can be minimized, it can never be eliminated. Aside from the details of implementation, there are other aspects of particular problems which may affect the choice of method; in the melting and solidification of alloys, the solid liquid interface is a mushy transition zone where there is no sharp interface. Single domain methods were shown to produce accurate results for such problems, [95],[13].

The equations for single domain formulations are based on the theory of thermodynamics of multiphase systems. However, simplifying assumptions are usually carried out: The components are assumed to be distinct and separable parts of the continuum. The continuum and conservation principles apply to each of the components individually. The mean behavior of the total system is governed by the same conservation principles. In a transient problem the composition of the mixture c , in a differential volume will generally change due to two effects, [44]: 1) The macroscopic motion of the media will move the volume as a whole without changing its concentration, (pure advection). Although the concentration at a point in space changes with time, the concentration in the traveling volume remains constant, (i.e. $\frac{Dc}{Dt} = \frac{\partial c}{\partial t} + u \cdot \nabla c = 0$). If the thermal and frictional dissipation is ignored, this motion is a thermodynamically reversible process. 2) A change in composition in a traveling volume due to molecular transfer (diffusion), reactions and phase change. These processes are irreversible and the conservation of the species equation is written as $\frac{Dc_i}{Dt} = \frac{\partial c_i}{\partial t} + u \cdot \nabla c_i = -\text{div}(j)$, where j is the flux, the amount of component c_i transformed per unit area in unit time. The conservation equations for the systems of components are expressed in terms of "total", or "mean" quantities such as "mean density". In some cases, these properties are

sometimes assumed to be weighted averages of the properties of each component, such as the mean density $\bar{\rho}$:

$$\bar{\rho} = \sum_{k=1}^N \rho_k f_k, \quad \sum_{k=1}^N f_k = 1$$

where a volume is a mixture of N components, ρ_k is the density of each component and f_k is the volume fraction. In cases where the quantity can not be expressed as a mathematical statement, empirical relations are substituted. The conservation laws such as the conservation of mass, conservation of linear momentum and conservation of energy apply without any modification with "total" material properties for the system. The material property can be very sensitive to the microstructure.

4.4.2 Systems of equations

The state variables such as u , p , T and ρ are coupled to each other in the systems of conservation equations. In dealing with coupled equations at this level, the nature of the physical problem and the mathematical formulation of the problem determine the coupled equations governing the phenomena. For example, if heat flow is studied in a domain with steady motion, the energy equation only governs the thermal flow; no correspondence between the energy equation and the Navier-Stokes equations of fluid mechanics is required because the flow is known, and need not be determined. In this sense, the Navier-Stokes and the energy equations are decoupled and the constant velocity field is used in the energy equation. However, in most realistic problems system of equations is coupled. Often, the coupling is critical. For example in a filling problem, the coupling between the Navier-Stokes equations and the free surface equations is important.

To analyze a wide range of problems under varying conditions, such as those encountered by the casting designers, methods and tools must be presented to fit their specific analysis requirements.

The following numerical examples demonstrate some coupled systems and the type of capability that can be constructed from the equations described above.

4.4.3 Navier-Stokes equations and free-surface evolution

An example of coupled conservation equations is used to simulate the filling of a casting. The Navier-Stokes equation and the hyperbolic Free Surface equation are solved together.

A good test problem is the "Dam-Break" problem. The problem initially appeared during early days of underwater nuclear testing. The explosion formed a vertical column of water in the air, which eventually collapsed. There were a series of studies and experiments to study the phenomenon, see Martin & Moyce [98]. The problem shares the same physics with the filling of a casting. The computational grid is 40x20 elements that has the dimensions of 4x2.5. The grid is non-uniform; elements near the boundary at the bottom of the domain are finer to capture the function variations through the boundary-layer more accurately. Initial velocity field is set to zero. Gravity acts in the negative y direction ($g=9.805$ in metric units). No-slip condition is applied as the dirichlet velocity boundary condition. Initial column of water is represented by setting the VOF to: $\psi_o(x,y) = \begin{cases} 1 & x \in [0,1], y \in [0,2] \\ 0 & \text{elsewhere} \end{cases}$.

The test was repeated with varying numerical setups and grid sizes. As the grid size was increased near the boundary, the no-slip condition dampened the spread of the

(a)

(b)

Figure 4.10: Collapse of a fluid column under the influence of gravity (a) The VOF ≥ 0.5 region and the velocity vectors at about $t=2.5$ seconds (b) A comparison of the simulation results with experimental data

surge front due to inappropriate modelling of the boundary layer. In either case the results matched the experimental data [98] quite closely, see Figure 4.10.

An additional experiment was performed by performing multiple steps of Free Surface equations while keeping the velocity field constant to increase performance, since the Navier-Stokes equations are considerably more expensive to solve than the Free Surface equation. Although encouraging computational economy was achieved during this test, the results were not as accurate. The numerical simulation of the surging front consistently lagged behind the experimental findings during this experiment.

4.4.4 Flow equations and thermal transport problem

The analysis coupling the effects of the flow equations and the energy equation are required for a multitude of problems in castings. Following examples provide a few of them.

4.4.4.1 Cold-shut problem

The freezing of a flow channel before complete filling is called the *cold-shut* problem in the manufacturing of shaped castings. The basic physics of the problem involves the interactions of several continuum phenomena.

To model the simultaneous flow and solidification that occur during the process of filling, the following equations are solved: Incompressible Navier-Stokes equations, hyperbolic free-surface equation and the energy equation with solidification kinetics. The problem also offers additional challenges because the energy equation can transform from a parabolic equation to a hyperbolic equation as the Peclet number increases with the narrowing of the flow channel. See Figure 2.10, demonstrating a numerical simulation of

the coupled analysis for a cold-shut problem in a horizontal pipe which is cooled from its walls.

4.4.4.2 Shrinkage analysis

Fluid flow due to shrinkage may be used to predict porosity and voids in castings. Pores are more likely to form at the liquid-solid front where the highest negative pressures occur.

Volumetric shrinkage during solidification due to the change in density between liquid and solid metal is $\beta = \frac{\hat{\rho}_l - \hat{\rho}_s}{\bar{\rho}}$, then the volumetric strain rate due to shrinkage on solidification should be $\beta \dot{f}_s$, where f_s is the fraction of solid as above. The rate of formation of pores $\frac{\rho_l}{\bar{\rho}} \dot{\epsilon}_{pore}$ contributes to the volumetric strain rate. This imbalance of the incompressibility condition in the Navier-Stokes equation (4.7), can be added to the governing equations as a distributed sink or source term. Pehlke [23], provided simple 2D solutions using the pressure-poisson equation, equation (4.12) to predict the velocity field introduced by the shrinkage due to solidification. The liquid-gas interface is then advected using a volume traction method. In order to analyze this phenomenon properly, accurate predictions of the Navier-Stokes, Energy and Free-Surface equations are needed. The shrinkage rate is predicted from the evolution of the fraction of solid in the energy equation, which is then used to solve the conservation of mass equation in Navier-Stokes to compute the velocities induced by the phase change.

Figure 4.11 presents the application of this model to casting of simplified cylindrical geometry. A similar phenomenon is observed in real castings where the Riser(Feeder) provides metal to other solidifying regions and as a consequence it is either completely consumed or covered with voids.

Figure 4.11: Formation of a cavity caused by the volumetric changes due to solidification. The cylindrical section loses heat on all the walls except the top. The solidification starts at the walls and propagates towards the interior. The velocity field induced by the volumetric change causes the sinking of the liquid-gas free surface

4.5 Summary

In this chapter, a mathematical framework for the study and analysis of casting was presented. The continuum models for the conservation equations provide a uniform treatment of the basic physics.

Although presented in an original and application oriented context, none of the mathematics is new. Most of the physics was known a century ago. Although mostly challenging to implement and apply to real problems, the computational methods were also known.

To this day, coupled systems of partial differential equations are traditionally handled and solved on a case by case basis; depending on the physics of the problem, a continuum model is derived, proper initial and boundary conditions are identified, numerical and computational challenges are examined and a framework to study the problem is developed. This method is arguably effective for problems with static description. However, very often the developed framework is not easily applicable and reusable for other problems, and sometimes not very easy to change either.

This problem of reusability and configurability are identified as major challenges in the remainder of this thesis. In manufacturing, analysis is critical to the design and development of a product; new problems are often described and frequently dynamically reconfigured and changed. The mathematical models are often exercised to their limits and require perpetual improvements and replacements. In their current state, analysis programs are inadequate to be a part of routine design and analysis.

Mathematics alone is not sufficient to provide analysis capability. The following chapters will discover other aspects of the problem related to software.

Chapter 5: Formalizing interactions - Multi Agent Systems for Solving Coupled Problems

Analysis of castings, like many other phenomena, involve solutions of coupled systems of partial differential equations (PDE). Traditionally, the way to deal with these equations is to write dedicated problem solvers for particular applications and specific equations. Finite element codes for stress analysis of solids, e.g., NASTRAN, finite difference codes for fluid mechanics, e.g. PHOENICS, are examples. Generally, the goals of these solvers are to provide fast and accurate solutions. Adaptability of the solvers to other problems and applications is rarely considered. Experience shows that it is unlikely that these codes can be extended to handle variations in mathematical models or be reusable components in a larger system, and any such capability can not be developed for these codes by patching. Redesign is needed.

The essential difficulty is properly decomposing a coupled problem and formalizing the subproblems and their interactions. The solvers should be defined at a very high level of granularity: In this thesis, solving a particular PDE is considered an

atomic¹ or primitive operation. A programming approach will be described where solvers for coupled systems of PDEs are expressed as programs of such atomic operations. These programs not only describe the interactions of the subproblems, but also can be designed to adapt to the requirements of a dynamic problem at runtime. Ultimately, this dynamic network of programs should behave as a single program with continuous behavior.

The primary motivation² of this approach is not to reduce computing time but to reduce software development effort and cost, to produce more robust software, and to make the software more readily usable by non-computer professionals. Improving expressiveness, modularity and reuse of mathematical software at a high level of granularity, and integration are the central concern. Given the time spent by an individual in manually integrating several software programs to solve a problem, the algorithmic complexity of individual software modules is not significant.

No doubt, a flexible analysis capability that could be configured and modified at run-time in response to a dynamic problem description as the problem evolves will also appeal to applications other than casting.

To summarize, the essential points must be emphasized: It should be recognized that design of the software is at least as important as the correct modelling of the physical problem. Our understanding of natural phenomena improves continually, so do the numerical methods. The software capability for analysis should be properly structured to evolve with advances in technology. Software should be modular; as better mathematical models and numerical methods are available, they should be adopted easily. It is important to recognize that the analysis capability can not be built as one large monolithic

---

¹ The term "atomic" is borrowed from database operations. An atomic operation maybe a series of events. The operation is executed in isolation and in order to be successful, all events must execute to completion. If any subtask fails, all tasks fail.

²This approach is similar in principle to the approach taken by visual programming environments.

software program. The analysis requirements and models for particular problems are also often very different. It is not possible to foresee every possible analysis requirement and build an all powerful piece of software. It is important to recognize that the descriptions of the real problems are dynamic. The solution of such problems can be considered as a combination of quasi-static problem descriptions and their solutions. The analysis software should also evolve with the description of the problem.

Standard parts, procedures and methods, uniform operations are very often used by engineers to study new problems, analyze existing ones or create new designs. This methodology provides a paradigm for the engineer to hierarchically build a new concept from more primitive parts using uniform methods which are well defined and understood (e.g.: circuit design, dynamic mechanical networks of masses, springs, dashpots, etc.; applications of finite element method). These methods are attractive for several reasons: They have a firm mathematical basis. There is no ambiguity about when they are valid and when they are not. They often provide algorithmic methodologies to study problems, therefore they are also more easily converted to computable systems. Unfortunately, successful formal systems (software), which are based on these models are limited to a few isolated applications: Electrical Engineers use network analysis and VLSI design tools as an integral part of their design and analysis process. Mechanical systems of piping, linkages, kinematic simulation systems, dynamic 2D/3D motion simulations of rigid bodies are examples of software that have matured into useful design aids.

These environments are ideal to support routine (variant) design and analysis: They provide a capability that restricts the space of available configurations. The availability of standard parts and rules of the assembly in the environment is not chosen by the user. They are chosen by the designers of the environment based on cost, current state of technology, etc. The selection and number of standard parts affect the productivity and creativity of the designer and also the usefulness of the environment.

Users should spend most of their time in modelling (which is the creative part of design). Other auxiliary operations such as integration of numerical analysis tools and models should be maintained by the infrastructure transparent to the user.

5.1 The model of integration

A decomposition of the structure in general coupled systems of equations is fundamental to the discussion that will follow. Our model of coupled problems is a set of abstract agents (subproblems) that cooperate by sending messages over an abstract network (the coupled problem). New agents may be introduced into this network, or existing agents can be removed. An agent can be replaced. New connections may be established or existing connections can be removed. The structure of the network is dynamic, its function is continuous.

An *agent* is an abstraction. One representation of an agent can be a software module. The agents are active because they can cooperate to solve larger problems than they could solve on their own.

A functional specification of agents is now described. A functional specification defines what an agent can do, but does not define how the agent does it. Any agent can be implemented in more than one way. By deferring detailed definitions of the structure of an agent generality of the concept can be preserved. A design specification giving implementation details will be presented later.

- **Subproblems:** The coupled system of problems can be represented by a set of agents where each agent is assigned a part of the work.

- ***The interface*³:** The behavior of an agent is accessed by the external world only through its interface. The interface characterizes the external view of an agent by the type of information it can provide and require from other agents. The external view of an agent can not characterize its internal structure or the environment it belongs to. The internal view of the agent is how the agent perceives itself. The external view of the agent is how it is perceived by an outside observer. Thus, an agent has both a private and public life.
- ***Organized behavior:*** All agents cooperate using a standard protocol for communication. Messages are the primary unit of interaction. Agents understand and respond to a set of messages. Their actions are based on their interactions with the group. The ensemble of agents define the behavior of the group. This behavior is generally complex, and can not be expressed as a trivial combination of individual behaviors of its components. Conway's game of life is an example. Similarly, the parts of a car do not describe the car.

Agents can represent a formal system and a natural decomposition of the problem into subproblems to deal with large complex phenomena. Formalized relations among the agents allow programmable connections and configurations of clusters of agents. As the connections and configurations of the agents change in a controlled manner, the program represented by this network also changes.

An environment based on this model contains a library of standard agents. Each agent in the library has well-defined behavior and a specification. The environment contains mechanisms to facilitate the cooperation of the agents. The cooperation has two dimensions: the management of the activities of agents in a group (control flow), and the domain-specific information that is shared and communicated between agents (data

---

³The terms "encapsulation" and "information hiding" are more commonly used in software engineering.

flow). In addition to performing an activity internal to an agent, an agent is also involved in transactions with other agents by providing and/or requesting information.

The definition of an abstract environment to describe a general set of components is:

Definition: The Environment (E) is a collection of standard parts called Agents (A), and a set of rules and procedures called a Protocol (P), describing the interactions (cooperation) and topology of the agents. $E: \{ A, P \}$.

Further discussion of the Agent Model will focus on the two components: The agents (A) and the protocols (P) for integrating and mediating the communications between agents.

5.2 The agent

One of the many reasons a single large program may not be made to adapt to dynamic problems easily and finally ceases to be valid is that it has an "implicit" mathematical model that can not be easily modified without a complete rewrite of the program. Sometimes, it is not even possible to identify the mathematical model of the program because it is too complex, see Brooks [38]. One strategy for overcoming this difficulty would be to "explicitly" represent the critical components of a large program. The likelihood of automatically adapting a program to the requirements of a dynamic problem is considerably higher in a structured environment. If the parts of the program that must be replaced/modified are explicit, and a method for adding/removing a new component that would meet the needs of the problem is defined, then the remainder of the problems are accidental.

The primary motivation for agents is to support the decomposition of a larger problem into subproblems and to establish an invariant relationship between the internal

environment of the agent and the overall task. The assumption that a complex problem can be partitioned is based on the concept of near decomposability, see Simon [99]. The context (semantics) of the agents abstract and separate the name and address spaces in a problem. Separate address and name space means that the agents have the freedom to define their own representation, semantics and symbols. The agent provides a natural abstract boundary between internal and external environments or private and public worlds.

The agent will carry out its functions as a response to interactions with the environment. The behavior of an agent can only be characterized in terms of its interface (external view) with the environment. A crucial implication of these properties is that the internal details of an agent does not depend on other agents, and that agents are truly heterogeneous.

