1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

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

Votre file - Votre référence

Notre file - Notre référence

NOTICE

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.

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

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.

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.

AVIS

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.

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

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.

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

Visual Descriptions, Formalisms and the Design Process

By

Francis Bordeleau

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

Master of Computer Science

**Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
July 16, 1993**

© copyright
1993 Francis Bordeleau

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

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 accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

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 propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89812-7

Canada

Name Francis Borden

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.

Computer Science

SUBJECT TERM

0984

U-M-I

SUBJECT CODE

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 0579
Slavic and East European 0575

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

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 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology 0626
General 0627
Criminology and Penology 0938
Demography 0631
Ethnic and Racial Studies 0628
Individual and Family Studies 0629
Industrial and Labor Relations 0630
Public and Social Welfare 0700
Social Structure and Development 0344
Theory and Methods 0709
Transportation 0999
Urban and Regional Planning 0453
Women's Studies 0453

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 0306
Biology 0287
General 0308
Anatomy 0309
Biostatistics 0379
Botany 0329
Cell 0353
Ecology 0369
Entomology 0793
Genetics 0410
Limnology 0307
Microbiology 0317
Molecular 0416
Neuroscience 0433
Oceanography 0821
Physiology 0778
Radiation 0472
Veterinary Science 0786
Zoology 0760
Biophysics 0786
General 0760
Medical 0425
Earth Sciences 0996
Biogeochemistry 0996
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Polynology 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 0381
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation 0605
Mathematics 0986
Physics 0606
General 0608
Acoustics 0748
Astronomy and Astrophysics 0607
Atmospheric Science 0798
Atomic 0759
Electronics and Electricity 0609
Elementary Particles and High Energy 0610
Fluid and Plasma 0752
Molecular 0756
Nuclear 0611
Optics 0463
Radiation 0346
Solid State 0984
Statistics 0984
Applied Sciences 0984
Applied Mechanics 0984
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 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

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

Visual Descriptions, Formalisms and the Design Process

**submitted by Francis Bordeleau, B.A.Sc, B.Sc
in partial fulfillment of the requirements for
the degree of Master of Computer Science**

Two handwritten signatures in black ink, one on the left and one on the right, both written over a horizontal line.

Thesis Supervisors

A handwritten signature in black ink, appearing to read 'T. R. P.', written over a horizontal line.

Director, School of Computer Science

**Carleton University
November 16, 1993**

Abstract

The use of visual notations is constantly increasing in distributed system design. They offer more flexibility to designers and give more expressiveness to resulting designs. On the other hand, Formal Description Techniques (FDTs) produce unambiguous specifications on which formal verification can be conducted.

In this thesis, we take steps towards the definition of a general framework for the integration of formal methods in the design process of distributed systems by defining formal interpretation methods for design. Also, the definition of a LOTOS interpretation method is discussed.

In order to define a LOTOS interpretation method, a LOTOS architectural interpretation method is defined. The method is based on a particular type of architectural representations called LARG (LOTOS Architectural Representation Graph). Both Grouping and UnGrouping algorithms are defined on LARGs. Then, for the generation of LOTOS architectural expressions from LARGs, the LAEG (LOTOS Architectural Expression Generation) method is defined.

Acknowledgements

I would like to express my grateful thanks to Dr. R.J.A. Buhr (Department of Systems and Computer engineering at Carleton University) and Dr. A. Obaid (Département d'informatique de l'Université du Québec à Hull) for the co-supervision of this thesis. They both provided me with excellent guidance in their respective field of specialisation (respectively distributed system design and formal methods). I especially want to thank them for the freedom they gave me in conducting this research project.

Many thanks to Daniel Amyot for reading the draft version of this thesis and for making judicious comments.

J'aimerais aussi remercier mes parents pour l'appui moral et financier qu'ils m'ont apporté au cours de toutes ces années.

J'aimerais surtout remercier Louise-Marie, Xavier et Raphaëlle pour leur amour, leur patience et leurs encouragements.

Finally, I would like to thank with gratitude TRIO (Telecommunication Research Institute of Ontario) for its financial support.

Table of Contents

Acceptance Sheet.....	ii
Abstract	iii
Acknowledgements.....	iv
Table of Contents	v
List of tables and specifications	ix
List of Figures	x
List of Appendices.....	xiv
Chapter 1 Introduction.....	1
1.1 Overview.....	1
1.2 Motivations.....	2
1.3 Objectives	3
1.4 Organization.....	5
Case Study.....	6
Chapter 2 The Distributed System Design Process	9
2.1 Design Process	9
2.2 Design Method.....	11
2.3 Design Model	12

Table of Contents

2.3.1	Description Model	13
2.3.1.1	Architectural Model	15
2.3.1.2	Behavioral Model	24
2.3.1.3	Design Tree Representation.....	25
2.3.2	Interpretation Method	28
2.3.2.1	Architectural Expressions	29
2.3.2.2	Architectural Interpretation Method	29
2.3.2.3	Behavioral Expressions	30
2.3.2.4	Behavioral Interpretation Method	30
2.3.2.5	Composition of Total Behavior.....	32
2.3.2.6	Definition Steps	32
2.3.3	Semantic model	33
2.4	Analysis Model	33
Chapter 3	A Framework for Formal Method Integration	34
3.1	Formal Method Integration.....	34
3.2	LOTOS Integration.....	36
Chapter 4	Definition of Generic Formal Interpretation Methods	39
4.1	Generic Formal Interpretation Methods.....	39
4.2	A Generic LOTOS Interpretation Method	42
4.2.1	Definition of the Generic LOTOS Interpretation Method	43
Chapter 5	LOTOS Architectural Interpretation Methods.....	48
5.1	Solution Overview	48
5.1.1	The AM Method	49
5.1.2	Generic LOTOS architectural interpretation method.....	50
5.1.3	Application of the Generic LOTOS architectural interpretation method to EGL, MachineCharts and ObjecTime	52
5.2	Nature of the LARG Model	53
5.2.1	AM Requirements	54
5.2.2	LAEG Requirements	54
5.2.2.1	Architecture Analysis	55
5.2.2.2	Graph Linearization	55