An example will be used to clarify some of the concepts: The study of the continuum mechanics of casting problems reveals conservation equations and constitutive relations that must be solved simultaneously. In solving a mold filling problem with thermal evolution, the equations in (5.1) can be used (see equations 4.7, 4.13, 4.34 for more detail).

$$\begin{aligned}
 \rho \dot{u} - \mu \operatorname{div}(\nabla u) + u \cdot \nabla u + \nabla p &= f & \text{- Navier - Stokes} \\
 \operatorname{div}(u) &= 0 & \text{- Conservation of Mass} \\
 \dot{\psi} + u \cdot \nabla \psi &= 0 & \text{- Free Surface} \\
 \dot{T} + u \cdot \nabla T &= \alpha \operatorname{div}(\nabla T) + \frac{L}{\rho C_p} \dot{f}_s + Q & \text{- Energy}
 \end{aligned}
 \tag{5.1}$$

There are numerical methods to solve these equations individually, (see previous chapter). In many cases, a method that is suitable to solve the equations for one problem may fail to provide answers for another problem for reasons that could be obvious numerically and/or mathematically. Therefore, it can not be assumed that one method is

appropriate for all problems. Consider Figure 5.1(a), which illustrates the diagram of a high level parse graph depicting the coupling between the equations in (5.1). From Figure 5.1(a) no higher level structure is obvious. Even if a program was written to solve an equation in this form for a particular application, would it be possible to apply it to another problem? How can we integrate off-the-shelf programs that can solve some of the equations, such as the Navier-Stokes equation, that are coupled with other programs to solve the complete system?

In Figure 5.1(b), a set of the nodes in 5.1(a) corresponding to a subset of the variables are duplicated. Duplicating variables is quite common in distributed systems and distributed databases. The dashed lines correspond to a mapping between the separated variables (identity mapping in this case). Duplicating the variables and establishing a mapping between them, allows the names of the variables to be changed without affecting the rest of the system. This is inherent in the concept of separate name and address spaces. Boundaries around parts of the graph are introduced in such a way that none of the mappings between the separated variables are contained within a boundary (it would be a perverse to do so, since we separated them to include them in separate boundaries). Traditionally, these boundaries are naturally introduced by mathematical abstractions when the equations are based on natural laws or assumptions that some relations are weaker than others. The agents are characterized by these boundaries. Each variable within a boundary can be named and represented independent of how its connected to a variable within another boundary as long as there is a mapping between them. This is the independence of internal representations of agents. There is a clear division between internal and external worlds. From the perspective of numerical analysis, the agents resemble the natural partitioning of conservation equations, i.e., Navier-Stokes equation, Energy Equation, etc. From an implementation perspective, the agents provide the mechanism to encapsulate the details of the equations and numerical methods and provide an atomic operation to solve each PDE. The final stage is removing

Figure 5.1: A transformation of conservation equations to agents

the transparency of the agents: a way of implementing multiple abstraction levels, see Figure 5.1(c). The internal representation contributes to the overall behavior indirectly through the interface of the agent; the external behavior of the agent can only be characterized by this interface. An agent providing a constant field for velocity and pressure could be a valid replacement for a Navier-Stokes agent in a steady-state problem.

It is clear that depending on the nature of the problem, each agent may be replaced to provide a more accurate, faster or cheaper solution. Off-the-shelf programs, if they can fulfill the function of an agent, can be easily be integrated by adding a "*wrapper*" around them to manage their interactions with the environment.

An agent model is essentially a distributed and delegated model for problem solving; each agent produces what it is designed to produce and consumes what it needs without knowing where the information is coming from or how it was produced. It operates in isolation. The only way to cooperate with other agents is through messages.

The relationships among agents can be represented by a graph; the *agent* model includes a graph where the nodes are the computational components with well defined external interfaces (ports), and the edges which connect these ports are the communication links, see Figure 5.2. As a general topological representation, graphs can easily represent conceptually complex structures. However, the graph can be at least as complex as the underlying concept. It is trivial to build graphs that are difficult to understand and manage. It would not be reasonable to expect a simple representation for an inherently complex phenomenon. Representations for the graph other than agents can have implementation advantages and disadvantages but can not remove the essential complexity of the graph.

Figure 5.2: The graph analogy separating internal and external representation

Multiple viewpoints can help to manage some of the complexity of the graph, i.e. each node in the graph may represent a view point, or an autonomous structure like a department, individual, process, plant, subassembly, etc. Similar to the negotiations that take place between persons or groups during the evolution of a design, each node in the graph is responsible for a subtask. The subtask defines their perspective to the rest of the world. The internal representation (microscopic behavior) of the agent is irrelevant to its membership in the group. Ideally, each agent should be replaceable by any other agent that is able to fulfill the requirements of the former as a member of the group. Induced (node induced) sub-graphs, like the one in Figure 4.6, provide a hierarchical mechanism to implement multiple abstraction levels. An example of abstraction levels are demonstrated in Figure 5.1: In 5.1(a) there are a total of 49 nodes and 66 edges, which are reduced to the 3 nodes and 7 edges in 5.1(c). When the equations are viewed at a higher level of abstraction.

An agent can be recursive. Since the internal representation is hidden, an agent which is internally represented by a network of cooperating agents is externally not different from any other equivalent agent.

To summarize, agents have the following features:

- Internal and external representations are clearly separated. Interface to the external world is invariant. Internal representation is independent and dynamic.
- Agents interact and cooperate in an environment by exchanging messages. They adhere strictly to the rules of the overall communication protocol.
- They can have hierarchical structure and Multiple Levels of Abstraction.
- Multiple Viewpoints in a delegated cooperative environment are supported in a natural manner.

Introducing a formal definition of an agent increases the likelihood of over specializing the concept of an agent. However, the following high level definition can be made without too much risk and would further clarify the concept of an agent.

Definition: The agent a , is an encapsulation of a finite set of state variables $\mathcal{V}$, and a well defined implicit mathematical relation between the state variables, represented by the atomic operator $\mathcal{R}()$. Given a valid initial state $\mathcal{V}^n$, $\mathcal{R}$ maps the state variables to the next state of the agent, $\mathcal{V}^{n+1}$. $\mathcal{V}^n$ is in the domain of $\mathcal{R}$, $\mathcal{V}^{n+1}$ is in the range. $\mathcal{R}$ is a mapping, that always produces a valid state. The agent does not mutate its state space in an interval, i.e., the size of the set represented by $\mathcal{V}^n$. It can also reverse a state transition (in general not all functions are invertible, here it is meant as restoring the initial state).

Agent $a = \{\mathcal{V}, \mathcal{R}()\}$

if $\mathcal{V}^n = \{x_1^n, x_2^n, x_3^n, \dots, x_m^n\}$, $\mathcal{V}^{n+1} = \{x_1^{n+1}, x_2^{n+1}, x_3^{n+1}, \dots, x_m^{n+1}\}$

then $\mathcal{R} : \mathcal{V}^n \rightarrow \mathcal{V}^{n+1}$

At any instant $\mathcal{V}$ can be expressed as:

$$\mathcal{V} = I \cup O \cup C$$

where

I : input variables

O : output variables

C : constants.

Surely, an agent can have private variables. However they are not included in $\mathcal{V}$. The variables $\mathcal{V}$ define the interface of an agent. They do define the space of the operator $\mathcal{R}$ but they do not restrict $\mathcal{R}$ in any other way. $\mathcal{R}$ can be a composite mapping, however, only the terminal states are observable to the external environment of the agent.

5.3 The protocol for cooperation

The procedures and rules that regulate the interactions among a group of agents, or their cooperation, will be called a *protocol*.

Messages⁴ are the units of data for cooperation among agents. Message-passing is the form of cooperation. There are two forms of cooperation among agents: Control flow and data flow. Therefore, the messages are also functionally divided into two corresponding categories. Control is a directed network describing the order and the flow of communications in the system. A data dependency network describes the flow of data (results), the causal⁵ relations among agents.

Clearly, there are two distinct coexisting relations for cooperating agents each representing control and data flow respectively. Methods of representing these networks include ring , star , tree , or graph topology. The choice of topology affects the nature and complexity of the algorithms, see Raynal [105]. As the topology becomes more general (a graph being the most general), the nature of the interactions of the agents also becomes more complex. However, it is always possible to

---

⁴Messages and message passing are also used as formal concepts in programming languages and in the theory of distributed processing. However, there are several different opinions on how they came into use: Abelson & Sussman [100] point to Hewitt [101] who describes tail-recursion using message passing as the abstraction. The artificial intelligence community on the other hand [102], point to Hoare's CSP [103] as the originator of messages. In any case, the concept is well established and there is no ambiguity on what it means in the framework of object oriented programming and distributed processing. Messages were used long before they became parts of theoretical work, Simula-67.

⁵Causality, mutual causality, the relation of causation to interaction is the subject of much philosophical controversy. See Mario Bunge [104], for a rather interesting and enlightening discussion.

express a topology in terms of another by inserting an extra layer of abstractions and procedures. The topology of a network is generally dynamic and is expected to reconfigure and change itself.

5.3.1 Control

A tree structure for managing the flow of control has certain conceptual advantages. In a tree arrangement, agents only communicate with their parents, all communications must go through the parent node (message-forwarding). In solving differential equations, it can be safely assumed that there is a causal relationship between the agents, in other words the data flows in known directions. Assuming the existence of a pre-determined causality, there should be a serial order of execution that satisfies the causal relationship. Because of the concept of a "global" parent node in a tree, the execution order of the children can be easily arranged by the parent node. In a case where causality can not be predetermined (such cases arise in constraint problems [106]), a graph topology is more appropriate. However, even in constraint management, given a change in causality, directions are temporarily assigned to the edges in the graph. Then, usually a set of branching trees (acyclic subgraphs) deduced from the underlying directed graph determine the order of activation to satisfy the constraints. In general, if the original undirected constraint graph has cycles, there may not be a solution that satisfies all the constraints.

5.3.2 Data dependency

A graph is the natural structure for representing the data dependency among agents. Variables are represented by the nodes of the graph. Edges are the links between the variables. If the edges in this graph are directed, then the causality of the graph is

predetermined. Generally, it can not be assumed that this link is an identity mapping. Therefore, a more general mapping is often associated with a link. In this thesis, a constraint⁶ will denote a multi-directional mapping. Then, in constraint graphs, the edges have no direction, and the mapping should have an inverse since an edge can be assigned either directions while keeping the constraints satisfied. In general, all constraints may not hold and inverse transforms may not exist in cases other than simple algebraic and linear constraints.

Definition: Agent *a* is *connected* to agent *b* if there is a mapping *F*, such that $x = F(y)$. *x*, *y* are state variables of *a* and *b* respectively.

Definition: A constraint is an undirected connection. Agent *a* is *constrained* to agent *b* if there is a invertible mapping *F*, such that $x = F(y)$ and $y = F^{-1}(x)$. *x*, *y* are state variables of *a* and *b* respectively.

Definition: A connection is valid if the mapping is valid. A constraint is valid if the mapping and its inverse are valid⁷.

5.3.3 Consistency

Maintaining the consistency of the overall system is crucial. Locally, an agent should progress from a consistent initial state to the next consistent state (local/internal consistency). Globally, the overall state of the agent network should reflect the local state changes in a consistent manner (global/external consistency). Local consistency does not guarantee global consistency.

---

⁶ See Friedman & Leondes [107-109], who wrote a series of articles on the theory of constraints. Borning, Benson and Maloney [110] describe a constraint solver in a more recent study.

⁷ The concept of validity is not defined precisely here since different error measures and criteria may be applied. However, we assume that there is an error measure that must be satisfied. Proofs could be developed if assumptions were made to restrict variables and mappings to certain classes of mathematical objects. (i.e. vector spaces and linear transformations).

Consistency can be enforced in more than one way. One method is to enforce local and global consistency at all times. For example, if the local state of an agent changes, global consistency should be enforced immediately to bring the complete network back to a consistent state. If a globally consistent state can not be reached, then all agents should either restore to their original state, or the agents that cause the inconsistency should be removed from the group.

Another method would be to arrange a rendezvous point and enforce global consistency only when all the agents reach this point. Temporary global inconsistency would be allowed within an interval between two meeting points. There is a certain amount of risk with this model. If a globally consistent state can not be reached, either all the agents restore their initial states, or some intermediate state is negotiated and iterations resume until a globally consistent state is reached. In an iterative approach, global consistency is not always guaranteed. If a consistent state can not be reached, the whole group fails.

Local consistency is an agent's responsibility. Global consistency is maintained by conflict resolution. Different algorithms, and conservative schemes can be used depending on the demands of the application, (if a network of automated cash machines are the agents, then one certainly would pick the safest model to minimize risk). In a non-linear problem, an iterative algorithm to bring the system back to a known consistent state would be a form of conflict resolution.

In a separate name and address space environment with agents, it is conceivable that data that is duplicated between agents are not consistent. The problem is usually dealt with by establishing an ordering between the related events that guarantees consistency.

Definition: The action of an agent is *locally consistent* if it has successfully advanced to a new state that is valid.

Definition: The network of cooperating agents is *globally consistent* if all agents are *locally consistent* and all the connections in the data-dependency network are valid.

5.3.4 Messages and handling messages in agents

A message is a self describing entity and must be processed by an agent without knowledge of its origin. A framework for message passing can be established without sacrificing generality of the message structures. In a data dependency network, the message is a package, which is divided into a common internode language for communication and a common language for encoding/decoding domain-specific information that will be passed.

Agents that can cooperate must understand and share this domain-specific language. The syntax of the domain-specific grammar must be described to enable translation of external and internal representations and mapping of data to the internal structure. The semantics of the language is defined by the coupled problem.

A message handler is an integrated part of the agent. The message handler receives, processes and sends messages. A uniform model can be used for all message handlers in a network of cooperating agents. Although this is not necessary, it is good software engineering. The essential feature of a message handler is that it understands the common internode language of the agents and the language of domain-specific information. It will receive and translate this information for the agent, which in return performs certain operations depending on the message. The results of the actions of an agent will be communicated out by the message handler after it is encoded into a form

which is self-sufficient and understandable by all other message handlers. A message handler performs atomic operations, i.e., it can not receive or send another message until it has completed processing the current message.

5.3.5 Task and result sharing

The semantics of the topology of a network of agents can be interpreted in two distinct ways: task sharing and result sharing, see Smith [111]. In either instance, a network of agents cooperate. However, the semantics of cooperation are quite different.

Task sharing usually takes place in a group of agents where all have the knowledge to solve the problem but none has the resources to do it on its own. In task sharing agents are usually symmetric, (i.e., they have the same program).

Result sharing is different in that none of the agents has the complete knowledge needed to solve a problem on its own. Each agent solves a part of the problem and then they share the results to reach a complete answer. In result sharing agents are usually asymmetric (i.e., they have different programs). Using the agent model for coupled continuum mechanics problems where each agent solves one of the conservation equations and interacts with others to provide them with data they need is a form of result sharing.

Both forms of cooperation are allowed and they can coexist in the Design Assistant. In the case of task sharing, there is an abstract agent (a manager), who represents the task sharing network of agents. This agent can be involved in result sharing with other agents. From a user's perspective, task sharing is best done transparently and the cooperation is perceived only as result sharing.

5.4 An implementation of agents for coupled problem solving

The agent model was not solely designed and implemented for PDE solving. It has been implemented at several levels in the Design Assistant (such as the rigging design module). However, solvers for systems of coupled partial differential equations are chosen for describing an implementation of the agent model because they incorporate many of its aspects in a clear mathematical context.

An environment based on the agent model is implemented for solving transient and steady-state coupled continuum mechanics problems. These problems are encountered in many engineering applications including casting. The study of the continuum mechanics of casting problems involves one or more conservation equations and constitutive relations that must be solved simultaneously. It is possible to describe these mathematical forms as formal software components with their interactions as moderated communication protocols. Although not necessary, a unified representation of various conservation equations is convenient when implementing them as a formal system.

From the perspective of numerical analysis as performed by individuals, the agents may resemble the natural partitioning of solvers for conservation equations, i.e. Navier-Stokes equation, Energy Equation, etc.

The environment has a library of agents that can solve certain PDEs. Some of these agents are: Navier Stokes (Fluid Flow) agent, Energy (Heat Transfer) agent, Free Surface agent, Stress Analysis agent, Hydrogen Diffusion (Reaction-Diffusion) agent, Solute Diffusion agent, etc. As clearly indicated by their names, these agents are classified either according to the class of problems they solve or the conservation equations they represent.

Each agent represents an entire PDE solver as an atomic operation. Some agents share the same conservation equations and numerical methods, and even share the same program. The semantics of an agent can change with no corresponding change in syntax. It is the name space associated with the equation that distinguishes each type of agent. Specifically, it is the mapping of variable names between the external and internal worlds of the agent that distinguishes the agent. This mapping contains a set of associations that provide the link between the syntax and semantics. For example, a generalized advective-diffusion formulation was used to solve a variety of diffusion problems including solute, hydrogen diffusion, and the heat transfer problem. The generalized scalar potential field in the equation was named solute concentration, hydrogen content or temperature as appropriate. Naturally, the constants in the equation come from the relevant material properties (also contained in the name space) for each problem.

5.4.1 Structures for agents

Possible structures of the PDE agents are described including a class of variables, the components of an agent (PDE solver and a message handler, structures and protocol for messages, and a domain-specific language for agent interactions). Control flow and data dependency and exchange models are also described.

5.4.1.1 Variables

The first task is to define a set of classes of variables that are related by a general PDE. A PDE can be represented as a differential operator. The space of this operator includes a set of natural and essential boundary conditions and a set of initial values, possibly null, in addition to the parameters in the conservation equation. These variables are defined on a spatial domain, which will be called the geometry. For transient problems, a time

Agent	PDE	Variables	Inputs	Outputs	Boundary Conditions	Initial Values
Fluid Flow	Incompressible Navier-Stokes	Velocity, Pressure, Temperature, Fraction of Liquid, Fraction of Solid, Solidification Rate	Temperature, Fraction of Liquid, Fraction of Solid, Solidification Rate	Velocity, Pressure	Velocity, Pressure, Traction	Velocity, Pressure, Temperature, Fraction of Liquid, Fraction of Solid
Energy	Advective-Diffusion	Velocity, Temperature, Fraction of Liquid, Fraction of Solid, Solidification Rate	Velocity, Fraction of Liquid	Temperature, Fraction of Solid, Solidification Rate	Temperature, Convection, Radiation, Flux	Velocity, Temperature, Fraction of Liquid, Fraction of Solid,
Stress	Conservation of Mass and Momentum	Displacement, Temperature, Strain, Stress, Reactions, Load, Stress Potential	Temperature, Load	Displacement, Strain, Stress, Reactions, Stress Potential	Displacement, load, traction	Displacement, Temperature, Stress,
Free Surface	Second Order Hyperbolic	Velocity, Fraction of Liquid	Velocity	Fraction of Liquid	flux (inlet)	Velocity, Fraction of Liquid
Hydrogen	Advective Diffusion	Hydrogen Content, Stress Potential, Temperature	Stress Potential, Temperature	Hydrogen Content	Hydrogen Content, Hydrogen Flux	Hydrogen Content, Stress Potential, Temperature
Solute Diffusion	Advective Diffusion	Solute Content, Velocity, Temperature	Velocity, Temperature	Solute Content,	Solute Content, Solute flux	Solute Content, Velocity, Temperature

Table 5.1 Agents for continuum mechanics

interval is also associated with the solution. The definition of a state variable is extended to include other parameters (e.g. specific heat) and the usual field variables (e.g. temperature).

```

<Variable Classes> ⇒  <state variables>
                        <initial values>
                        <boundary conditions>
                            <essential boundary conditions>
                            <natural boundary conditions>
                        <geometry & material>
                        <time interval>

```

Figure 5.3: Classes of variables for PDE agents

The variables of a particular agent are a set of instances of the class of variables described above. An agent also specifies the causal relations among these variables because it specifies the mapping from input to output variables. Table 5.1 describes variables for a set of agents. These variables are used to define the interfaces of these agents. Internally, there is no restriction on the number of variables to be used, or how they should be classified. However, the agent must have the proper translation/mapping mechanism to convert to/from the interface variables.

5.4.1.2 PDE solver and the message handler

The agent has two major components: A PDE solver and a Message Handler. The PDE solver has a very restricted mode of operation and a well defined interface. Given a valid set of input data it will move or step to its next state. Its function is abstracted as a three tier operation: initialize, execute, report. These operations are sequential and each must succeed in order to proceed. Each operation is atomic and can not be interrupted.

The message handler is the main mechanism for facilitating cooperation. Although a common message handler program can be used for all agents, this is not necessary. The only requirement is that the message handler has the capability to process

the incoming and outgoing messages: this involves understanding an internode message protocol and a common language for encoding and decoding domain specific information packaged into the messages.

Figure 5.4: Components of an Agent

Once a message is received by the agent, the message handler decodes the information in the message. A message usually indicates either an operation to be performed by the agent (get/set data, initiate an activity, etc.), or communicates shared data. A message received will be either directly processed in the message handler (messages such as interrupt/signal, etc.), or the data in the message is mapped to/from an internal representation, or the message will be passed to the PDE solver to activate an operation. In general, it can not be assumed that the data as received is in a form that can be directly applied to the PDE solver. Therefore, the data should include enough information to allow the mapping, and the message handler should have the capability to do the data mapping, encoding and decoding.

The communication between the message handler and the PDE solver is strictly serial. While communicating with the PDE solver, the message handler can not be involved in any external activity.

The activities of passive agents (servers) can be grouped under three categories, expect messages, act on the messages and respond to messages. Active agents (clients) also send messages based on their local goals.

5.4.2 Messages

A message is an association of a message identifier and data, see Figure 5.5. Data in a message, once it leaves an agent, must contain enough information in it to be recovered by any other qualified agent without reference to the agent that sent the data. The message ID defines the syntax where internode protocols are described. Although, the syntax is shared between all agents so that it can be understood, the semantics of a message may differ between agents (e.g.: a message called *execute* would mean solving the Navier Stokes equation for the Fluid Flow agent, while the same message can be interpreted as writing a set of files in an agent that provides visualization capability).

$$\langle \text{Message} \rangle \Rightarrow \begin{array}{l} \langle \text{identifier} \rangle \\ \langle \text{data} \rangle \end{array}$$

Figure 5.5: Message structure

Encoding data into a message is an implementation issue. However, it is also a critical detail since it involves properly characterizing and efficiently representing domain-specific knowledge. A special grammar was developed to express continuum mechanics problems. Special constructions for geometry, and data (including state variables, boundary conditions and initial values), material descriptions were introduced. The message handlers are equipped with parsers and code generators to translate this grammar to the internal representation of the agents. See Appendix-A for a description of the grammar.

5.4.3 Solving a transient problem: managing data and control

As mentioned earlier, there are various representations for control flow and data dependency in a coupled problem. Here, the control flow is managed in a tree structure, while data dependency is represented by a graph.

Let us describe a set of agents, and their causal relationships. The flow of liquid metal in a channel which loses heat at its walls is considered. A mathematical model for the problem can be constructed, where mass and momentum are conserved by the Navier Stokes equation, the energy is conserved by the Energy equation that also accounts for the liquid to solid phase transition, and the evolution of the liquid-air free surface is solved with a hyperbolic equation. These differential equations are modelled by three agents: Fluid Flow agent, Energy agent and Free Surface agent. These agents have clearly defined interface variables and specification of their inputs and outputs, see Table 5.1.

Fluid Flow agents solve the flow equations for coexisting phases. Provided with the volumetric fractions of each phase, and the temperature field as inputs, it is capable of providing an approximation to the velocity and pressure fields. The Energy agent will provide temperature and volumetric fraction of solid, provided that it is supplied with the fraction of liquid and air, and the velocity. Similarly, the Free Surface agent will provide an answer for the volumetric fraction of air field, provided that it is supplied with the velocity field. All agents must also be supplied with the proper boundary conditions, initial values and the geometry. Each agent may have its own discretization of the geometry, (i.e. meshes can be dissimilar). The coupling among agents are described with a simple grammar, see Figure 5.6(a). In figure 5.6(b), the same graph representing the data flow is illustrated.

The simulation of a transient coupled problem involves the ordering and execution of the agents (control), result sharing (data dependency) and time stepping. The time step size used by agents are limited by the underlying numerical methods. Since there is no restriction on the choice of the numerical method, in general the time step size used by different agents is not expected to be equal. The control mechanism must synchronize the agents. The data mapping may involve interpolating between two consecutive time steps since results may not be available at the exact time specified.

Figure 5.6: Data dependency graph

5.4.4 The agent graph

A graph describing the relations of the agents and the data that is shared between them can be defined as $G = (V, E)$, where V and E are the sets of nodes (vertex) and edges respectively such that:

$$V \equiv \{n \mid n \in N_A \vee n \in N_d\}$$

where each node n is either a member of N_A , the set of agents, or N_d , the set of data. An edge is defined by the pair of nodes at its endpoints. The set of edges is described as:

$$E \equiv \{e \mid e \in E_1 \vee e \in E_2\}$$

$$E_1 \equiv \{\langle u, v \rangle \mid u \in N_A \wedge v \in N_d\}$$

$$E_2 \equiv \{\langle u, v \rangle \mid u \in N_d \wedge v \in N_d\}$$

E_1 are the edges between the agents and their data. These edges represent the ownership of the data by an agent. The directions of the edges E_1 describe if an agent requires or provides a data (required data are inputs, provided data are outputs). Members of E_1 are usually contained within a single name and address space. E_2 are the set of edges that represent the data mapping between the name and address spaces. The direction of these edges specify the direction of data flow between agents. There is a mapping implicit to the data flow link, which may involve interpolation in time and/or space to map one data onto the other. In Figure 5.7, an example of this graph is illustrated.

Figure 5.7: An agent graph

5.4.4.1 Agent activity and the versions of data in the graph

An agent can be activated only if all of its incoming data are provided (available). When activated, the agent produces a new set of output data with a new version stamp. A function can be defined to assign a version (ω), to the output data, i.e., the version of an output can be calculated as $\omega_o = \text{MAX}(\forall \omega_i) + \Delta$, where the Δ is a strictly positive non-zero increment and ω_i are the versions of the inputs. In this case an agent will assign a uniform version number to all of its output data.

When activated, an agent updates all of its output data to a new version. In the graph defined above, there is a forest of branching trees corresponding to the flow of data between the agents. Each of these trees correspond to a coupling among agents. If an agent is activated, the changes that are induced must be propagated along these trees. In Figure 5.8, the set of branching trees embedded in the graph described by Figure 5.7 are given.

Figure 5.8: Branching trees

5.4.4.2 Transient analysis

States of the agents N_A in the graph can represent transient solutions of a system of equations (state of an agent would be the values and versions of the data N_d that are owned by the agent). The version of data can be a time stamp. Then, the version increment would be the length of a time step.

A global time step can be defined such that all nodes N_A in the graph must complete the time step denoted by $[t^n, t^{n+1}]$. An agent would complete the time step $[t^n, t^{n+1}]$ by updating its output data given the history of its inputs in the interval $[t^n, t^{n+1}]$, i.e., provides a history for its outputs in the interval $[t^n, t^{n+1}]$.

The type of an agent and the coupled problem determines the algorithm to use, i.e., "transient analysis agent" and "time-step manager". In some cases, a partial or complete ordering in which the agents should work must also be specified.

5.4.4.3 Time-walk problem

Consider a problem where the data are pointwise, i.e., the versions denote a point in time t_n . Furthermore, assume that agents require all their inputs to have the same version. When an agent activates, it updates the versions of its outputs to be $t_{n+1} = t_n + \Delta$, Δ is the agent's local time step size. In general, all agents can not be made to have a uniform Δ . A problem can arise when an agent has input data from agents with different versions. In Figure 5.7, if some of the edges that cause the loops are removed⁸, the graph can be *topologically* sorted, i.e., no agent node that has an incoming edge from another agent node precedes that node in a linearly sorted list. It is easy to show that even in this

---

⁸A graph must be directed and acyclic (dag), in order to be sorted topologically., see [112] page 485. The agent graphs are not restricted to dags. However, this temporary restriction simplifies the demonstration of our problem.

simplified graph, given a consistent initial state, i.e., all data have the same version t_n , the system can never reach another globally consistent state t_{n+1} . The nodes D, and A do not have consistent incoming data. Unless those agents were provided with methods to resolve this conflict or the definition of a version and the conditions for the activation of the agents were modified, the graph would be in a deadlock, see Figure 5.9.

Figure 5.9: Topological sort of the agent graph for a time step

Clearly, this model is only suitable for a small set of problems where the agents have a uniform time step size. Improved models that can also deal with non-uniform time step sizes and nonlinearities due to coupling between the agents can be described. These models must maintain global consistency of data at each global time step. A class of time-walk models can be developed using different methods to mimic uniform time step size and maintaining data consistency at the end of the time steps.

One such model is based on defining a hierarchical time-walk with global and local time-walks. Globally, agents use a negotiated uniform time step size (the global time step), while locally they are "*decoupled*" and are allowed to use non-uniform time step sizes and multiple time steps to provide a solution for the global interval. All agents must mutually decide on a common end point for a global interval and as they all reach it with their micro local time intervals, they should resolve a globally consistent state for the overall graph.

If all agents have computed their outputs, then given these results, all other agents that use these outputs as their inputs must decide whether or not the new set of input data would significantly change their solutions, i.e. whether or not they should recompute their solution. This would lead to iterations. A time step is considered complete, when all agents agree that the current set of their inputs and outputs are consistent. In order to do this, an agent must have two kinds of information: First is the difference between the two sets of input data. Second kind is an information internal to the agent, i.e., the agent must know its sensitivity of its outputs to the input data and estimate whether the change is significant.

5.4.4.4 Managing time

The agents are expected to produce locally consistent results. Global consistency can be maintained either by a control strategy based on strict ordering of execution of the agents determined by the branching trees in the graph (Figure 5.8), or by the hierarchical time-walk (rendezvous mode). The global time step would be complete when all the agents complete their local times steps. The process is then repeated for the following time steps.

Forest of Trees - The directed graph describing the flow of data between the agents contains a forest of branching trees (variables of the agent are the nodes). Since multiple trees can be affected when an agent is activated, and agents in general do not have equal step sizes, then the difficulty in traversing each tree turns out to be in synchronizing the time between agents. Agents, in general will not be able to follow the time step used by their parent in the tree, this will eventually lead to a time shifting between the agents. Two possible algorithms for avoiding a time shift in tree traversal are:

- **Algorithm 1:** No agent exceeds the time step size of its parent. If an agent can not take a time step size as large as its parent, it will take multiple steps to reach it before the next level in the tree is executed. This algorithm is based on local decision making. An agent decides on a time step size using local (parent) information.
- **Algorithm 2:** A minimum time step size acceptable to all agents is predetermined and fixed for all agents during the traversal. This is a pessimistic algorithm based on global conflict resolution (all agents contribute).

Both solutions assume that an agent may take indefinitely small time steps; this is not possible due to stability limitations in the numerical methods. The second method also has the disadvantage of making very slow progress in time if any of the agents requires a very small time step. (It is true that probably this can not be avoided in any case, using the smallest time step for all agents will introduce additional computation which is otherwise unnecessary). The advantage of the methods is that they will insure global and local data consistency at all times.

Rendezvous - Another style of time stepping is to allow temporary global inconsistency. This method is based on negotiating a time interval among the agents, then meeting at the end of the interval and negotiating to reach a globally consistent state. Within the interval the agents are not required to be globally consistent. One possible algorithm is to negotiate the maximum available interval among all agents (i.e.: the maximum possible time step size any agent suggests), and identify the end of this interval as the meeting point. All agents proceed to the target with their own time steps without synchronizing. The data required by the agent can be provided as a function (set) which spans its current interval of time. The agent evaluates the function to acquire the data at any desired instant of time. However, this requires that the agent provides the data function before the dependent agent starts. It also requires a history for data be maintained. A less restrictive mode is to provide an agent with any available data even if this data was defined for an earlier time interval and an agent assumes that the data has

not changed since. The rendezvous method has the advantage of permitting concurrency among the agents. Different degrees of concurrency can be defined depending on the method of data flow used. When all agents reach the end of the global time interval, a consensus must be reached to guarantee global consistency. This may require iterations over the interval. Other methods can also be defined to guarantee global consistency. In some nonlinear⁹ problems, iterations are a natural method of negotiating consistency.

Obviously, the two distinct methods described are not the only control strategies that can be defined for cooperating agents. The important point is that the high level of abstraction in the agent model provides a framework in which these algorithms¹⁰ can be discussed and implemented as modes of cooperation.

The data structures may be implemented in more than one way. A simple model of control structures and data flow mechanism is described.

Figure 5.10: The tree structure for agents

5.4.5 Structures for managing cooperation

Agents are represented naturally by a graph as discussed above. However, implementing cooperation algorithms for agents by using trees is simpler because the hierarchy and an

⁹Interval arithmetic is often used to find a suitable test interval that contains a solution of a nonlinear system. However, there is no general numerical method to find such an interval, see Alefeld[113].

¹⁰Temporal logic is a widely accepted language in theoretical studies of algorithms for reactive systems and their behavior over time, see [114].

ordering relation is explicit between the levels of a tree, see Figure 5.10. A parent agent, which will be referred to as the manager, is defined. The coupled problem is represented by the manager agent. The manager is a complex agent since it represents a group. Agents which can not be further subdivided are called primitive agents. There is an associated tree corresponding to each subproblem. Each subproblem is represented by an agent (manager) that is the root of this tree. Subproblems can also be managers themselves. However, from a managers perspective primitive or complex agents are not treated differently.