5.2.3	Summary	56
Chapter 6	The LARG Model.....	58
6.1	LARG Visual Notations	58
6.1.1	LARG Components	59
6.1.2	LARG Semantic Conventions.....	62
6.2	Formal Description of LARG	73
6.2.1	Functions on LARGs	75
6.2.2	Example of LARG Formal Description	77
6.2.3	Partition.....	80
6.2.4	Grouping	80
6.2.4.1	The Grouping Algorithm.....	81
6.2.4.2	Definition of generalized intersection operator « :.....	83
6.2.4.3	Example of Application of the Grouping Algorithm	83
6.2.5	UnGrouping.....	86
6.2.5.1	The UnGrouping Algorithm	87
6.2.5.2	Example of Application of the UnGrouping Algorithm	88
6.3	Discussion.....	91
Chapter 7	Issues in Formal Interpretation of LARGs	94
7.1	Design Errors	94
7.1.1	Partially Defined Gates.....	95
7.1.2	Undefined Gates.....	95
7.1.3	Ambiguously Defined Gates	96
7.2	Determinism vs. Non-Determinism	98
7.2.1	Deterministic LARGs	99
7.2.2	Non-Deterministic LARGs	102
7.2.2.1	Non-deterministic parallel-interactions	105
7.2.2.2	Non-deterministic interaction-choice.....	111
7.2.3	Discussion (Possible Solutions to LAEG problem)	116
7.4	Relation Between these Issues and Description Models	119
7.4.1	Relation with EGL.....	119
7.4.2	Relation with ObjecTime.....	119

Chapter 8	The LAEG Method.....	120
8.1	Description of the LAEG method	120
8.1.1	LAEG Algorithm	121
8.1.2	LARG Analysis Method.....	122
8.1.3	Binary Grouping Method.....	124
8.2	Possible Applications of the LAEG Method	128
8.2.1	LOTOS Architectural Interpretation	128
8.2.2	Architecture Refactoring.....	131
8.2.3	LOTOS Correctness Preserving Transformation	133
Chapter 9	Conclusions and Future Work.....	137
9.1	Thesis Summary.....	137
9.2	Contributions	138
9.3	Future Research	140
List of	References.....	141

List of tables and specifications

Table 2.1	Projection of different architectural models on the different planes.....	23
Specification 4.1	Derivation of the skeleton of a LOTOS specification from the design tree representation	45
Specification CS6.1	The skeleton of the elevator system LOTOS specification	72
Table CS6.1	External effects of architectural artifact implied behavior on LOTOS specifications (explanation in the external effect section in page 67-68).....	73

List of Figures

Figure CS1.1	Factoring of the Elevator system problem.....	8
Figure 2.1	Component-centered design process.....	10
Figure 2.2	Design steps in component-centered design process.....	11
Figure 2.3	The Design method	12
Figure 2.4	The Design model.....	13
Figure CS2.1	MachineCharts notations used in this thesis	16
Figure CS2.2	A MachineCharts architecture of the elevator system	18
Figure 2.5	Projection of architectural design on structural design.....	22
Figure 2.6	Abstract example of architectural design	26
Figure 2.7	Abstract example of design tree representation	27
Figure CS2.3	Design tree representation of the elevator system MachineCharts design	28
Figure 2.8	Example of architectural effect of behavior interpretation	31
Figure 3.1	Integration of formal method Y in design method X.....	35
Figure 3.2	Integration of LOTOS in design method X	37
Figure 4.1	Design models.....	40
Figure 4.2	A generic interpretation method in relation with design models.....	41
Figure 4.3	A generic LOTOS interpretation method	43
Figure CS4.1	A LOTOS interpretation method for MachineCharts designs	47

List of Figures

Figure 5.1	Solution for LOTOS architectural interpretation method	49
Figure 5.2	LARG intermediate model in relation with generic architectural models.....	51
Figure 5.3	The LARG model in relation with EGL, MachineCharts and ObjecTime.....	52
Figure 5.4	Linearization by grouping.....	56
Figure 5.5	Tree representation and LOTOS architectural expression of the linearized LARG	56
Figure 6.1	Example of a simple LARG.....	59
Figure 6.2	Interaction notations	59
Figure 6.3	Illustration of 2-way and 3-way interactions in LARG.....	60
Figure 6.4	Example of grouping in LARG.....	61
Figure 6.5	Process architectural description box	62
Figure 6.6	Interleaved processes	63
Figure 6.7	2-way non-deterministic interaction-choice vs. 3-way interaction	64
Figure CS6.1	The LARG representation of the elevator system MachineCharts design	65
Figure CS6.2	The Car_SC subsystem LARG representation	66
Figure CS6.3	The Coordination subsystem LARG representation	66
Figure CS6.4	Tree representation of the elevator system LARG.....	70
Figure 6.8	LARG example.....	77
Figure 6.9	Application of the grouping algorithm.....	84
Figure CS6.6	Application of grouping to the elevator system LARG	86
Figure 6.10	Application of the ungrouping algorithm.....	89
Figure CS6.7	The ungrouped version of the elevator system LARG.....	91

List of Figures

Figure 6.11	General 3-component and 4-component LARGs.....	92
Figure 7.1	Partially defined gate error	95
Figure 7.2	Undefined gate error.....	96
Figure 7.3	Correction of ambiguously defined gate error	97
Figure 7.4	Example of deterministic AR	99
Figure 7.5	Linear vs. non linear LARGs.....	101
Figure 7.6	General 3-component deterministic LARG	101
Figure 7.7	Three different groupings.....	102
Figure 7.8	Example of non-deterministic LARG.....	103
Figure 7.9	No LAEG solution LARG	104
Figure 7.10	Interacting process grouping.....	107
Figure 7.11	Addition of Extended gate identifier in LARGs	108
Figure 7.12	Process subset decomposition grouping	109
Figure 7.13	LARG without "pure" grouping solution	110
Figure 7.14	Hybrid grouping solution	111
Figure 7.15	2-way non-deterministic parallel-interactions choice grouping.....	114
Figure 7.16	4-way non-deterministic parallel-interactions choice grouping.....	115
Figure 8.1	Different groupings in a 4-process LARG	127
Figure 8.2	Illogic LAEG grouping in LARG	129
Figure 8.3	Using the LAEG method for architecture refactoring.....	132
Figure CS8.1	Refactoring of the elevator system	133
Figure 8.4	Application of the CPT method.....	136
Figure CS1.1	Factoring of the Elevator system problem.....	148