Child agents are not allowed to send messages to each other directly. Although, the underlying data dependency is a graph, this graph is maintained by the parent agent. Message passing is only allowed between a parent and its children. The client/server relation between the children of the managing agent is transparently handled by the managing agent. In this model, the only client/server relation is between the manager and the child. A manager properly forwards the data to other children as required. The manager always has the role of the client and the children always have the role of the servers. This style of communication avoids deadlock: no two processes can have a conflict over which one serves the other, see Silberschatz [115]. It is possible to implement less structured and hierarchical control schemes (no particular restrictions on association and number of servers and clients). However, these models have considerably more complex rules of interaction, see Bernstein [116].

Complex agents do not have built-in PDE solvers, but instead, they manage their children. A complex agent does not have a local state other than the one represented by its children, neither does it possess any global information that is required by its children. Violation of these rules would be a contradiction in the agent model since some agents would not be able to function without others because their function can not be locally described. The reason for having a complex agent is to provide abstraction for a lower

level coupled problem (result sharing). They will also serve a double purpose later when a distributed model of computation is described. If distributed processing is available, it is possible to distribute an agent that is not capable of handling a problem because it lacks adequate resources. A complex agent abstracts a distributed agent in a coarse grained parallelism (task sharing).

A complete description of a protocol for solving all coupled problems is not realistic. A simple message protocol is developed for cooperation of agents to solve transient continuum mechanics problems. A generalization of this protocol must be avoided because of this bias in its design. Some of design criteria for a good message protocol are qualitative: they should not cause deadlocks in a program which is otherwise free of this problem [105]; they should be capable of describing complex behavior (deterministic or not); the functionality of messages should not overlap. There are also quantitative design considerations. Messages should be expressive, i.e., not too many messages should be needed to complete an operation.

Figure 5.11: Messages and the message handler

The messages are structured into two main groups: data exchange messages and messages for hierarchical control, see Figure 5.11. The messages for hierarchical control allow a manager to control the activities of the agent. In a heterarchical (distributed) control schema, these messages would not be necessary since the agent can be given a

decision making mechanism to decide locally when to execute, or expect and provide data.

The hierarchical mechanism was simpler to implement since a distributed control strategy requires dynamic scheduling, which is not trivial. A data exchange protocol is not specific to any control strategy; it facilitates query and modification of the local states of the agents. Messages which are encoded with data must be self-contained. For example, in a message carrying the temperature field in a casting, the temperature can not be interpreted without the discrete representation of the geometry (mesh). Therefore the geometry is also provided in the message.

The message handler is composed of a transaction processing part and data mapping module. The transaction processing module implements the message protocol that was described. The message handler interacts with the PDE solver sequentially: it will not process a new message until the current one is processed. A data mapping module provides the capability to understand the content of messages. It parses the data which is encoded in a domain-specific language. Since the syntax and geometry in the message may not be exactly the same as the agent's own, the agent maps data onto its internal structure and geometry.

Agents can assume the role of both a client and a server at the same time, see Figure 5.10. An agent is a client if it manages other agents. If it is managed by another agent at the same time, it is also a server. Depending on its current role as a server or a client, the states of an agent can be described as in Figure 5.12.

Figure 5.12: The states of an agent. (a) The states of the agent assuming the role of a server. (b) The states of an agent assuming the role of a client.

The states of a server agent, as described in Figure 5.12(a), are the following:

- **Wait** - The state of no activity. The agent listens to messages that are directed to itself or requires it to take an action. When it detects such a message, it will proceed to its next state.
- **Receiving stage** - The agent receives data that it requires to take an action. This data is usually specified by the agent itself. However, it can also be a generic message requesting the agent to provide a time step size estimate or to provide a specification of the data it requires.
- **Performing the local task** - This is the primary function of an agent that it must execute in order to perform its part in solving the overall problem. For a computational agent this could mean solving an equation. For an agent that is also a client itself, this could mean managing other agents.

- *Providing Stage* - The agent provides data to its client. This data is usually computed as a result of the agent's current activity.

The states of a client agent, as described in Figure 5.12(b), are the following:

- *Negotiating a time step* - This is the state that the client agent requests information from its servers to negotiate the new time step size for the next global time step.
- *Forwarding stage* - The servers are connected to each other in a data flow graph, see Figure 5.7. However, the servers are not allowed to communicate directly. This state forwards the data between the servers that require it from each other. The client may also send requests to its servers to collect information that it needs.
- *Collecting stage* - The client collects the data that it requested from its servers.

5.5 Distributed/Parallel processing

The agent model is primarily designed to manage coupling and integration. However, because of its inherent distributed and delegated design, it also provides a natural abstraction for coarse grained distributed and parallel processing.

Agents share many of the problems and techniques of distributed/parallel processing. Scheduling, distributing data and control, network traffic, load balancing, data consistency, mutual exclusion, dead-lock, reliability, fault tolerance, etc., must also be apprehended in both an agent model and a distributed/parallel processing model.

The capability to do distributed/parallel processing is a pleasant side-effect of the agent model. It was studied only in a limited framework. All algorithms and the details of the implementation can be easily extended for distributed processing by implementing

message handlers which can send/receive messages across processes. The control mechanisms that were described earlier were serial in nature.

With distributed processing, there is an increase in computational resources; the coupled problem is no longer limited to a single machine, or a single process. There are more processors and more memory. Using the same serial control schemes, the simplest implementation would be to solve larger problems by distributing each agent as a separate process.

It is very appealing to introduce limited forms of concurrency to improve performance. Two such opportunities can be created based on task and result sharing.

The first case is result sharing where individual problems are executed concurrently. The coupled problem solving should be deterministic, i.e. solving the same system of equations twice in a row should not produce two different answers. This is not a problem since the PDEs considered here are causal systems. Mathematically, if the system can be solved, there is an implied order (precedence¹¹ graph). This does not mean a problem can not be solved concurrently.

In a transient application using a rendezvous style control strategy, agents can be executed concurrently within a time slice. Each agent's life span is characterized by a sequence of time steps, $([t_1, t_2], [t_2, t_3], [t_3, t_4], \dots, [t_{n-1}, t_n])$. Usually, agents do not agree to use the same time steps. Therefore, the rendezvous mechanism is used to synchronize the agents. The start and end points of the sequence are fixed and each agent is allowed to choose its own sequence in this interval. Agents that reach the end must wait until permission to start another global time step is negotiated. Consistency is maintained as before. Within a time step various scheduling¹² models can be used. An agent can

¹¹In a precedence graph (directed) where the nodes represent processes, the edges represent a precedence relation. If there is an edge from a to b, the task b must wait until task a is finished.

¹²Preemptive processes are usually discussed in problems of scheduling. A process is nonpreemptive if it must run to completion once it is started. If it can be interrupted and resumed later, it is called preemptive.

diverge on an incorrect solution path due to inadequate initial data. Preemptive scheduling obviously is advantageous in this case. The agent can be interrupted (aborted) and restarted from a different initial state.

The second application of concurrency is in task sharing. Dividing an individual problem into smaller pieces and running the pieces concurrently. This form of parallelism is more developed and studied in literature. Domain Decomposition [118, 119] technique is an example.

A distributed version of the agent model is prototyped in the Parallel Virtual Machine (PVM) framework on a network of heterogeneous workstations, see [120].

5.6 Relation to other work

As in all research, parts of this work were inspired by and build on the works of many other research and existing concepts. The most important software concepts are object oriented paradigms, distributed and parallel processing, computer integrated and flexible manufacturing technology, manufacturing languages, standards such as the Open Systems Interconnection Reference Model (OSI), distributed artificial intelligence and other multi-agent systems. These are discussed below.

In these discussions, precedence is based on the priority of the process. There are messages such as wait and signal(resume). The concept of interruption in preemptive scheduling is also useful to our model. Representing the time slice as a time series allows the process to be interrupted, rewound or resumed within a time step (preemptive), or, they can be made to run until the completion of the time slice (nonpreemptive). Recent theoretical studies and evaluations of alternative scheduling models can be found in references such as Coffman & Garey [117]

5.6.1 Object oriented programming

In object oriented designs, the objects are the units of expression that personify real-life objects, people, concepts, etc. Objects introduce abstraction boundaries in the design space. The objects perform a computation by interacting with each other. Encapsulation and interaction are essential to the object oriented paradigm. The dimensions of the object-oriented models are listed as encapsulation, reactivity (interaction), distribution and concurrency, Wegner [121].

In object oriented programming, objects have a local state (private memory) and a set of operations. The operations specify the type of access to the local state of the object. The operations (methods) define the behavior of the object. The objects only allow access to their behavior and representations through these methods. Objects usually interact by message passing.

There are variations of object-oriented implementations such the prototypical actors [122], class based systems, inheritance mechanisms [34], which are classified as object-based, class-based and object-oriented systems by Wegner [123]. There are disagreements on this classification and the conditions of being object-oriented [124]. In general, we will refer to all of them as object-oriented (OO) models.

The agent model has many similarities to Object Oriented Programming (OOP):

- *Local state* - Objects have a private memory. In the object oriented terminology they are called instance variables. Instance variables are truly local. The only way to manipulate them is by using the object's own operations. They are not visible from outside the object. Agents provide the same mechanism by separating their internal and external representation.
- *Structured interaction mechanism* - Similar to the agents, objects are heavily restricted in the way they can interact. The only mechanism of interaction is by sending and receiving messages. An object's state can not be changed in any

other way. An object can not be made to take an action without evoking one of its methods.

- *Message passing* - Most object oriented systems use a form of message passing as the primary abstraction for communication among objects.
- *Encapsulation* - Similar to objects, the agent provides an encapsulation mechanism that separates the representation and the function of the agent. There is no restriction on how the behavior is implemented.
- *Nondeterminism* - Agents do not prescribe any specific order in which their operations should be evoked. Similarly, objects do not impose an order in which their methods are send. There is no specific input-output relation. The reaction of the agent or the object to a message depends on its current local state.
- *Modularity* - OOP systems are strongly open architectures and support modularity and incremental development. The design naturally allows new object and classes to be added to existing systems.

There are also differences. Agents do not possess all aspects of OOPS, e.g., they do not possess inheritance. Although, it may be argued that it is possible to derive new agents from the existing ones, this is not sufficient to support inheritance. Agents are truly heterogeneous: the internal representation is not restricted to adhere to a uniform programming model. Agents are a programming model. A standard library of agents are used to build systems and programs that can solve coupled problems. Problems are expressed in terms of agents. Agents are very coarse entities corresponding to high levels of abstraction. Agents provide a clear conceptual model for the application domain, the relations among the objects, the multiple viewpoints and abstraction levels that are an integral part of the environment. OOP does not directly provide such mechanisms. This shortcoming of OOP is recognized and research on extended object models that specifically deal with these improvements is ongoing [125].

5.6.2 Distributed/Parallel processing

Distributed and parallel algorithms are differentiated from the traditional sequential approach by the fact that multiple control structures exist simultaneously. The system is decomposed into modular elementary computational units (processes). The activities of these units must be managed. This gives rise to the concept of a network of processes and communications. Concepts and algorithms (protocols) are developed based on these abstractions.

Concepts such as local states, encapsulation, communication modularity are also common to distributed and parallel processing. Mutual exclusion and election (distribution), scheduling, fault tolerance, exception handling, reliability, deadlocks, consistency and gaining consensus are important issues in distributed/parallel processing. Most of these topics are also relevant to the agent model. Many alternative algorithms and abstractions [102, 126, 127] have been developed to deal with these problems. They can also be applied to problems in the agent model. Distributed processing is an active research area, for a review see Raynal [105].

Messages and message-passing are often used as the means of communication, Hoare [103]. Algorithms and frameworks for reliable transaction methods and communication protocols are developed based on message passing, Moss [128].

Hillis [129], describes a different kind of a computer where processors can be connected in any desired pattern; the programming strategy is based on arranging these processes in a way that models the natural structure of the problem. Agents seek to accomplish a similar goal by providing a framework where agents can be connected to solve a problem. Agents attempt to provide abstractions that are as natural to the application domain as possible.

Heterogeneous computing [130, 131], is an approach where an application runs in an environment that incorporates several kinds of autonomous machines to solve a problem. This approach allows the distributed parts of the problem to run on machines which best fit their requirements and it provides a uniform framework to define the problem. Parallel Virtual Machine (PVM) [120], and Linda [132] are examples of such frameworks. A distributed version of the agent model was implemented within the PVM framework.

From a conceptual point of view, distributed processing and object-oriented paradigms share some familiar abstractions, see [121, 133]; there are multiple autonomous entities with local states that cooperate by sending messages. There are encapsulation boundaries, and resources and local state of an entity is inaccessible except through messages. Transparency and encapsulation are almost synonymous terms generally preferred in the distributed processing and object-oriented paradigms respectively. However, the abstraction concerns are driven by different motivations; in OOP the encapsulation is logical, whereas in distributed processing it also has a physical aspect. Merging the two paradigms seems logical and useful since many applications could benefit from it. However, in practice this has proven to be a very hard problem, see Meyer [133]. Concurrent Object-Oriented Programming (COOP) [134, 135], and object-oriented parallel languages such as Mentat [136], are some of the current attempts in merging the benefits of both worlds. These languages provide the natural benefits of data abstraction, software modularity and code reuse of OOP along with different ways to provide a capability for distributed-volatile objects, remote-local message passing, object mobility across address spaces at different levels of transparency, see Moss [137], Bennet [138]. However, currently these languages are very much at the research level. Issues such as how to use objects and classes in a distributed system, systematic approaches to design COOP languages, and concurrency mechanisms are still under discussion.

There is also other related research in the area of mathematical software. Most common forms of parallelism and concurrency are discussed under the topics of Domain Decomposition [118, 119]. An implementation that envisage using these methods as a part of the agent model was discussed in a previous section. There is also work on software libraries for solving differential equations that can be parallelized at different levels of granularity [139].

5.6.3 Artificial intelligence and multi-agent systems

A branch of Artificial Intelligence (AI) is concerned with distributed and decentralized methods of problem solving. Global problems are solved with collaborative actions of a group of entities. Collaboration is defined as sharing local information to solve a global task. The group of entities (data and control) can be distributed logically, physically or geographically. This branch of AI is usually called Distributed AI (DAI), see Bond & Gasser [140]. Bond & Gasser distinguish between Distributed Problem Solving (DPS), Smith [111], and Multi Agent Systems (MAS) as examples of agents in DAI. In DPS agents are generally more uniform (common language and semantics), and more or less powerless to act alone. The whole group is needed to solve the global problem. In MAS agents are more independent and do not share a language. They have local goals and they cooperate for reasons of efficiency and results sharing but potentially can solve the whole problem alone.

Decentralized AI, is very similar to Distributed AI but it is concerned with the activities of an agent in a multi agent environment, [141-143]. The agent is assumed to be intelligent and reactive to its environment. It acts intentionally according to its local goals. The primary motivation of this type of research is to study the structure and behavior of an individual agent, i.e., to understand social behavior.

Smith [111], discusses distributed problem solving and the "contract net" concept in cases where more than one agent can perform a given task. Archon [144] an ESPRIT program project, describes an application of a multi-agent system to an electricity management problem; detailed discussion of the architecture is provided in the book. Werner [145], discusses cooperation algorithms for agents in a multi-agent environment.

5.6.4 Manufacturing automation systems and languages

Integration of manufacturing equipment involves linking, scheduling and operating a variety of automatic equipment in a highly heterogeneous, dynamic and irregular environment in an efficient, profitable, flexible, safe and dependable manner. It is a good research problem but is very complex and hard to solve.

There are various algorithms and systems, both commercial and research products. Duffie & Piper [146], give examples of some of these systems and claim the superiority of heterarchical and local decision mechanisms for control of manufacturing cells. They provide a comparison of several methods on a spectrum from centralized control to highly hierarchical structures to distributed heterarchical models. On a related topic, Hatvany [147] argues for cooperation models based on heterarchic models and local intelligence.

The Manufacturing Automation Language Environment (MAPLE), is another interesting approach, [148]. The main purpose of MAPLE is to provide a common support facility for multiple, independent, programming languages for manufacturing devices and control.

5.6.5 The OSI reference model

There are several available architectures and hierarchy of protocols that support communications network architectures and processes. The Open Systems Interconnection Reference Model (OSI) is an international standard (ISO 7498) to provide an incentive to the industry to standardize these architectures, see Heap [149].

The OSI model contains seven layers. Each layer provides an interface for the layer above and uses the layer below to perform more primitive tasks. These layers are defined such that the coupling between the layers is minimized and the resulting layers handle tasks that are very different. The seven layers of the OSI model with decreasing level of abstraction are:

- *Application layer* - The user interface layer that provides access to the services of the lower layers
- *Presentation layer* - The layer designed to abstract multiple representations of data and free the application layer from syntactic issues.
- *Session layer* - The layer that provides control structures to manage communications between cooperating applications.
- *Transport layer* - The layer that facilitates transparent and reliable transfer of data.
- *Network layer* - The layer that abstracts the network transmission technology.
- *Data-Link layer* - The layer that provides reliable transfer of data across physical links.
- *Physical layer* - The layer that abstracts the physical medium.

The first three layers of the OSI model are closely related to the concepts in the agent model. The application layer is the analog of the intelligent design assistant. The presentation layer and the session layer are concerned with distributed and negotiated problem solving similar to that in the agent model. The presentation layer abstracts the

multiple representations and view points, and the session layer is concerned with the cooperation of distributed applications.

5.7 Summary

In this chapter, coupled problems were presented from the view point of integration and software engineering.

The problem was conceptualized in a distributed and delegated model called the agents model. The agents model describes a natural decomposition scheme for large, inherently distributed complex phenomena by providing encapsulation boundaries and representations for cooperation. The internal and external representation issues are clearly separated. The concept of local state and reduction of global information is introduced. Modes and mechanisms of cooperation are discussed.

An implementation for coupled continuum mechanics problems in the framework of agents is presented. Possible implementation agents and cooperation mechanisms are described. Topological structures for control flow and data flow are presented. Messages, message protocols and message handlers are developed. Distributed and parallel processing is introduced as an extension of the agent model.

The relation of this work to other research was discussed in a brief survey.

Chapter 6: Agent Examples

This chapter provides demonstration examples for the agent model. The purpose of these examples is to illustrate decomposition of the problem into agents, structure of the agents, internal behavior of the agents, their external interface, method of interaction between the agents and the behavior of the coupled system. Agents provide a method for structuring software and problems, they are not a substitute for good mathematics and modeling.

There are three examples. The first example demonstrates a weakly coupled system with autonomous agents. The second example is a solver for a strongly coupled system and demonstrates a use for agent hierarchies. The third example demonstrates a system that has complex agents and a coupled system of partial differential equations are modelled.

6.1 Example I - Autonomous agents

Van der Pol's equation is defined as:

$$\ddot{u} + (u^2 - 1)\dot{u} + u = 0 \quad (6.1)$$

which can be reduced to two first order ordinary differential equations by making the substitutions $u = x$, $\dot{u} = y$, where u , x , y are scalar functions of time t :

$$\begin{aligned} F(x, y): \quad \dot{x} &= y \\ G(x, y): \quad \dot{y} &= (1 - x^2)y - x \end{aligned} \quad (6.2)$$

In this example, two autonomous agents can be defined to represent F and G respectively, i.e., neither agent has information about the other's internal function. This system of equations is a case of weak coupling, i.e., $F(x, y)$ has no x terms. The model of interaction can make use of this property. Also, it is only an accident that this problem can be solved without introducing any private variables, i.e. variables other than x , y and t . However, this does not restrict the discussion.

6.1.1 Agent A_1 :

- *Function* - $F(x, y)$
- *State variables* - $\mathcal{V} = \{x, y\}$. The input and output are $I = \{y\}$ and $O = \{x\}$ respectively. Since the agent has two variables and one equation, if $\{y\}$ is data (input), it can provide an answer for $\{x\}$.
- *Interface* - Variables $\{x, y\}$ are accessible to the external environment of this agent.

6.1.2 Agent A_2 :

- *Function* - $G(x, y)$

- *State variables* - $\mathcal{V} = \{x, y\}$. The input and output are $I = \{x\}$ and $O = \{y\}$ respectively. Since the agent has two variables and one equation, if $\{x\}$ is data (input), it can provide an answer for $\{y\}$.
- *Interface* - Variables $\{x, y\}$ are accessible to the external environment of this agent.

6.1.3 The coupling

The agents A_1 and A_2 are coupled to solve the van der Pol's equation, see Figure 6.1.

Figure 6.1: The coupling between agents A_1 and A_2

6.1.4 The behavior of the system - solving the equations

The transient response of the van der Pol's equation (6.1 or 6.2) can be computed using numerical integration methods such as the Runge-Kutta¹ method. A study of this equation can be found in [150].

The Runge-Kutta method was used by each agent to compute the transient response of the parts (F and G) of the system, see equation (6.3). The agents solve an initial value problem, therefore given a set of initial values, each time step of an agent is locally consistent. Furthermore the agents can be arranged such that at each time step they execute in the order $A_1 \Rightarrow A_2$.

¹See [89] page 569 for a description.

$$\begin{aligned}
 A_1 &: \int_{t_n}^{t_{n+1}} F(x, y)|_{y=y_n} dt \quad \text{initial value } x_n = x_o \\
 A_2 &: \int_{t_n}^{t_{n+1}} G(x, y)|_{x=x_{n+1}} dt \quad \text{initial value } y_n = y_o
 \end{aligned} \tag{6.3}$$

A typical time step $[t_n, t_{n+1}]$ can be done as follows:

1. Set initial values $x_n = x_o$ and $y_n = y_o$.
2. Provide the data required by the agent A_1 - (Data flow)

Set $y_1 = y_n$ (Input)
 Set $x_1 = x_n$ (Initial Value)
3. Activate the agent A_1

Internal operation - Integrate $F(x, y)$ for the interval $[t_n, t_{n+1}]$.
 The output is x_{n+1} .
4. Provide the data required by the agent A_2 - (Data flow)

Set $y_2 = y_n$ (Initial value)
 Set $x_2 = x_{n+1}$ (Input data)
5. Activate the agent A_2

Internal operation - Integrate $G(x, y)$ for the interval $[t_n, t_{n+1}]$.
 The output is y_{n+1} .

6.1.4.1 The solution

The results provided by the agents are compared with the results obtained by integrating the system of equations simultaneously. Figure 6.2 is the trajectory of the solutions of the equations. Figures 6.3 and 6.4 provide the transient solutions of the system. Solid lines are computed by solving the equations simultaneously, the overlaid points are the solutions computed by the agents.

Figure 6.2: The trajectory of the solutions of the van der Pol's equation

Figure 6.3: The periodic solution for $x(t)$

Figure 6.4: The periodic solution for $y(t)$

Of course, an agent could be constructed that uses the Runge-Kutta method to integrate the equations simultaneously.

In solving the van der Pol's equation we were not concerned with how the agents solve the equations. No agent needs to perform any global operation. The main concern is the coupling structure to provide the desired behavior.

The same kind of alternating iteration framework can be applied to other systems, even when x and y are higher dimensional vectors, provided that the coupling pattern is the same. There may be other cases where different forms of iterations produce faster convergence and more accurate results. However, this would not change the agents in any way. The time-walk algorithm is replaced with a different one to allow such methods.

However, when there is a stronger coupling, i.e., simultaneity of the equations are genuine and the equations must be solved together, the agent structure that was just described would not suffice.

6.2 Example II - Strongly coupled agents

In general, if the coupling is sufficiently strong, the mathematics may require equations to be solved simultaneously. The equations can not be simply decomposed and then locally solved. However, it is still possible to decompose the system in a hierarchical manner. A higher level agent can perform the activities on all data that must be operated on simultaneously. The topology of the agents can reflect the simultaneity and the dependence structure.

Consider the following system of equations:

$$Kx = b \quad (6.4)$$

where K is a matrix of coefficients, not necessarily constants, x and b are the unknown and the right hand side vectors respectively. In the following discussion linear equations are used for simplicity, however the same structure also applies to non-linear problems. The main concern is to deal with the simultaneity of the equations.

Let us assume that due to some difficulty, the matrix K is too hard to deal with. It can be partitioned into simpler matrices that can be dealt with as follows:

$$K = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, x = \begin{bmatrix} u \\ v \end{bmatrix}, b = \begin{bmatrix} p \\ q \end{bmatrix} \quad (6.5)$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}$$

Now it is possible to separate the system of equations into two sets:

$$\begin{aligned} F(u, v) &: A u + B v = p \\ G(u, v) &: C u + D v = q \end{aligned} \quad (6.6)$$

F and G can be assigned to two agents. However, in general if the system is strongly coupled, the solutions of an alternating iteration method described in the first example will not always converge. In this case, a method using global information about the system is more suitable. It is still possible to maintain the partitioning of the equations without paying the price of the large K matrix. The compromise is an iterative method in the case of some linear equations. Strang [151], describes semi-direct and iterative methods to solve such systems of equations. The idea is to start with some initial guess for x , then every iteration leads to the next guess (hopefully better) as follows:

$$x_{n+1} = x_n - M^{-1}(K x_n - b) \quad (6.7)$$

where M is a simpler matrix (sometimes diagonal) than K , and not every M would work for all equations, see Strang [151] for a discussion of choices for M . A new parent agent can be designed to perform equation (6.7). The two previous agents can compute the

residual $r = (K x_n - b)$ by forming $(A u + B v) - p$, $(C u + D v) - q$, and their contributions to the M matrix. These contributions are then assembled by the parent agent to perform the next iteration. We can now introduce the following form and define agents:

$$\begin{aligned}
 F(u, v) &: A u + B v = p \\
 G(u, v) &: C u + D v = q \\
 S(u, v, r, m) &: \begin{bmatrix} u_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} u_n \\ v_n \end{bmatrix} - \begin{bmatrix} 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \quad (6.8)
 \end{aligned}$$

where

$$\begin{aligned}
 m_{1ii} &= A_{ii}, \quad m_{2ii} = D_{ii} \\
 r_1 &= (A u + B v) - p, \quad r_2 = (C u + D v) - q
 \end{aligned}$$

6.2.1 Agent A_1 :

- **Function** - $F(u, v, m_1, r_1)$: The agent computes the diagonal entry $m_1 = A_{ii}$ and the residual $r_1 = (A u + B v) - p$
- **State variables** - $\mathcal{V} = \{u, v, m_1, r_1\}$. The input and output are $I = \{u, v\}$ and $O = \{m_1, r_1\}$ respectively.
- **Interface** - The variables $\{u, v, m_1, r_1\}$ are accessible to the external environment of this agent.

6.2.2 Agent A_2

- **Function** - $G(u, v, m_2, r_2)$: The agent computes the diagonal entry $m_2 = D_{ii}$ and the residual $r_2 = (C u + D v) - q$
- **State variables** - $\mathcal{V} = \{u, v, m_2, r_2\}$. The input and output are $I = \{u, v\}$ and $O = \{m_2, r_2\}$ respectively.
- **Interface** - The variables $\{u, v, m_2, r_2\}$ are accessible to the external environment of this agent.

6.2.3 Agent A_3

- **Function** - $S(u, v, m_1, r_1, m_2, r_2)$: The agent assembles the full residual vector and the full diagonal and computes the new value of u and v .
- **State variables** - $\mathcal{V} = \{u, v, m_1, r_1, m_2, r_2\}$. The input and output are $I = \{m_1, r_1, m_2, r_2\}$ and $O = \{u, v\}$ respectively.
- **Private variables** - This agent has three private variables: r , the full residual vector. m , the full diagonal vector. x_{old} , the previous values of u and v . Note that m and r are global vectors.
- **Interface** - The variables $\{u, v, m_1, r_1, m_2, r_2\}$ are accessible to the external environment of this agent.

6.2.4 The coupling

The agents A_1 , A_2 and A_3 are coupled to solve the simultaneous system of equations.

Figure 6.5 describes the data flow.

Figure 6.5: Data flow

Figure 6.6: Control flow

There is a strict dependency between the agents. Neither A_1 , A_2 nor A_3 can solve the equations on its own. A_1 and A_2 compute parts of the equations. They can do their parts independently (possibly in parallel). However, even if they could join their efforts,

they are not capable of providing an answer. A_3 , on the other hand, is capable of computing information about the equations simultaneously and doing global iterations. However, it needs other agents to compute its parts. Since they truly depend on each other, they can be given a hierarchical control structure. A_3 controls A_1 and A_2 to perform part of its operations and calls them repeatedly until the system converges, see Figure 6.6.

6.2.5 Managing interactions of the agents

The question of convergence and the rate of convergence are paramount to this method. They depend on the exact nature of the equations and the choice of M . These questions will not be dealt here. A typical time step $[x_n, x_{n+1}]$ can be done as follows:

1. Set initial values. Start with an initial guess $x_0 = \{u_0, v_0\}$.
2. Activate A_3
  - 2.1 Activate Sub-Agents A_1 and A_2
    - 2.1.1. Provide the data required by the agent A_1 - (Data flow)
Set u and v (Input provided by A_3)
    - 2.1.2. Activate the agent A_1
Internal operation - $r_1 = (A u + B v) - p$ and $m_1 = A_{ii}$
    - 2.1.3. Provide the data required by the agent A_2 - (Data flow)
Set u and v (Input provided by A_3)
    - 2.1.4. Activate the agent A_2
Internal operation - $r_2 = (C u + D v) - q$ and $m_2 = D_{ii}$
  - 2.2. Provide the data required by the agent A_3 - (Data flow)

Set m_1	(Input data provided by A_1)
Set r_1	(Input data provided by A_1)
Set m_2	(Input data provided by A_2)
Set r_2	(Input data provided by A_2)
  - 2.3. Perform internal operations of A_3

Internal operation - Assemble m from m_1 and m_2 $\begin{bmatrix} 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix}$.

Internal operation - Assemble r from r_1 and r_2 $\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$.

Internal operation - $x_{n+1} = \begin{bmatrix} u_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} u_n \\ v_n \end{bmatrix} - \begin{bmatrix} 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$

Internal operation - Compute $error = |x_{old} - x_{n+1}|$

2.4. If error is less than some ϵ stop the iterations. Otherwise set $x_{old} = x_{n+1}$ and repeat 2.

The agent A_3 manages the other two. As a part of its operation it will manage the interactions and activities of its subagents when it is activated. It is easy to see that the subagents themselves can also manage other agents.

Consider the following numerical example. A linear system of equations is solved with the above method.

$$2u - v = 8$$

$$2v - u = 2$$

In matrix form the problem is stated as:

$$Kx = b \Rightarrow \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \end{bmatrix}$$

Then the sub matrices and the diagonal matrix M of K are:

$$A = [2], \quad B = [-1], \quad C = [-1], \quad D = [2]$$

$$P = [8], \quad Q = [2],$$

$$M = \begin{bmatrix} 2 & \\ & 2 \end{bmatrix}$$

$$m_1 = 2, \quad m_2 = 2$$

For this problem, it can be proved² that this iterative method converges for positive definite matrices and the error is halved at each iteration. Using the above method the following results were obtained for this problem:

N	u	v
0	0.0	0.0
1	4.	1
2	4.5	3
3	5.5	3.25
4	5.625	3.75
5	5.875	3.8125
6	5.90625	3.9375
7	5.96875	3.95312
8	5.97656	3.98437
9	5.99219	3.98828
10	5.99414	3.99609
11	5.99805	3.99707
12	5.99854	3.99902
13	5.99951	3.99927
14	5.99963	3.99976
15	5.99988	3.99982
16	5.99991	3.99994
17	5.99997	3.99995
18	5.99998	3.99998
19	5.99999	3.99999
20	6.0	4.0

The same method can be applied to systems of equations where the K matrix and the vector x are larger. Also, this discussion can be extended to deal with non-linear systems of equations with little modification.

The method described in this example can be naturally implemented as a distributed algorithm for solving systems of equations where the sub-agents execute in parallel (there can be more than two agents if the matrix is partitioned into smaller parts). Another aspect of this example is that the subagents A_1 and A_2 are quite similar. In fact, they share the same algorithm and their programs can be symmetric. With a little work, two instances of the same agent can be used for A_1 and A_2 . However, the important point here is that a system of simultaneous equations can be solved by introducing an agent where global operations were performed. Simultaneity and the dependence structure is embedded in the topology of the agents.

²See Strang [151] page 406.

6.3 Example III - Fluid flow and free surface

The previous sections provided examples based on simple mathematical problems. Usually, the programming effort required to simulate such systems is not extensive. It may be preferable to write simple dedicated programs to solve them. There are however problems that are not easy to program. The programming and testing effort required for such systems can be so prohibitively expensive that they justify a more sophisticated software approach.