List of Figures

Figure CS1.2	Elevator Problem.....	149
---------------------	------------------------------	------------

List of Appendices

Appendix A	Description of the elevator system problem.....	147
-------------------	--	------------

Chapter 1 Introduction

1.1 Overview

This thesis is part of an ongoing TRIO project which aims at integrating formal description techniques (FDTs) into the design of concurrent, distributed systems. The first part of this TRIO project led to M. Vigder's Ph.D thesis ([Vig92]), and two technical reports by S. Lamouret ([Lam91], [Lam92]). While [Lam91] defines a method for state machines to LOTOS translations, the other two reports, ([Vig92]) and [Lam92], aim at integrating FDTs (LOTOS) in the design process of distributed systems by means of "timethreads" (see [Buh92b]) ("slices" in [Vig92] are now called "timethreads").

In this project, we take a more general approach; the project aims at defining a general framework for the integration of FDTs in the design process of distributed systems. The framework is based on the use of formal interpretation methods for designs. The FDT LOTOS ([ISO89]) is used the main formalism. For people who are interested in LOTOS, we suggest two tutorials: [Bol90] and [Log92].

In this thesis, we take steps towards the definition of such a general framework for FDT integration in the design process of distributed systems by analysing the issues in defining generic formal interpretation methods. The focus of this thesis is on the conventional component-centered design process (see definition of component-centered design process in §2.1). For this type of design process, we define a generic LOTOS interpretation method which enables the formal interpretation of different types of architectural models in LOTOS terms. The method is based on a particular type of architectural representation model called the LARG (LOTOS Architectural Representation Graph) model. Also, we define a method, called LAEG (LOTOS Architectural Expression Generation) method, for the generation of LOTOS architectural expressions from LARGs. Finally, we draw up a list of topics for future research.

1.2 Motivations

The importance of visual notations in design is continually increasing in the distributed system design community. Many CAD tools based on visual notations have been developed. TimeBench [Buh92a], EGL [Oba90] and ObjecTime [Sel92] are such examples. The use of visual notations in system design offers the advantages of being more expressive and easier to conceptualize than textual descriptions; visual notations are particularly expressive when used for the description of system architecture in which only system components and interactions among them are described. A picture is sometimes worth a thousand words. It is natural for humans to first sketch a diagram when searching the solution of a particular problem. Also, visual description methods are concerned with less syntactic details than textual description. Therefore, visual description techniques give the designer more freedom to express system component interactions. As discussed in [Buh91], the use of visual description also allows the designer to "play" the design, i.e. to exercise temporal behavior in design terms before the implementation.

On the other hand, FDTs offer the advantage of producing unambiguous specifications. They possess a sound theoretical basis which allows the use of formal validation techniques. Also, FDTs allow the formal verification of system's properties such as liveness, safeness and races, which constitute important features in the design of concurrent systems. However, most FDTs have the drawback to be difficult to learn and to be only known by few specialists.

LOTOS, which is the FDT used in this project, is based on process algebra. It is compilable and executable. Different proof methods have been developed for LOTOS ([Boi89], [Fan90], [Gal91]). Many tools for LOTOS have also been developed for interpretation ([Log90]), compilation ([Man89], [Nom90]), syntax and semantic analysis ([Log90], [Boc89]), execution ([Gui88]), simulation ([Log90], [Ohm90]) and code generation ([Eij90]). Also, design methods, for distributed system design, based on the use of LOTOS have recently been defined ([Sch92]). LOTOS has been chosen as the FDT because it possesses high level constructs such as parallelism and interesting features such as non-determinism which is needed for the design of distributed systems, and because it is supported by a fair amount of tools (the number of LOTOS-tools is constantly increasing).

1.3 Objectives

The main objective of this thesis is to take steps towards the definition of a framework for the integration of formal methods into the design process of distributed systems. In order to integrate formal methods into the pictorial, or visual, design, we, in this thesis, propose the introduction of formal interpretation methods for design.

We want to emphasize that the goal of this project on formal method integration in the distributed system design process, in the context of which this thesis has been developed, is not to change existing informal design methodologies for a formal one, but rather to add formalism to existing informal methodologies. In this thesis, we propose to integrate formal methods to the design process by means of a formal interpretation method of designs. Thus, the formal specification comes as the result of the interpretation of the design description. Therefore, the designer does not have to change his or her working methodology in order to benefit from the power of formalism. Also, the industry is usually reluctant to integrate formalism into their design process, because most of the proposed methods until now (e.g. [Sch92]), for such an integration, involve deep changes of their design methodology, and therefore big time and money investment. This is not what we propose here. We aim at developing a method which will lead to a soft-integration of FDTs into the design process.

Also, the communication aspect constitutes a crucial one in distributed system design. Therefore, since LOTOS has proven to be a proper FDT to model the communication aspect in distributed system design, we, in this thesis, define an interpretation method which allows the generation of LOTOS specifications as the result of the formal interpretation of distributed system designs. Such a method offers the advantage of combining both the strength of visual description techniques and the strength of FDTs. The FDT LOTOS is used, in this project, to model the communication aspect of systems.

The LOTOS interpretation method presented in this thesis is a generic method for the generation of LOTOS specifications, in that it aims at generating LOTOS specifications from any type of description models. Also, since some methods already exist for the LOTOS interpretation of behavioral description ([Lam91],[Oba90]), we oriented our research towards the LOTOS interpretation of architectural descriptions, which correspond to the architectural part of design description models (see §2.3.1). The method is based on the use of a particular type of architectural representation called LARG (LOTOS

Architectural Representation Graph) (§6) which serve as an intermediate model between architectural descriptions and LOTOS specifications. For the LARG model, we define both visual notations and formal descriptions. The visual notations used in the LARG model to represent processes and gate sets are similar to the ones used in GLOTOS ([ISO89b]). However, the two models are conceptually very different. For example, the LARG model allow the direct representation of both arbitrary topologies of interacting components and N-way interactions which is not the case for GLOTOS.