Coupled fluid flow and free surface analysis can be such a problem. In Chapter 4, the equations and the numerical methods were described in detail. The parametric space of these problems are very large. For a typical casting simulation, the dimension of the state space can approach millions of variables. Geometry, materials, boundary conditions, initial values, etc., must also be defined along with the more obvious variables such as velocity and temperature. The following equations describe the partial differential equations (see equations 4.7 and 4.13 for more detail).

$$\text{Fluid flow} \quad : \quad \rho \dot{u} - \mu \operatorname{div}(\nabla u) + u \cdot \nabla u + \nabla p = f$$

$$\text{Free surface} \quad : \quad \dot{\psi} + u \cdot \nabla \psi = 0$$

where u , p , ρ , μ and ψ , the velocity, pressure, density, viscosity and fraction of liquid fields respectively. The physical domain (the geometry), boundary conditions and initial values must also be defined. FEM based numerical methods are used to simulate fluid flow and the free surface equations.

A superficial inspection of the variables indicates a one way coupling: The free surface agent requires velocity field. However, this coupling is obviously not sufficient. The values of ρ and μ strongly depend on ψ . For convenience, ρ and μ can be attributed to materials that can be provided as library calls, which are functions of, along

with other parameters, ψ . Also, we can define a set of variables BC , M , Geo , which correspond to boundary conditions, material definitions and geometry.

The agents described below omit a set of variables, which are otherwise necessary, to reduce complexity. In fact, the representation of these agents is a problem in itself. A special purpose language was developed for representing continuum mechanics problems. It is explained in Appendix A.

6.3.1 Fluid flow agent A_{ff}

- *Function* - $NavierStokes(u, p, \psi, BC, M, Geo)$
- *State variables* - $\mathcal{V} = \{u, p, \psi, BC, M, Geo\}$. The input and output are $I = \{\psi, BC, M, Geo\}$ and $O = \{u, p\}$ respectively. Given initial values and boundary conditions for velocity and pressure, the fluid flow agent provides the velocity and pressure within a time interval using an FEM approximation.
- *Interface* - Variables $\{u, p, \psi, BC, M, Geo\}$ are accessible to the external environment of this agent.

6.3.2 Free surface agent A_{fs}

- *Function* - $Hyperbolic(u, \psi, BC, Geo)$
- *State variables* - $\mathcal{V} = \{u, \psi, BC, Geo\}$. The input and output are $I = \{u\}$ and $O = \{\psi\}$ respectively. Given initial values and boundary conditions for velocity and ψ , the free surface agent provides ψ within a time interval.
- *Interface* - Variables $\{u, \psi, BC, Geo\}$ are accessible to the external environment of this agent.

6.3.3 The coupling

The coupling between the agents A_{ff} and A_{fs} are similar to the coupling in the first example. The agents are designed to be autonomous. These agents certainly require other

parameters. However, this will not limit our discussion. The simplified data flow is described in Figure 6.7.

Figure 6.7: The coupling between agents A_{ff} and A_{fs}

6.3.4 Solving the equations

The transient response of the coupled fluid flow and free surface equations can be computed with a time-walk algorithm identical to the one described in Example I. A typical time step $[t_n, t_{n+1}]$ can be done as follows:

1. Provide initial values and boundary conditions for u and ψ .
2. Provide the data required by the agent A_{fs} (Data flow)
Set u (Input from fluid flow)
3. Activate the agent A_{fs}
Internal operation - Integrate the hyperbolic equation for the interval $[t_n, t_{n+1}]$.
The output is ψ .
4. Provide the data required by the agent A_{ff} (Data flow)
Set ψ (Input data from the free surface)
5. Activate the agent A_{ff}
Internal operation - Solve Navier-Stokes equations for the interval $[t_n, t_{n+1}]$.
The output is u and p .

Note that the individual processes may require implicit time-stepping, but the implicitness does not extend to coupling, i.e., the equations solved in the implicit time-stepping are not coupled. In other problems, a fully implicit method might be needed so that on each time step a simultaneous system in (u, ψ) might have to be solved. This would require the hierarchical agent structure described in the second example.

3 of/de 3

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISION^{SM} RESOLUTION TARGETS

6.3.5 Numerical example

A two dimensional geometry was used. The geometry was composed of a 36×36 grid. The velocities at the walls of this boundary were prescribed to zero. Initial velocities were zero everywhere. The gravity was specified to act downward along the vertical axis. Initial fraction of liquid field was defined such that it resembled a pool of liquid and a large droplet standing still away from the pool. The materials were such that the density difference between the liquid and the gas resembled water and air. Figure 6.8 is the evolution of fraction of liquid and the velocity field. Darker regions are the liquid.

Figure 6.8: The droplet

" It's only a small matter of programming ...

In the programming community, one often hears this epigrammatic comment following a lengthy technical discussion of some difficult problem of software design. ..."

A Small Matter of Programming
1993, Bonnie A. Nardi

Chapter 7: Postscript

This chapter is a brief overview of what has been accomplished in this thesis and what issues are outstanding.

7.1 Contributions

The contributions of this thesis can be summarized in the following viewpoints:

▷ *Casting design*

- Development of the Virtual Foundry : A software environment that integrated the capability to analyze fluid flow, heat transfer and solidification into the casting design process.

- A method or strategy for using the basic analysis capability to support casting design.

▷ *Mathematics and physics*

- A framework to solve coupled equations describing the basic physics relevant to the casting process.
- Solving complex conservation equations are treated as primitive operations.

▷ *Software*

- A formal system to represent interactions in coupled problems and to integrate software to support mechanical design.
- A programmable analysis system that is flexible and expandable. This system can be configured (programmed) to model the behavior of different problems. New modules can be added to provide additional capability.
- The agent model is a distributed, heterogeneous and asynchronous model, which allows inconsistencies and incompleteness.
- Software structures for describing multiple viewpoints and abstraction levels are provided with the agent model.

▷ *Design theory*

- An example of analysis capability integrated into the design process is provided.
- A model for integrated design software environment is provided.

7.2 Outstanding questions

This thesis exposes the tip of the iceberg for a large set of problems in casting, design theory, and software to support design environments. A detailed and complete account of all outstanding issues is not possible. Nevertheless, a brief summary within the boundaries of our knowledge is:

▷ *Casting design*

- The validation of software in an industrial environment is as important as the software itself. How the casting community will accept the virtual foundry is to be seen. End-user's perspective and needs are yet to be discovered.
- The casting physics involves many physical phenomena. This software only deals with a small subset of these events. It is expected that more detailed analysis and models will be needed for designing castings as the benefits from this level of sophistication diminish.

▷ *Software*

- Describing interactions among agents is a hard problem. It is not possible at this point to provide a general representation or a language to program the procedure of interactions or the behavior of a system using this procedure. The capability provided in this thesis is restricted to some classes of interactions, such as solving transient problems, or iterative systems, etc.
- Task-specific control structures, such as the time-marching, can be primitives in a possible representation for interactions of agents.
- Resolving conflict is key to dealing with inconsistent and incomplete systems. In this thesis, negotiation between agents is done in a specific context. To agree, agents must have a capability to do local cost-benefit analysis and specific local goals. It can be necessary to do a global analysis of conflicts. This is naturally tied to defining and representing a design function at local and global levels.
- Distributed and parallel implementations of the agent model is not complete and it is expected to have its unique problems.

▷ *Design*

- Describing context is a hard problem. If two programs were to communicate and cooperate, their contexts must be compatible. Otherwise, the meaning of cooperation is unclear. The cooperative behavior would be random at best. Communication or integration of software must be stated within a context. This in our view is not possible in a general framework. Automation and integration

of software will probably always exist in domain-specific areas where the context is common to all agents (such as casting).

- Version management, such as archiving designs, is necessary if expert help is to be provided.

7.3 An overview

The intention of this overview is to provide an overall awareness of the problems discussed in this thesis. It is neither a complete summary of the thesis nor a conclusion.

7.3.1 Virtual foundry

The idea of constructing software environments to support designers (or the design process) is not new. It has also long been realized that integrating analysis capability and software design environments into the design process is a hard problem. There have been various ineffective attempts to provide general purpose answers, such as design databases, for integrating software into the design process and manufacturing.

In the first chapter, a taxonomy of difficulties that limit the creation of software environments for design was presented. This thesis sought answers for these problems in the domain-specific context of casting while attempting to develop results that can be transferred to other domains.

Deep representations of the casting design problem with models of the casting process based on basic physics provided a concrete basis for the design environment. Formal representations for agents and their interactions were developed to implement the design environment in software.

Learning for design with software is an alternative to learning with experiments. Software is based on mathematical models of casting physics, and provides formal methods for the study of castings. Conceptual models developed using software tools are different from those developed from experiment. These are two very different views of the casting reality.

The main concept of the virtual foundry environment is an Intelligent Design Assistant. This assistant is an autonomous software entity that is capable of making decisions about low level activities in the design process such as setting up and managing details of analysis. The design assistant provides help to the casting designer by doing analysis that frees him to concentrate on creative and higher level aspects of the design. The Virtual Foundry is an environment where the casting designer and the intelligent design assistant interact.

7.3.2 Basic physics of casting

Most physical phenomena described in this thesis involve interaction among two or more complex events. A foundation for the physical background, the underlying mathematics and their discretized forms for a numerical treatment relevant to the analysis of castings was developed. Although presented in an original and application oriented context, almost¹ none of the mathematics or the computational methods were new.

The basic physics of castings including multi-phase fluid-flow and free-boundary evolution; heat transfer and solidification; and porosity development were described. This is a small but important set of the macroscopic physical events that occur in castings. The capability to analyze a casting and its behavior, even if it is limited, is an important step

---

¹The use of linear programming in the implementation of van Leer's method for free boundary flow for unstructured grids is original.

forward. The examples of analysis demonstrated the use of numerical simulations to support casting design.

Mathematics alone is not sufficient to provide software capability that can be integrated into design environments. In their current state, available analysis programs are inadequate to be a part of routine design and analysis. This thesis achieved greater adaptability of the solvers to other problems and applications.

7.3.3 Software

The essential difficulty is properly decomposing a coupled problem and formalizing the subproblems and their interactions. These programs not only describe the interactions of the subproblems, but also can be designed to adapt to the requirements of a dynamic problem at runtime. Ultimately, this dynamic network of programs should behave as a single program with continuous behavior.

Coupled problems were presented from the view point of integration and software engineering. The problem was conceptualized in a distributed and delegated model called the agent model. A heterogeneous set of agents solves the coupled problem. Their interactions are made possible by providing system level support for common communication protocols.

Agents are implemented for problem solving. This abstraction is a level above individual programs, processes, networks or machines.

This model of coupled problems is a set of these abstract agents that cooperate in an abstract network (the coupled problem). The agent model describes a natural decomposition scheme for large, inherently distributed complex phenomena by providing encapsulation boundaries and representations for cooperation. New agents may be

introduced into this network, or existing agents can be removed. An agent can be replaced. New connections can be established or existing connections can be removed. The structure of the network is dynamic, and its function is continuous. The internal and external representation issues are clearly separated. Modes and mechanisms of cooperation are discussed. Agents address the difficulties associated with multiple viewpoints and multiple abstraction levels.

Appendix A

This appendix describes the components of the FEM agents that were used to simulate the physics of the casting process including but not exclusive to the Navier-Stokes, Energy and Free Surface agents. In contrast to some of the simpler examples described in Chapter 6, the description of the casting problems and physics involve large number of parameters, and the state spaces of the problems are richer. Nevertheless, continuum mechanics and the partial differential equations provide a basis to define a structured method for describing these complex agents. The mathematical basis of these problems was covered in the thesis. An abstract definition of the agent model was also provided. This section will provide a more concrete example of the implementation of the agents.

A minimum representation that describes these problems must include at least the following.

At the level of individual problems:

- *Problem type* - The underlying equation (Advection Diffusion, etc.) or the physical problem (Fluid Flow, etc.).
- *Geometry* - A discrete representation of the domain and its boundary. Its sub domains and sub boundaries.
- *Materials* - A library of materials describing relevant physical properties, compositions, chemistry, etc.
- *State Variables* - The set of all variables and parameters including the inputs, outputs, constants, parameters, internal variables, etc., required to specify the problem.
  - *Variables*
  - *Initial Values* - The initial values of the state variables
  - *Boundary Conditions* - Boundary conditions for the state variables

At the level of agents:

- *Data Coupling* - Description of the data flow graph among agents.
- *Interaction mode* - Type of problem being solved and the method used such as transient analysis, steady-state analysis, root-finding, constraint solving, etc.

- *Causal Order* - Depending on the type of the problem and the numerical method, a partial or complete ordering of agents that can be required for iterations, time walks, etc.
- *Agent Hierarchy* - The parent - children relationship between agents.
- *External Interface* - A specification of the agents external interface describing its inputs and outputs (requires - provides lists).
  - Inputs
  - Outputs

A grammar was described to define such problems. Based on this grammar, a compiler was constructed using the compiler construction system called Eli [152]. The grammar and the compiler is used to transfer problem information between agents and to archive problems to file systems. In Figure A.1, an overview of this description is provided. A problem contains a series of states (history), each of which corresponds to a state of a problem in an interval.

Figure A.1: (a) The problem description and (b) The agent description

A.1 The grammar

In the rest of this appendix the definition of the agent-problem grammar is presented. The text has been reformatted from the Eli definition of the grammar, see [#198]. However, neither the grammar nor the following description was designed for humans. It is not easily comprehensible and it is not claimed to be complete. However it has been implemented and was found to be sufficient for the problems that were encountered in this thesis.

A.1.1 Coupled Transient Problems

This is a description of the agents involved in a problem and the coupling among them. Data flow is defined by describing the connections of each agent with others. The agents have a name, type, mode and a reference to their parent. The mode of interactions among agents is also defined, i.e., transient problems, steady state problems, iterative problems, constraint problems, etc. The agents inherit the interaction mode from the higher level unless it is explicitly specified.

```
coupledProblem:
    interactionMode '{' agents '}'.

agents:
    'Agents' '{' agent * '}'.

agent:
    'Agent' '{' agentDescription connectivityDescription '}'.

agentDescription:
    agentAttributes * .

agentAttributes:
    'Name' '=' agentName ',' /
    'Type' '=' agentType ',' /
    'Mode' '=' interactionMode ',' /
    'ParentAgent' '=' agentName ',' /
```

```

connectivityDescription:
    'Connections' '{ connection * '}' .

connection:
    '(' dataName 'from:' '(' dataName 'agent:' agentName ')' ')'

interactionMode :
    'Transient' /
    'SteadyState' /
    'Iterative' /
    'ConstraintSolver' /

```

A.1.1.1 Description of the external interface of agents

This is a simple account of all the public data for an agent. The data is classified into two sets: provided (outputs) and required (inputs). These set may overlap. Boundary conditions and geometry may also be defined as public data.

```

contextDescription:
    'Contexts' '{ context * '}' .

context:
    'Context' contextField 'EndContext' .

contextField:
    contextName contextProvides contextRequires .

contextName:
    'Name' '=' name ';' .

contextProvides:
    'Provides' dataName * 'EndProvides' ';' .

contextRequires:
    'Requires' contextEntry * 'EndRequires' ';' .

contextEntry:
    '(' dataName 'bc:' logical ')' .

```

A.1.2 The continuum mechanics problem

This is a detailed description of what is in a continuum mechanics problem. Most continuum mechanics agents internally use this representation. A problem contains materials, geometry and data. Boundary conditions are represented as special data (which is called untyped). All data are strongly typed. They must belong to one of the fields in the problem, i.e., the problem itself (this is the most generic entity), the geometry and its components, such as nodes, elements, surfaces, faces. The problem contains a short report of its contents, i.e., check sum of its geometry, names of the materials, specifications of the data. This allows rapid queries.

```
problemContentsDescription:
    'Problem' problemContent 'EndProblem' .

problemContent:
    fieldDescriptions contextDescription .

specificFieldsDescription:
    specificFieldInformation * .

fieldDescriptions:
    'Fields' '{' field * '}' .

field:
    'Comment' quotedString 'EndComment'           /
    'Geometry' geometryChecksum ';'                 /
    'Materials' materialSpecifications 'EndMaterials' /
    'DataItem' dataItemSpecification dataItemChecksum
```

A.1.2.1 Materials

Materials are provided by a material database. This database has a set of built-in materials. They are referred by their names in the problem. The problem has an ordered list of material names. The following provides a method to define custom material and material properties that are not in this database. Piece wise polynomials are used to define material properties.

```

materialSpecifications:
    compositeMaterial * .

compositeMaterial:
    '(' materialName * ')' .

individualMaterialItem:
    'UserMaterial' materialItemSpecification 'EndMaterial' ';' .

materialItemSpecification:
    '(' materialSpecificationItems ')'.

materialSpecificationItems:
    basicMaterialDescription materialPropertyInfo.

basicMaterialDescription:
    materialName numberOfProperty.

materialName:
    'Name' '=' name ';' .

numberOfProperty:
    'NumberOfProperties' '=' integer ';' .

materialPropertyInfo:
    'MaterialProperties' '(' materialProperty * ')'.

materialProperty:
    'MaterialProperty' '('
        propertyName
        numberOfPolynomials
        'Polynomials' '(' polynomial * ')'
    ')'.

numberOfPolynomials :
    'NumberOfPolynomialRegions' '=' integer ';' .

polynomial :
    'Polynomial' '('
        polyLowerBound
        polyUpperBound
        polyDegree
        polyCoefficients
    ')'.

polyLowerBound:
    'LowerBound' '=' real ';' .

polyUpperBound:
    'UpperBound' '=' real ';' .

polyDegree:
    'Degree' '=' integer ';' .

polyCoefficients:
    '(' real * ')' .

```

A.1.2.2 Data

Data is the generic mechanism to specify the state variables of the problem. It is composed of two parts: a specification and the actual data. Boundary conditions are also represented as data. A special type of data, called the Untyped data is not interpreted by the compiler. It must be processed by a user defined interpreter (dataUntypedSpecifier). The boundary conditions are defined in this manner. The interpreter for boundary conditions is supplied by the system.

```

dataItemChecksum:
    'Checksum-DataItem' '=' integer 'EndChecksum' .

dataInfo:
    individualDataItem * .

individualDataItem:
    'DataItem' dataItemSpecification * actualData 'EndDataItem'.

dataItemSpecification:
    'Specification' 'is' '(' dataSpecificationItems * ')'
    'EndSpecification'.

actualData:
    beginData lineOfData * endData.

lineOfData:
    '(' chunkOfData * ')' /
    '(' '(' integer ')' quotedString ')'.

chunkOfData:
    integer / real / double / character / quotedString / logical.

dataSpecificationItems:
    dataItemName
    dataIndexSpecifier
    dataTypeSpecifier
    dataFormatSpecifier
    dataSizeSpecifier
    dataDimensionSpecifier
    dataPrivate
    dataUntypedSpecifier.

dataItemName:
    'Name' '=' name ';' .

dataIndexSpecifier:
    'Index' '=' indexIdentifier ',' .

```



```

dataTypeSpecifier:
    'Type' '=' typeIdentifier ';' .

dataFormatSpecifier:
    'Format' '=' formatIdentifier ';' .

dataSizeSpecifier:
    'NumberOfEntries' '=' integer ';' .

dataDimensionSpecifier:
    'Dimensions' '=' integer * ',' .

dataPrivate:
    'Private' '=' logical ';' .

dataUntypedSpecifier:
    'UntypedSpecifier' '=' name ';' .

indexIdentifier :
    'per_problem' /
    'per_part' /
    'per_element' /
    'per_node' /
    'per_surface' /
    'per_edge' .

typeIdentifier:
    'REAL' /
    'INTEGER' /
    'DOUBLE' /
    'CHARACTER' /
    'STRING' /
    'UNTYPED' .

dataFormatSpecifier:
    'scalar' /
    'vector' /
    'matrix' .

```

A.1.2.3 Geometry

The geometry is a discrete representation of the physical domain. A part is a domain (region). Sub parts are sub domains. Surfaces are the boundaries. Elements are the primitive components of the parts. Faces are the primitive components of the surfaces. Surfaces are bounded by the edges. The nodes represent coordinates. All entities of the problem, except nodes, are hierarchical, i.e., they may have a parent or children.

```

geometryChecksum:
    'Checksum-Geometry' '=' integer 'EndChecksum' .

geometryInfo:
    partHierarchy .

partHierarchy:
    part *.

part:
    'PartDescription' partDescription 'EndPart' .

partDescription:
    definePartGlobalID geometrySizes individualPartField * .

individualPartField:
    elementInfo      /
    nodeInfo         /
    surfaceInfo      /
    subPartInfo.

definePartGlobalID:
    'GlobalID' '=' partGlobalID ',' .

geometrySizes:
    geometrySize * .

geometrySize:
    elementSizes      /
    nodeSizes         /
    surfaceSizes      /
    subPartSizes.

elementSizes:
    'NumberOfElements' '=' integer ','.

nodeSizes:
    'NumberOfNodes' '=' integer ','.

surfaceSizes:
    'NumberOfSurfaces' '=' integer ','.

subPartSizes:
    'NumberOfSubParts' '=' integer ','.

subPartInfo:
    'SubPart' 'of' partGlobalID ','.

elementInfo:
    'Elements' '{' element * '}' 'EndElements' .

nodeInfo:
    'Nodes' '{' node * '}' 'EndNodes' .

element:
    elementDefinition / elementReference .

elementDefinition:

```

```

    '(' localID type nodeNumber* elementGlobalID')'.

elementReference:
    '(' elementNumber 'ref:' elementNumber ')'.

node:
    nodeDefinition / nodeReference .

nodeDefinition:
    '(' localID coordinate* nodeGlobalIDactiveFlag ')'.

nodeReference:
    '(' nodeNumber 'ref:' nodeNumber ')'.

elementGlobalID:
    partGlobalID
    '(' 'parent:' elementNumber ')' .

nodeGlobalID:
    globalID.

nodeNumber:
    integer .

elementNumber:
    integer .

elementType:
    integer .

coordinate:
    real .

localID:
    integer .

addressSpaceNumber:
    integer .

canonicalFaceNumber:
    integer .

surfaceInfo:
    'Surfaces' '(' surfaceDescriptions * ')' 'EndSurfaces' .

surfaceDescriptions:
    '(' surfaceID surfaceType surfaceContents ')'.

surfaceContents:
    surfaceFaceList .

surfaceContents:
    surfaceFacelist surfaceEdgeList .

surfaceID:
    '$surface' surfaceIDDescriptors *

```

```

surfaceIDDescriptors:
    surfaceParentSurface /
    surfaceAdjacentSurfaces /
    surfaceGlobalID .

surfaceParentSurface:
    'ParentSurface' '(' partGlobalID ':' localNumber ')' ',' .

surfaceAdjacentSurfaces:
    'AdjacentSurfaces' '{' adjacentSurface * '}' ',' .

adjacentSurface:
    '(' partGlobalID ':' localNumber ')' .

surfaceGlobalID:
    'Part' partGlobalID 'Number' localNumber ',' .

partGlobalID:
    '[' addressSpaceNumber localNumber ']' .

surfaceType:
    'SurfaceType' '=' name ',' .

surfaceFaceList:
    'Faces' faceDescription * 'EndFaces' .

surfaceEdgeList:
    'Edges' brepEdgeInfo * 'EndEdges' .

faceDescription:
    '(' 'e:' elementNumber 'f:' canonicalFaceNumber ')' .

brepEdgeInfo:
    brepEdgeNodeDescription .

brepEdgeInfo:
    brepEdgeAdjacentEdges brepEdgeNodeDescription .

brepEdgeInfo:
    brepEdgeCorners brepEdgeAdjacentEdges brepEdgeNodeDescription .

brepEdgeCorners:
    'EdgeVertices' '{' edgeVertices * '}' .

brepEdgeAdjacentEdges:
    'AdjacentEdges' '{' adjacentEdge * '}' .

adjacentEdge:
    '(' 's:' surfaceGlobalID 'edgeNumber:' integer ')' .

brepEdgeNodeDescription:
    '(' nodeIdNumbers ')' .

```

References

1. Finger, S. and J.R. Dixon, *A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Process*. Research in Engineering Design, 1989. 1: p. 51-67.
2. Finger, S. and J.R. Dixon, *A Review of Research in Mechanical Engineering Design. Design. Part II: Representation, Analysis, and Design for the Life Cycle*. Research in Engineering Design, 1989. 1: p. 121-137.
3. Simpson, B.L., *History of the metal casting industry*. 1969, Des Plaines, IL: American Foundrymen's Society.
4. Kuhn, T.S., *Structure of scientific revolutions*. 1970, Chicago: The University of Chicago Press .
5. Fulton, R.E., *A Framework for Innovation*. Computers in Mechanical Engineering, 1987. 5(5): p. 26-40.
6. Voelcker, H.B. and A.A. Requicha, *Geometric modelling of mechanical parts and processes*. IEEE Comput., 1977. 10(12): p. 48-57.
7. Requicha, A.A., *Representations for Rigid Solids: Theory, Methods and Systems*. Computing Surveys, 1980. 12(4): p. 437-463.
8. Beeley, P.R., *Foundry Technology*. 1972, London: Butterworths.
9. Copp, N.H. and A.W. Zanella, *Discovery, Innovation, and Risk*. 1993, Cambridge, Mass.: M.I.T. Press.
10. Brooks, A.N. and T.J.R. Hughes, *Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations*. Computer Methods in Applied Mechanics and Engineering, 1982. 32: p. 199-259.
11. Gresho, P.M., et al., *A Modified Finite Element Method for Solving the Time-Dependent Incompressible Navier-Stokes Equations. Part I: Theory*. International Journal for Numerical Methods in Fluids, 1984. 4: p. 557-598.
12. Jiang, C.B., *A Least-Squares Finite Element Method for Incompressible Navier-Stokes Problems*. International Journal for Numerical Methods in Fluids, 1992. 14: p. 843-859.
13. Beckermann, C. and R. Viskanta, *Natural Convection Solid/Liquid Phase Change in Porous Media*. International Journal of Heat and Mass Transfer, 1988. 31(1): p. 35-46.
14. Stefanescu, D.M., *Critical Review of the Second Generation of Solidification Models for Castings: Macro Transport-Transformation Kinetics Codes, in Modeling of Casting, Welding and Advanced Solidification Processes-VI*, T.S.

- Piwonka, V. Voller, and L. Katgerman, eds. 1993, Warrendale: The Minerals, Metals & Materials Society. p. 3-20.
15. Rappaz, M. and P. Thevoz, *Solute Diffusion Model for Equiaxed Dendritic Growth*. Acta Metallurgica, 1987. 35(7): p. 1487-1497.
  16. Hughes, E., et al., *Experience with Macro-Micro Modelling of Solidification in Castings*, in *Numerical Simulation of Casting Solidification in Automotive Applications*, C. Kim and C.-W. Kim, eds. 1991, Warrendale: The Minerals, Metals & Materials Society. p. 99-120.
  17. Glimm, J., *Nonlinear and Stochastic Phenomena: The Grand Challenge for Partial Differential Equations*. SIAM Review, 1991. 33(4): p. 626-643.
  18. Karsay, S.I., *Ductile Iron III, Gating and Riserling*. 1981, Sorel, Quebec: QIT Fer et Titane Inc.
  19. AFS, *Metalcaster's Reference and Guide*. 1989, Des Plaines, Illinois: American Foundrymen's Society.
  20. Monroe, R.W., *Expandable Pattern Casting*. 1992, Des Plaines, Illinois: American Foundrymen's Society.
  21. AFS, *Patternmaker's Manual*. 1986, Des Plaines, Illinois: American Foundrymen's Society.
  22. Campbell, J., *Castings*. 1991, Jordan Hill, Oxford: Butterworth-Heinemann Ltd.
  23. Pehlke, R.D., *Formation of Porosity during Solidification of Cast Metals*, in *Foundry Processes, Their Chemistry and Physics*, S. Katz and C.F. Landefeld, eds. 1988, New York: Plenum Press. p. 427-445.
  24. Ohnaka, I., *Modelling Fluid Flow and Solidification in Casting*, in *Modeling of Casting, Welding and Advanced Solidification Processes-VI*, T.S. Piwonka, V. Voller, and L. Katgerman, eds. 1993, Warrendale: The Minerals, Metals & Materials Society. p. 337-348.
  25. Dai, N.M., *An Implementation of the Icon Paradigm to Systems Modeling and Analysis*, Master of Applied Science Thesis, Technical University of Nova Scotia, 1989
  26. Tjoa, R.T. and M. New, *Rigging User's Manual*, Technical Report, Dept. Mechanical and Aerospace Engineering, MRCO group, Carleton University, Ottawa, 1993
  27. New, J.M., *Application of the medial axis transform in a design environment for die casting*, Master of Engineering Thesis, Carleton University, 1993
  28. Hill, J.L., J.T. Berry, and S. Guleyupoglu, *Knowledge Based Design of Rigging Systems for Light Alloy Castings*. Trans. American Foundrymen's Society, 1991.
  29. Rich, C. and R.C. Waters, *Programmer's Apprentice*. 1990, New York: ACM Press, Addison-Wesley.

30. Sowa, J.F., *Conceptual Structures*. 1984, Reading, Mass.: Addison-Wesley.
31. New, J.M. and R.T. Tjoa, *Casting User's Manual*, Technical Report, Dept. Mechanical and Aerospace Engineering, MRCO group, Carleton University, Ottawa, 1993
32. Dixon, J.R., *Designing with Features: Building Manufacturing Knowledge into More Intelligent CAD Systems*, in *ASME Manufacturing International-88*, 1988. ASME, Atlanta, Georgia.
33. Dai, N.M. and J.A. Goldak, *An Embedded Constraint Based Programming System and Methodology for Creating Mechanical Design and Assembly Environments: Application to the Manufacturing of Castings*, in *Symposium on Manufacturing Automation Programming Language Environments*, 1993. National Research Council of Canada, Ottawa.
34. Goldberg, A. and D. Robson, *Smalltalk-80 The Language*. 1989, Reading, Mass.: Addison-Wesley.
35. Dai, N.M. and J.A. Goldak, *Design Environments*, in *First Atlantic Congress on Advanced Manufacturing and Engineering*. 1991, Halifax: p. 25-39.
36. Maloney, J., A. Borning, and B. Freeman-Benson, *Constraint Technology for User-Interface Construction in ThingLab-II*, (Also Published in OOPSLA '89) Technical Report 89-05-02, Computer Science Department, University of Washington, 1989
37. Struss, P., *Multiple Representation of Structure and Function*, in *IFIP Working Group 5.2, Expert Systems in Computer-Aided Design*. 1987, Amsterdam: North-Holland.
38. Brooks, F.P.J., *No Silver Bullet - Essence and Accidents of Software Engineering*. IEEE Computer, 1987. (April).
39. ASM, *Metals Handbook, Casting*. 9th ed. Vol. 15. 1988, Metals Park, Ohio: American Society for Metals.
40. ASM, *Metals Handbook, Forging and Casting*. 8th ed. Vol. 5. 1970, Metals Park, Ohio: American Society for Metals.
41. Chung, T.J., *Finite Element Analysis in Fluid Dynamics*. 1978, New York: McGraw-Hill.
42. Majda, A., *Compressible fluid flow and systems of conservation laws in several space variables*. Applied Mathematical Sciences, eds. J.E. Marsden, F. John, and L. Sirovich. 1984, New York: Springer-Verlag.
43. Chorin, A.J. and J.E. Marsden, *A Mathematical Introduction to Fluid Mechanics*. Texts in Applied Mathematics, eds. F. John, et al. 1993, New York: Springer-Verlag.
44. Landau, L.D. and E.M. Lifshitz, *Fluid Mechanics*. 2nd ed. Course of Theoretical Physics, Vol. 6. 1987, Oxford: Pergamon Press.