In this thesis, we discuss the application of our interpretation method in relation with three description models: EGL, MachineCharts (description model of TimeBench) and ObjecTime. For the generation of LOTOS architectural expressions from LARGs, we define a method called the LAEG (LOTOS Architectural Expression Generation) method. In this thesis, we distinguish between two different types of LOTOS expressions: architectural expressions, which are expressed using only the parallel, the enable and the disable operators, and behavioral expressions, which are expressed using only action prefix and choice operators. However, in standard LOTOS there is no such distinction.

The LAEG method presented in this thesis may be used to formally support architectural transformations within the design. In LOTOS, system architectures are described by means of architectural expressions, i.e. a LOTOS expression in which only parallel composition operators are involved. The LAEG method allows, for a given LOTOS architectural expression, the generation of the set of all equivalent LOTOS architectural expressions with respect to the former one. Therefore, having an architecture A, we may use the LAEG method to formally support the refactoring, or transformation, of the architecture into an equivalent architecture A'. However, the LAEG method supports architecture refactoring in the LARG world, not in the designer's conceptual world. In order to use the LAEG method to support architecture refactoring in designer's world some issues still have to be resolved. This will be discussed in §8.

The LAEG method may also be used as a LOTOS CPT (correctness preserving transformation) for parallel operators. In this way, our method is related to the study of architectural design transformation rules discussed in [Bol90] and [Vis89]. In [Vis89], four general architectural transformation laws are given. These are proven to be weak bisimulation equivalent. In [Bol90], an algorithm for the generation of LOTOS architectural expressions from PGNs (Process Gate Nets) is defined. Bolognesi's algorithm can also be used to generate equivalent LOTOS architectural expressions for a given one, since the

algorithm allows the generation of all equivalent LOTOS architectural expressions. However, both papers only consider totally deterministic architectures (totally deterministic is defined in §7.2), i.e. architectures in which the processes participating in a given interaction can be determined simply by the knowledge of the interaction gate name. Our method is concerned with both deterministic and non-deterministic architectures (deterministic and non-deterministic are defined in §7.2). However, the generation of LOTOS architectural expressions from non-deterministic LARGs is not always possible (see discussion in §7.2.3). Also, our method addresses other important aspects of distributed system design such as validation and analysis of architectural designs. In this thesis, we also discuss the possible integration of the LAEG method in a global distributed system design process.

The objectives of this thesis are:

- 1^o) to analyse the design process of distributed system,
- 2^o) to take steps towards the definition of a framework for the integration of formal methods in the design process of distributed systems, and to analyse the issues for such an integration,
- 3^o) to present an intermediate model solution for the formal interpretation of architectural descriptions, which includes:
 - description of the LARG intermediate model,
 - analysis of the issues in LARG formal interpretation using LOTOS, and
 - presentation of the LAEG method.
- 4^o) to illustrate the different definitions and methods by examples.

1.4 Organization

This thesis is organized as follows. In chapter 2, we analyse the design process and define the terminology. In chapter 3, we take steps towards the definition of a framework for the integration of formal methods into the design process of distributed systems. The framework is based on the use of formal interpretation methods for designs. In chapter 4, we analyse the issues in defining formal interpretation methods for designs. In this chapter, we also describe the basis of a LOTOS interpretation method. In chapter 5, we propose an intermediate method solution for the definition of a LOTOS architectural interpretation method. First, we give an overview of the solution and discuss the purposes of introducing

an intermediate model in the LOTOS architectural interpretation method, and then we discuss the nature of such intermediate model. In chapter 6, we present the LARG model used as the intermediate model in the LOTOS architectural interpretation method. In chapter 7, we discuss two important issues in relation with the LOTOS interpretation of architectural designs: design errors, and deterministic and non-deterministic architectures. In chapter 8, we describe the LAEG method and discuss some of its possible applications. Finally, in chapter 9, a conclusion is provided. We summarize the work done in the thesis, enumerate the different contributions, and draw up a list of topics for future research.

Case Study

To relate the material presented in this thesis to distributed system design, a distributed system case study is provided. The system chosen is the elevator system (see description of the elevator problem below). This system is used through the thesis to illustrate the different definitions and methods. The elevator problem has been chosen mainly for the two following reasons:

- 1^o) it constitutes an example of distributed system to which everyone can relate to, and
- 2^o) it is complex enough to show that our method can be applied to practical distributed system design.

The second reason can be justified as follows:

- the elevator system is complete enough to illustrate real design problems,
- it is complex enough to require system decomposition (and therefore refactoring can be needed at some points in the design process),
- the solution domain requires the use of both hardware and software components, and
- the solution domain requires the use of different types of software components (processes, subsystems, objects, etc.).

To make a clear distinction between the theoretical content of the thesis and the relation to the elevator system, we use a framing to enclose everything related to the elevator system. Thus, boxes labelled "Case Study: The Elevator System" have been inserted all along the thesis to illustrate how the material developed here relate to distributed system design. Also, case study figures are differentiated from other ones using the special label CS at the beginning of the figure identifier (e.g. figure CS1.1 means that the figure is the first case

study figure of chapter 1). The case study also serves to identify research area for future works.

The description of the elevator system and the resulting design have taken from [Buh93a]. The design is given using the MachineCharts description model [Buh90].

Case Study: The Elevator System

Description of the Problem

A computer-controlled elevator system with N cars and M floors is to be installed in a building. The problem is to design the architecture of the strategic control software, allowing for the possibility that the hardware architecture may be either a single computer, or a computer for each car plus a central computer. The regime of operation of concern to this problem begins with all cars idle on some arbitrary initial floor; during this regime, cars are either in motion, visiting a floor, or idle at the last floor visited.

The elevator system outside the computer consists of cars, elevator shafts, hoist cables and winches, motors to drive the winches, doors on both the shafts at each floor and the cars, electromechanical controls, and sensors to give information on elevator position and movement, and buttons to request service. Each car has M buttons, 1 for each floor, to request a car to visit that floor. Each floor has 2 buttons (except bottom and top, which only have one each), one to request a car going up and one to request a car going down.

The elevator system is illustrated in figure CS1.1. We observe, in this figure, that the system is decomposed in two distinct parts: tactical control and strategic control. In this case study, the problem consists in designing the strategic control of the system.

Chapter 2 The Distributed System Design Process

In this chapter, we analyse and give a general description of the design process and its different components, and set up the terminology used in this thesis.

2.1 Design Process

In this thesis, we focus on the conventional component-centered design process. This design process is the one by which a system is decomposed into interacting subcomponents. The process is recursively applied to every subcomponent until a point is reached where components are simple enough to be specified by means of either a state machine or a sequential specification. These simple components are called primitive components, while the others, i.e. the ones that are decomposed into interacting subcomponents, are called architectural components. The purpose of system decomposition is to provide a white box description, or specification, of a system that has been previously described as a black box. Thus, the white box description can be viewed as an implementation of the black box. The design process is illustrated in figure 2.1 (this figure has been taken directly from [Buh93]).

Figure 2.1 Component-centered design process

This type of design process is referred as a component-centered design process in [Buh93].

A component-centered design process is conducted as follows:

- 1- the different components of the system are defined,
- 2- the interface of each component is defined,
- 3- the component interactions are defined, and
- 4- components are described as black boxes.

The design steps conducted in the component-centered design process are illustrated in figure 2.2 (this figure has been taken directly from [Buh93]).

Figure 2.2 Design steps in component-centered design process

2.2 Design Method

A **design method** is a method, based on a particular description model, developed to support the design process, i.e. to support system development from requirements to implementation. The design method is based on three models: a **description model**, a **semantic model**, and an **analysis model**.

An illustration of the design model is given in figure 2.3. In this figure, we illustrate the sequential ordering of the different models included in the design method: the description model, the semantic model and the analysis model. The design method is given in relation with user requirements and implementation. In Figure 2.3, we also identify two particular arrows which are labelled: **interpretation method** and **feedback to architecture model**. The **interpretation method** (described in §2.3.2) is a method which enable the interpretation of the description model in terms of the semantic model. The description model together with the semantic model and the interpretation are regrouped under the **design model**. The **feed back to architecture model** arrow illustrates that after the analysis of the design description is performed in the analysis model, the designer may go back to the description model to make some correction on the design, or to extend and/or refine the functionality of the design. Therefore, incremental and cyclic approaches are allowed in such a design method. However, if the design is correct and complete, an implementation can be derived from the design after the analysis phase has been performed in the analysis model.

Figure 2.3 The Design method

2.3 Design Model

The **design model** is the central model of the design method. In this thesis, we use a two layer approach design model which include: an **description model** layer and a **semantic model** layer. The **design model** itself is composed of three components: the **description model**, the **semantic model** and an **interpretation method** which enables the interpretation of a given design, described using the description model, in terms of the target semantic model. An illustration of the design model is given in figure 2.4. The three main components of the design model are described in the following sections (§2.3.1, §2.3.2 and §2.3.3).

Figure 2.4 The Design model

2.3.1 Description Model

The **description model**, also called design description model, is a particular model in which systems can be described. It possesses constructs that allow the description of both the architecture and the behavior of the system. Description models are based on specific sets of notations (visual and/or textual) which serves to describe systems. Buhr's MachineCharts (description model of TimeBench) [Buh90], ObjecTime description model and EGL description model are examples of description models. There is a description model, which can be more or less precise, expressive and flexible, associated with each and every system's design.

In this thesis, we are interested in description models that uses visual, or pictorial, notations for architectural design description.

We define a design as the combination of both architecture and behavior (Design = Architecture + Behavior). Therefore as illustrated in figure 2.4, the description model is composed of two distinct models: the **architectural model** (§2.3.1.1) and the **behavioral model** (§2.3.1.2). Each of these is associated with its own type of description which are respectively **architectural description** (also called architectural decomposition) and **behavioral description**. These two types of description are used separately, i.e. at each decomposition step of the design, the designer can either describe a component by means of interacting subcomponents, which correspond to a architectural description, or describe the internal behavior of a component as a sequence of actions, which correspond to a behavioral description. We refer to this type of description model as a "two steps" description model. Using such description model, the description of a centralized control component can be achieved using both types of description. However, the two types of description still have to be used separately, i.e. the architectural description of the system is conducted and then the behavioral description of the centralized control can be achieved.

Case Study: The Elevator System

The MachineCharts Description Model

In this thesis, we use the MachineCharts [Buh90] description model for the design the elevator system. MachineCharts uses rich and expressive visual notations (MachineCharts architectural model) for architectural description, and allows the use of both sequential code (MCL) and state machines (ASMs) for behavioral description. These two MachineCharts models (architectural and behavioral) are described in their respective sections (§2.3.1.1 and §2.3.1.2) in relation with the elevator system.

A MachineCharts Design of the Elevator System

The elevator system design used in this thesis is based on the architecture given in figure CS2.2 (§2.3.1.1). This design is only one of many possible designs for the elevator system, different system decomposition and/or control allocation would lead to different designs. In [Buh93a], different control allocations of the same system decomposition are discussed. The design presented here has been chosen mainly because it possesses a

distributed control. The design chosen does not pretend to be any better than the other ones; the choice of a particular design for a given system has to be based on criteria (e.g. performance, distributed control allocation, etc.) that are not discussed in this thesis. The design steps which leads to the elevator system design used in this thesis are described in [Buh93a] using a timethreads approach. The timethreads design approach is not part of this thesis, interested readers may refer to [Buh92b], [Buh93b].

The system architecture and the behavior of each of its components are respectively described in §2.3.1.1 and §2.3.1.2. Also, a tree representation of the design is provided in §2.3.1.3.