45. Lai, M.W., D. Rubin, and E. Krempfl, *Introduction to Continuum Mechanics*. 1978, Oxford: Pergamon.
46. White, F.M., *Viscous Fluid Flow*. 1974, New York: McGraw-Hill.
47. Landau, L.D. and E.M. Lifshitz, *Fluid Mechanics*. 1968, Oxford: Pergamon.
48. Chorin, A.J., *A Numerical Method for Solving Incompressible Viscous Flow Problems*. Journal of Computational Physics, 1967. 2(1): p. 12-26.
49. Chorin, A.J., *Numerical Solution of the Navier-Stokes Equations*. Mathematics of Computation, 1968. 22(104): p. 745-762.
50. Chorin, A.J., *Computational Fluid Mechanics, Selected Papers*. 1989, San Diego: Academic Press.
51. *Finite Elements in Fluids*, Gallagher R.H., et al., ed. Vol. 1. 1975, New York: John Wiley.
52. Gresho, P.M., R.L. Lee, and R.L. Sani, *On the time-dependent solution of the incompressible Navier-Stokes equations in two- and three-dimensions*, in *Recent Advances in Numerical Methods in Fluids*. 1980, Swansea: Pineridge Press. p. 27-81.
53. Gresho, P.M., et al., *A Modified Finite Element Method for Solving the Time-Dependent Incompressible Navier-Stokes Equations. Part 2: Applications*. International Journal for Numerical Methods in Fluids, 1984. 4: p. 619-640.
54. Gresho, P.M., *On the Theory of Semi-Implicit Projection Methods for Viscous Incompressible Flow and Its Implementation via a Finite Element Method that also Introduces a Nearly Consistent Mass Matrix. Part 1: Theory*. International Journal for Numerical Methods in Fluids, 1990. 11: p. 587-620.
55. Gresho, P.M. and S.T. Chan, *On the Theory of Semi-Implicit Projection Methods for Viscous Incompressible Flow and Its Implementation via a Finite Element Method that also Introduces a Nearly Consistent Mass Matrix. Part 2: Implementation*. International Journal for Numerical Methods in Fluids, 1990. 11: p. 612-659.
56. Hansbo, P., *The Characteristic Streamline Diffusion Method for Convection-Diffusion Problems*. Computer Methods in Applied Mechanics and Engineering, 1992. 96: p. 239-253.
57. Hansbo, P., *The Characteristic Streamline Diffusion Method for the Time-Dependent Incompressible Navier-Stokes Equations*, Preprint, Technical Report 1991-14/ISSN 0347-2809, University of Göteborg, Department of Mathematics, 1991
58. Hughes, T.J.R., *Finite Element Methods in Fluid Mechanics*, Technical Report, Stanford University, 1989
59. Hughes, E., *Computation of Incompressible Navier-Stokes Flows*, Personal Communication, Carleton University, 1992

60. Shakib, F., *Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations*, Ph.D. Thesis, Stanford University, 1988
61. Pironneau, O. and S. Huberson, *Characteristic-Galerkin and the Particle Method for the Convection-Diffusion Equation and the Navier-Stokes Equations*. Lectures in Applied Mathematics (American Mathematical Society-SIAM), 1991. 28: p. 547-565.
62. Guj, G. and F. Stella, *A Vorticity-Velocity Method for the Numerical Solution of 3D Incompressible Flows*. Journal of Computational Physics, 1993. (106): p. 286-298.
63. Fortin, M., *Old and New Finite Elements for Incompressible Flows*. International Journal for Numerical Studies in Fluids, 1981. 1: p. 347-364.
64. Sani, R.L., et al., *The Cause and Cure(?) of the Spurious Pressures Generated by Certain FEM Solutions of the Incompressible Navier-Stokes Equations: Part 2*. International Journal for Numerical Methods in Fluids, 1981. 1: p. 171-204.
65. Sani, R.L., et al., *The Cause and Cure(?) of the Spurious Pressures Generated by Certain FEM Solutions of the Incompressible Navier-Stokes Equations: Part 1*. International Journal for Numerical Methods in Fluids, 1981. 1: p. 17-43.
66. Phillips, T.N. and G.W. Roberts, *The Treatment of Spurious Pressure Modes in Spectral Incompressible Flow Calculations*. Journal of Computational Physics, 1993. (105): p. 150-164.
67. Ghia, U., K.N. Ghia, and C.T. Shin, *High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multi-Grid Method*. Journal of Computational Physics, 1982. 48: p. 387-411.
68. Batchelor, G.K., *An Introduction to Fluid Dynamics*. 1967, Cambridge: Cambridge University Press.
69. Hyman, J.M., *Numerical Methods for Tracking Interfaces*. Physica, 1984. 12D: p. 396-407.
70. Glimm, J., et al., *Front tracking for hyperbolic systems*. Adv. Appl. Math., 1981. 2: p. 91-119.
71. Glimm, J. and O.A. McBryan, *A Computational Model for Interfaces*, UC-32 DOE/ER/03077-263, U.S. Department of Energy, 1985
72. Glimm, J., D. Marchesin, and O. McBryan, *A Numerical Method for Two Phase Flow with an Unstable Interface*. J. Comp. Physics, 1981. 39: p. 179-200
73. Welch, J.E., et al., *The MAC Method: A Computing Technique for Solving Viscous, Incompressible, Transient Fluid Flow Problems Involving Free Surfaces*, Report LA-3425, Los Alamos Scientific Laboratory, 1966
74. Hirt, C.W. and B.D. Nichols, *Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries*. Journal of Computational Physics, 1981. 39: p. 201-225.

75. Jun, L. and D.B. Spalding, *Numerical Simulation of Flows with Moving Interfaces*. Physico-Chemical Hydrodynamics. 1988. **10**(5/6): p. 625-637.
76. Noh, W.F. and P. Woodwad, *SLIC (Simple Line Interface Calculation)*, in *Lecture Notes in Physics*, vol 59: *Fifth International Conference on Numerical Methods in Fluid Dynamics*, A.I.v.d. Vooren and P.J. Zandbergen, eds. 1976, Berlin: Springer-Verlag.
77. Baines, M.J., A. Maffio, and A.D. Filippo, *Unsteady 1D Flows with Steep Waves in Plant Channels: The Use of Roe's Upwind TVD Difference Scheme*. Advances in Water Resources, 1992. **15**: p. 89-94.
78. Baines, M.J., *On Algorithms for Best L_2 Fits to Continuous Functions with Variable Nodes*, Numerical Analysis Report 1/93, University of Reading, Dept. of Mathematics, 1993
79. Priestley, A., *A Quasi-Conservative Version of the Semi-Lagrangian Advection Scheme*, Numerical Analysis Report 2/92, University of Reading, Dept. of Mathematics, 1992
80. Staniforth, A. and J. Côté, *Semi-Lagrangian Integration Schemes for Atmospheric Models-A Review*. Monthly Weather Review, 1991. **119**: p. 2206-2223.
81. Swaminathan, C.R. and V. Voller, *An 'Enthalpy' Formulation for the Numerical Modelling of Mold Filling*, in *Modeling of Casting, Welding and Advanced Solidification Processes-VI*, T.S. Piwonka, V. Voller, and L. Katgerman, eds. 1993, Warrendale: The Minerals, Metals & Materials Society. p. 365-372.
82. Daly, B.J., *Numerical Study of the Effect of Surface Tension on Interface Instability*. The Physics of Fluids, 1969. **12**(7): p. 1340-1354.
83. Daly, B.J., *Numerical Study of Two Fluid Rayleigh-Taylor Instability*. The Physics of Fluids, 1967. **10**(2): p. 297-307.
84. Bermejo, R. and A. Staniforth, *On the Conversion of Semi-Lagrangian Advection Schemes to Quasi-Monotone Schemes*, Technical Report, Recherche en Prévision Numérique, Service de l'Environnement Atmosphérique, Dorval, Quebec, Canada, 1991
85. Hughes, E., *Free-Boundary Methods for Fluids*, Personal Communication , Carleton University, 1991
86. Brackbill, J.U., D.B. Kothe, and C. Zemach, *A Continuum Method for Modeling Surface Tension*. Journal of Computational Physics, 1992. (100): p. 335-354.
87. Van Leer, B., *Toward the Ultimate Conservative Scheme IV: A New Approach to Convection*. J. Comp. Physics, 1977. **23**: p. 276-299.
88. Dantzig, G.B., *Linear Programming and Extensions*. 1963, Princeton: Princeton University Press.

89. Press, W.H., *et al.*, *Numerical Recipes in C, The Art of Scientific Computing*. 1988, Cambridge: Cambridge University Press.
90. Prigogine, I. and I. Stengers, *Order out of Chaos*. 1984, Toronto: Bantam Books.
91. Kurz, W. and D.J. Fisher, *Fundamentals of Solidification*. 1989, Aedermannsdorf, Switzerland: Trans Tech Publications Ltd.
92. Flemings, M.C., *Solidification Processing*. 1974, New York: McGraw-Hill.
93. Tjoa, R.T.H., *Assessment of the Accuracy of a Computational Casting Model*, Master of Engineering Thesis, Carleton University, 1992
94. Bennon, W.D. and F.P. Incropera, *A Continuum Model for Momentum, Heat and Species Transport in Binary Solid-Liquid Phase Change Systems- I. Model Formulation*. Journal of Heat Mass Transfer, 1987. **30**(10): p. 2161-2170.
95. Bennon, W.D. and F.P. Incropera, *A Continuum Model for Momentum, Heat and Species Transport in Binary Solid-Liquid Phase Change Systems- II. Application to solidification in rectangular cavity*. Journal of Heat Mass Transfer, 1987. **30**(10): p. 2171-2187.
96. Buyevich, Y.A., *Heat and Mass Transfer in Disperse Media- I. Averaged Field Equations*. International Journal of Heat Mass Transfer, 1992. **35**(10): p. 2445-2452.
97. Buyevich, Y.A., *Heat and Mass Transfer in Disperse Media- II. Constitutive Equations*. International Journal of Heat Mass Transfer, 1992. **35**(10): p. 2453-2463.
98. Martin, J.C. and W.J. Moyce, *An Experimental Study of the Collapse of Liquid Columns on a Rigid Horizontal Plane*. Philosophical Transactions of the Royal Society of London, Series A, 1952. **244**: p. 312.
99. Simon, H., *The Sciences of the Artificial*. 1969, Cambridge, Mass.: M.I.T Press.
100. Abelson, H., G.J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*. 1985, Cambridge, Mass.: The MIT Press.
101. Hewitt, C., *Viewing control structures as patterns of passing messages*. Journal of Artificial Intelligence, 1977. **8**(3): p. 323-364.
102. Taubner, D., *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*. Lecture Notes in Computer Science, eds. G. Goos and J. Hartmanis. 1989, Berlin: Springer-Verlag.
103. Hoare, C.A.R., *Communicating Sequential Processes*. Communications of the ACM, 1978. **21**(8): p. 666-777.
104. Bunge, M., *Causality and Modern Science*. 1979, New York: Dover Publications Inc.
105. Raynal, M., *Distributed Algorithms and Protocols*. 1988, New York: John Wiley & Sons.

106. Borning, A., *ThingLab: A Constraint-Oriented Simulation Laboratory*, Ph.D. Thesis, Stanford University, 1979
107. Friedman, G.J. and C.T. Leondes, *Constraint Theory, Part III: Inequality and Discrete Relations*. IEEE Trans. Systems Science and Cybernetics, 1969. SSC-5(3): p. 191-199.
108. Friedman, G.J. and C.T. Leondes, *Constraint Theory, Part I: Fundamentals*. IEEE Trans. Systems Science and Cybernetics, 1969. SSC-5(1): p. 48-56.
109. Friedman, G.J. and C.T. Leondes, *Constraint Theory, Part II: Model Graphs and Regular Relations*. IEEE Trans. Systems Science and Cybernetics, 1969. SSC-5(2): p. 132-140.
110. Borning, A., B. Freeman-Benson, and J. Maloney, *An Incremental Constraint Solver*. Communications of the ACM, 1990. 33(1): p. 54-63.
111. Smith, R.G., *A Framework for Distributed Problem Solving*. 1981, Ann Arbor, Michigan: UMI Research Press.
112. Cormen, T.H., C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. 1990, Cambridge, Mass.: The MIT Press.
113. Alefeld, G., A. Gienger, and F. Potra, *Efficient Numerical Validation of Solutions of Nonlinear Systems*. SIAM Journal on Numerical Analysis, 1994. 31(1): p. 252-260.
114. Alur, R. and T.A. Henzinger, *A Really Temporal Logic*. Journal of the ACM, 1994. 41(1): p. 181-204.
115. Silberschatz, A., *Communication and Synchronization in Distributed Systems*. IEEE Transactions on Software Engineering, 1979. 5(6): p. 542-546.
116. Bernstein, A.J., *Outputs Guards and Non-Determinism in CSP*. ACM Toplas, 1980. 2(2): p. 234-238.
117. Coffman, E.G.J. and M.R. Garey, *Proof of the 4/3 Conjecture for Preemptive vs. Nonpreemptive Two-Processor Scheduling*. Journal of the ACM, 1993. 40(5): p. 991-1018.
118. *Domain Decomposition Methods*, Chan, T.F., et al., ed. 1989, Philadelphia, Penn.: SIAM.
119. *Domain Decomposition Methods for Partial Differential Equations*, Glowinski, R., et al., ed. 1991, Philadelphia, Penn.: SIAM.
120. Beguelin, A., et al., *Visualization and Debugging in a Heterogeneous Environment*. IEEE Computer, 1993. 26(6): p. 88-95.
121. Wegner, P., *Dimensions of Object-Oriented Modeling*. IEEE Computer, 1992. 25(10): p. 12-20.

122. Agha, G., *Actors, A Model of Concurrent Computation in Distributed Systems*. 1986, Cambridge, Mass.: The MIT Press.
123. Wegner, P., *Dimensions of Object-Based language Design*, in *OOPSLA '87*, 1987. ACM/SIGPLAN, Orlando, Florida.
124. Rosen, J.P., *What Orientation Should Ada Objects Take?* Communications of the ACM, 1992. **35**(11): p. 71-76.
125. Rettig, M., G. Simmons, and J. Thomson, *Extended Objects*. Communications of the ACM, 1993. **36**(8): p. 19-24.
126. de Bakker, J.W., W.-P. de Roever, and G. Rozenberg, *Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctness. REX Workshop, Mook, The Netherlands, May/June 1989 Proceedings*. Lecture Notes in Computer Science, 1990, Berlin: Springer-Verlag.
127. Bal, H., J. Steiner, and A. Tanenbaum, *Programming Languages for Distributed Computing Systems*. ACM Computing Surveys, 1989. **21**(3): p. 261-322.
128. Moss, J.E., *Nested Transactions: An Approach to Reliable Computing*. 1985, Cambridge, Mass.: The MIT Press.
129. Hillis, W.D., *The Connection Machine*. ACM Distinguished Dissertations, 1985, Cambridge, Massachusetts: The MIT Press.
130. Khokhar, A.A., et al., *Heterogeneous Computing: Challenges and Opportunities*. IEEE Computer, 1993. **26**(6): p. 18-27.
131. Freund, R.F. and H.J. Siegel, *Heterogeneous Processing*. IEEE Computer, 1993. **26**(6): p. 13-17.
132. Carriero, N. and D. Gelernter, *Linda in Context*. Communications of the ACM, 1989. (April 1989): p. 444-458.
133. Meyer, B., *Systematic Concurrent Object-Oriented Programming*. Communications of the ACM, 1993. **36**(9): p. 56-80.
134. Lea, R., C. Jacquemot, and E. Pillevesse, *COOL: System Support for Distributed Programming*. Communications of the ACM, 1993. **36**(9): p. 37-45.
135. Agha, G., *Concurrent Object-Oriented Programming*. Communications of the ACM, 1990. **33**(9): p. 125-141.
136. Grimshaw, A.S., *Easy-to-Use Object-Oriented Parallel Processing with Mentat*. IEEE Computer, 1993. **25**(5): p. 39-51.
137. Moss, E. and W. Kohler, *Concurrency Features for the Trellis/Owl Language*, in *ECOOP '87*, 1987. , Paris, France.
138. Bennet, J., *The Design and Implementation of Distributed Smalltalk*, in *OOPSLA '87*, 1987. ACM/SIGPLAN, Orlando, Florida.

139. Ribbens, C.J., T.W. Layne, and C. Desa, *Toward Parallel Mathematical Software for Elliptic Partial Differential Equations*. ACM Transactions on Mathematical Software, 1993. 19(4): p. 457-473.
140. Bond, A.H. and L. Gasser, *Readings in Distributed Artificial Intelligence*. 1988, San Mateo, Calif.: Morgan Kaufmann.
141. *Decentralized A.I. 2*, Demazeau, Y. and J.-P. Müller, ed. 1991, Amsterdam: North-Holland.
142. *Decentralized A.I.*, Demazeau, Y. and J.-P. Müller, ed. 1990, Amsterdam: North-Holland.
143. *Decentralized A.I. 3*, Werner, E. and Y. Demazeau, ed. 1992, Amsterdam: North-Holland.
144. *Archon: An Architecture for multi-agent systems*, Wittig, T., ed. 1992, New York: Ellis Horwood.
145. Werner, E., *Distributed Cooperation Algorithms*, in *Decentralized A.I.*, Y. Demazeau and J.-P. Müller, eds. 1990, Amsterdam: North-Holland.
146. Duffie, N.A. and R.S. Piper, *Non-Hierarchical Control of a Flexible Manufacturing Cell*. Robotics & Computer Integrated Manufacturing, 1987. 2(2): p. 175-179.
147. Hatvany, J., *Intelligence and Cooperation in Heterarchic Manufacturing Systems*. Robotics & Computer Integrated Manufacturing, 1987. 2(2): p. 175-179.
148. Graefe, U., *An Overview of MAPLE*, in *Symposium on Manufacturing Automation Programming Language Environments*, 1993. National Research Council of Canada, Ottawa.
149. Heap, N., *An Introduction to OSI*. 1993, Oxford: Blackwell Scientific Publications.
150. Parker, T.S. and L.O. Chua, *Practical Numerical Algorithms for Chaotic Systems*. 1989, New York: Springer-Verlag.
151. Strang, G., *Introduction to Applied Mathematics*. 1986, Wellesley, Mass.: Wellesley-Cambridge Press.
152. Gray, R.W., et al., *Eli: A Complete, Flexible Compiler Construction System*. Communications of the ACM, 1992. 35(2): p. 121-131.

END

2 5-0 5-9 5

FIN