2.3.1.1 Architectural Model

The **architectural model** is the part of the description model in which the system architectures are described. In this model, components (systems or subsystems) are described in terms of interacting subcomponents. In architectural description, only subcomponents and the interactions between them are described. Thus, an architecture is the composition of both components and interactions (architecture = components + interactions). The term **architectural artifacts** is used to refer to the set of architectural objects which includes both architectural components and interaction wires (links).

Interactions

In this thesis, the terms interaction and communication have the same meaning. In general, we use the term interaction because it is more abstract and because that is the term used in LOTOS. However, the term communication may also be used in some place in the thesis to mean the same thing. In practice, interactions can be achieved by any kind of communication schemes (rendez-vous, asynchronous communication, ...).

Case Study: The Elevator System

The MachineCharts architectural model

As previously mentioned, the MachineCharts architectural model uses rich and expressive visual notations. In this thesis, for the purpose of the elevator system, we only use a subset of these notations which includes: subsystems, processes, objects, buttons, mailboxes and wires. These notations, except wires which are represented by a simple lines, are illustrated in figure CS2.1.

Figure CS2.1 MachineCharts notations used in this thesis

These notations have the following semantics (see [Buh90]):

- subsystems are aggregation of other elements (including other subsystems) (e.g. packages in Ada, modules in Modula-2, units in Turbo Pascal, objects in object oriented programming languages and so forth),
- processes are active autonomous component (e.g. tasks in Ada),
- objects are data abstraction (e.g. Car_Status), procedural abstractions (e.g. Car_Controller), or both,
- buttons are unary operators that provide procedural (or functional) abstraction (e.g. procedures or functions in any programming language),
- mailboxes are asynchronous interface operators,
- engines are primitive, sequential machines that provide the motive power for the activity of the machines to which they belong, and
- wires serve to joins components to interface operators, indicating contractual paths for interaction.

For a complete MachineCharts description, see [Buh90]).

In this case study, the word "button" is overloaded; it is used with two different meanings: a "MachineCharts button" and an "elevator system button" (car button or floor button). However, we may distinguish them in the following way: when used alone the word "button" refers to a "MachineCharts button"; if we want to refer to a button in the elevator system, we either use the term "car button" or "floor button".

Description of the architecture

Since the objective of the elevator system problem consists in designing the strategic control of the elevator system, the elevator system has been decomposed in two parts: the **Strategic Control** and the **Tactical Control**. The scope of the elevator system problem proposed here is the design of the Strategic Control. Therefore, the Tactical Control entities (Cars and Building) constitutes the environment of the Strategic Control, and for this reason the Building and Cars are considered to be black boxes in the elevator system design.

In figure CS2.2, a MachineCharts architecture for the elevator system is given. In this architecture, we observe that the Strategic Control has been decomposed in two different types of subsystems: Coordination subsystem and Car_SC subsystem. There is one instance of the Coordination subsystem and multiple instances of the Car_SC subsystem (one for each car). The communication between the Tactical Control and the Strategic Control is realized as follows:

- the Building subsystem communicates with the Coordination subsystem via the fb wire,
- each Car subsystem communicates with the Coordination subsystem via a cs wire, and
- each Car subsystem communicates with its peer Car_SC subsystem via a cb wire.

The different architectural components of the system are described hereafter (their behavior is described in §2.3.1.2).

Legend:

fb: Floor Button

fb_rep: Floor Button Report

cb: Car Button

cb_rep: Car Button Report

cs: Car Status

new_cs : New Car Status

msg: message

msg_rec: message received

Report_fb: Report Floor Button

Change_cs: Change Car Status

Rec_msg: Receive message

Report_cb: Report Car Button

Figure CS2.2 A MachineCharts architecture of the elevator system

The Coordination subsystem

The Coordination subsystem is responsible of the external control of the system. It communicates with both types of tactical control entities (Building and Cars). It receives both pending floor requests from the Building tactical control entity via the fb wire, and car status information from the Car tactical control entities via cs wires. There is a cb wire between the Coordination subsystem and every Car. The Coordination subsystem also communicates with every Car_SC subsystem. It sends information to a given car via the msg wire to inform a car whenever a particular action (Move or ServeFloor) is required from that car. It also sends car status information to Car_SC subsystem using the same wire when the status of the car changes.

Internal Composition

The Coordination subsystem is composed of:

Components:

- the Dispatch process,
- the Pending_Floor_Requests object (data abstraction object), and
- the All_Car_Status object (data abstraction object),

ports:

- a Change_cs button, and
- a Report_fb button,

wires:

- fb_rep (internal between the Report_fb button and the Dispatch process),
- new_cs (internal between the Change_cs button and the Dispatch process), and
- msg (external between the Dispatch process and the Car_SC subsystems).

The Dispatch process

The Dispatch process is the control entity of the Coordination subsystem. It receives inputs (fb_rep and new_cs) via its own mailbox port and delivers outputs via the msg wire. It also has implicit access to both objects contained in the Coordination subsystem, i.e. the Pending_Floor_Requests object and the All_Car_Status object.

The Car_SC subsystems

There exists one Car_SC subsystem for each car in the elevator system. The Car_SC subsystems are responsible of the internal control of each individual car. Each of them communicates with both its associated car tactical control and the coordination subsystem. It receives action messages (Move or ServeFloor) and car status information from the coordination subsystem via the msg wire, and pending car requests from the Car tactical control entities via the cb wire.

Internal Composition

The Car_SC subsystem is composed of:

Components:

- the Control process (described in 2.1.1),
- the Pending_Car_Requests object (data abstraction object), and
- the Car_Controller object (procedural abstraction object),

ports:

- a Rec_msg button, and
- a Report_cb button,

wires:

- msg_rec (internal between the Rec_msg button and the Control process), and
- cb_rep (internal between the Thesis_cb button and the Control process).

The Car_Controller object is a procedural abstraction object which issues ServeFloor and Move commands (and perhaps also stores the state of the car it control).

The Control process

The Control process is the control entity of the Car_SC subsystem. It receives inputs (msg_rec and cb_rep) via its own mailbox port. Also, it has implicit access to both objects contained in the Car_SC subsystem, i.e. the Pending_Car_Requests object and the Car_Controller object.

Architecture vs. Structure

In this thesis, the term architecture, or architectural design, is used in a general manner to refer to system designs in which only components and interactions between them are described. Other people in the software community often use the term "structure" to mean more or less the same thing. In this thesis, we use the two terms with different meanings. In fact, we define the structure as one of the three aspects which composes an architecture; the other being the component implied behavior and the interaction implied behavior imposed by the architecture on the total behavior of the system.

By **implied behavior**, we mean the implicit behavior which is attached to an architectural artifact. In description models in which architectural artifacts have implied behavior, the complete behavior of an artifact is obtained by the composition of both the implied behavior of the artifact and the (possibly empty) internal behavioral description of the artifact.

For example, in MachineCharts actors, subsystems, objects and ports are all visually represented by different icons (see figure CS2.1), and the shape of an icon partially express the behavior of the artifact to which it corresponds. Also, in MachineCharts, synchronous and asynchronous interaction links (wires) are visually differentiated, and in this case the behavior of an interaction link is completely expressed in its visual representation (there no behavioral description of links). Thus, in MachineCharts both components and interaction links have implied behavior. In EGL, interaction links have implied behavior, which is synchronous (rendez-vous) in all cases, but components (processes) are abstract components with no implied behavior. The behavior of a component (process) is completely determined by its internal description, which can be architectural and/or behavioral description. In ObjecTime, neither interaction links (ports and bindings) nor components (actors) have implied behavior, and therefore the behavior of any architectural artifact is completely described by means of internal description.

Figure 2.5 gives an illustrations of the existing relation between the architecture and its three different aspects. This figure illustrates the structure (structural design) as a projection of the architecture (architectural design) on the component-interaction, also called wiring, plane. As illustrated in this figure, the structure is a representation of the architecture in which components and interactions have no implied semantics. In this way, the structure is a "pure" wiring abstraction of the architecture. It is composed of: black box components and abstract interaction links.

Figure 2.5 Projection of architectural design on structural design

The following table (table 2.1) gives the result of the projection of different architectural model on the three different planes of the previous diagram (figure 2.5). In this table, "yes" is written in a cell (x,y) if the projection of the architectural model (x) on the plane (y) gives a non-empty result; otherwise "no" is written. As discussed above:

- EGL has both structure and interaction implied behavior (which is rendez-vous),
- ObjecTime is pure structure, i.e. there is no implied behavior at all, and
- MachineCharts has structure, component implied behavior and interaction implied behavior.

Architectural Models			
	EGL	ObjecTime	MachineCharts
Structure	yes	yes	yes
Component Implied Behavior	no	no	yes
Interaction Implied Behavior	yes	no	yes

Table 2.1 Projection of different architectural models on the different planes

In the case where the architectural model only has structure (no implied behavior), e.g. **ObjecTime**, we say that the architectural model is degenerated to a structural model. In this case we may say that the architectural model is in fact a structural model. However, for the purpose of generality, we use the term architectural model in all cases, even in the case of degenerated architecture (structural model). The analysis that we make in this thesis for architectural models may also apply to the particular case of structural models.

Architectural Representation

We use the term architectural representation to refer to partial representations of architectural designs. By partial representation, we mean a representation of the system in which only certain aspects, or viewpoints, of the system are described. In the sequel, **AR** is used as an acronym for **Architectural Representation**.

Thus, an **AR** can be viewed as a projection of an architectural design on a certain space (n-dimensional) in which the system can be represented. The space, in which the architectural design is projected, is composed of the different aspects of the system that we want to represent. The maximum space in which the system can be projected is the complete space in which the architectural design is described, i.e. the space which includes all aspects of

the system, and the minimum space is the one containing only one aspect of the system. In this thesis, we are interested in the communication (interaction) aspect of distributed systems, and thus we use the term AR with respect to the communication (interaction) aspect of architectural design.

2.3.1.2 Behavioral Model

The **behavioral model** is the part of the description model in which the temporal ordering of actions which are executed by a single component is described. In the behavioral description, the temporal ordering of actions is explicitly expressed. Component behavior is usually described by means of either state machines or sequential code. In LOTOS terms, the behavioral description of a component can be performed using monolithic or state-oriented styles.

Case Study: The Elevator System

The MachineCharts Behavior model

As previously mentioned, the MachineCharts behavior model allows the use of either MachineCharts language (MCL) or MachineCharts state machines (ASMs) for behavior description. However, because the focus of this thesis is on architectural description and not behavioral description, we choose, in order to reduce the background knowledge required to read this thesis, not to introduce MachineCharts behavioral description model in this thesis. Therefore, we describe component behavior using textual description rather than MachineCharts code or MachineCharts state machines.

Description of the Behavior

For lack of space and because behavioral description is not a main concern in this thesis, we choose not to give the behavioral description of every architectural component contained in the elevator system design (figure CS2.2). However, as an example, we give the textual behavioral description of the Dispatch process.

The Dispatch process

The behavior of the Dispatch process may be described as follows. One responsibility of the Dispatch process is to keep the objects (data objects Pending_Floor_Requests and All_Car_Status) up to date. Therefore, each time a message is received, the Dispatch process must update the appropriate objects.

Also each time a car status (cs) message is received, the Dispatch process must decide what to do. If the status of the car is MovingReady, then the Dispatch process must decide if the car must serve the next floor or not. If the status of the car is StoppedReady, then the Dispatch process must decide if the car should move or not. Also, after receiving car status (cs) information from a given Car, the dispatch process may decide, in order to apply a global strategy, to move another car. In all cases, if the decision taken by the Dispatch process requires a car to do an action (ServeFloor or Move), then it communicates with the appropriate Car (tactical control) subsystem by sending it a message via the msg wire (to the Rec_msg interface of the Car (tactical control)). In the case where no action is required from the car that sent a car status message, the Dispatch process must still communicate with the Car_SC associated with that car to inform it of the new status of its car. Thus if the car has to stop at the next floor because it has a pending car request for that floor, it will be able to do so (we recall that the car status information is only sent to the Coordination subsystem).

As mentioned in the description of the elevator system problem, the decision algorithm is outside the scope of our system design.

2.3.1.3 Design Tree Representation

In a conventional design process (component-centered design process [Buh93]), the objective of the process is to decompose components into interacting subcomponents until a point is reached where components are simple enough to be described by means of behavioral description. These simple components are called primitive components, and we define as a **primitive component** any component which is sufficiently simple to be describe by means of behavioral description, i.e. a component for which no further architectural description (decomposition) is needed.

An abstract example of an architectural description of a system is illustrated in figure 2.6. In this example, rectangles represent architectural components and rectangles filled with an ellipse represent primitive components. Architectural components are labelled S_i and primitive components are labelled P_j . The thick black lines (labelled $g_1, \dots, g_{10}$) between components represent interactions.

Figure 2.6 Abstract example of architectural design

Also, the architectural design of a system can be represented by means of a design tree representation, in which the leaves (called behavioral nodes) represent primitive components and all other nodes (called architectural nodes) represent architectural components. Primitive component are described by means of behavioral descriptions, and architectural components are described by means of architectural descriptions. Thus, in design tree representations, every architectural node must keep the information concerning the wiring (interactions) between its subcomponents. This wiring information is kept in the form of architectural expressions (§2.3.2.1). An example of design tree representation is given in figure 2.7. The design tree representation illustrated in figure 2.7 corresponds to the system given in figure 2.6. In this example, we use the same visual notations as in figure 2.6, i.e. rectangles represent architectural components and rectangles with internal circle represent primitive components.

Figure 2.7 Abstract example of design tree representation

We use design tree representation for internal representation. This type of representation is particularly useful for the composition of the total behavior of systems (see § 2.3.2.3).

Case Study: The Elevator System

Design Tree Representation

In figure CS2.3, a design tree representation of the elevator system MachineCharts design (figure CS2.2) is provided. In this figure, MachineCharts notations are used to illustrate the different nodes of the tree. The nodes are labelled to correspond to the architectural components of the elevator system design illustrated in figure CS2.2.

In this figure, we observe that engines (square box in which an "X" is drawn) are illustrated which was not the case in figure CS2.2. Engines are usually hidden in architectural designs. However, they constitute important components of the design, and thus they have to be represented in the tree representation of the design.

2.3.2 Interpretation Method

The **interpretation method** (see figure 2.4) is the part of the design model in which the whole system described in the description model is translated in a **total behavior** in the semantic model. Therefore, the interpretation method is a translation method between a given description model and a given semantic model, and thus its definition has to be governed by both the description model and the semantic model. In fact, there exists a specific interpretation method for every different pair of <description model, semantic model>.

In the interpretation method, we distinguish three distinct methods: the **architectural interpretation method**, the **behavioral interpretation method** and the **composition of total behavior method**. Also, two types of intermediate expressions

are used: **architectural expressions** and **behavioral expressions**. The three methods and the two types of intermediate expressions are described hereafter.

2.3.2.1 Architectural Expressions

Since the objective of the interpretation method is to produce the total behavior of the system in a particular semantic model, the choice of the type of architectural expressions used in the interpretation method is governed by the type of semantic model used in the design model. For example, if LOTOS is the intended semantic model, LOTOS expressions can be used as architectural expressions in the interpretation method.

Architectural expressions describe how subcomponents interact together to provide the intended behavior of the component. In LOTOS, architectures can be described using component-oriented style. Also, in LOTOS, architectural expressions are described using high level operators, e.g. parallel operators, enable operator and disable operator. In the sequel, **AE** is sometimes used as an acronym for **Architectural Expression**.

2.3.2.2 Architectural Interpretation Method

The **architectural interpretation method** is the part of the interpretation method in which every architectural component is interpreted in terms of architectural expressions. The architectural interpretation of a component may also have a direct impact on the generation of the internal behavior of the component, this is illustrated in figure 2.4 by the dotted oriented line which links the architectural interpretation method to behavioral expressions. In fact, in cases where the component has implied behavior, the behavior of the component is obtained by a composition of both the implied behavior and the internal behavioral description of the component. This type of interpretation is also provided by the architectural interpretation method. Thus, the architectural interpretation method is the part of the interpretation method in which:

- 1°) every architectural component is interpreted in terms of architectural expressions (subcomponents and interaction between them), and
- 2°) the implied semantics of every type of component is interpreted.

The cross link (the one between the architectural interpretation method and the behavioral expressions) appears in an interpretation method when there is incompatibility between the

types of component defined in the description model and the ones defined in the semantic model, i.e. when there exists a type of component defined in the description model for which there exist no direct mapping in the semantic model. In this case, the component must be mapped to a more abstract type of component of the semantic model, and the implied behavior of the component must be composed together with the component behavioral description to give the behavior of the newly created component in the semantic model.

For example in MachineCharts, the mailbox icon has an implied behavior that partially determines the behavior of the mailbox component. In fact, the behavior of the mailbox is a composition of both the implied behavior of the icon and the internal behavioral description of the mailbox. On the other hand, in LOTOS, there exists only one type of abstract component which is called *process*. Therefore, if MachineCharts is the description model and LOTOS the semantic model, mailbox icons will be mapped into LOTOS processes, and the behavior of the mailbox LOTOS processes will be partially determined by the implied behavior of the MachineCharts mailbox icon.

2.3.2.3 Behavioral Expressions

As in the case of architectural expressions, the choice of the type of behavioral expression used in the interpretation method is governed by the type of semantic model used in the design model. For example, if LOTOS is the intended semantic model, LOTOS expressions can be used as behavioral expressions in the interpretation method.

Behavioral expressions describe the internal behavior of components. Since component behavior is usually described by means of either state machines or sequential code, it can be directly expressed in LOTOS using either the state-oriented or the monolithic style. Thus, by LOTOS behavioral expressions we mean LOTOS expression written in either state-oriented or monolithic style. In the sequel, BE is sometimes used as an acronym for **Behavioral Expression**.

2.3.2.4 Behavioral Interpretation Method

The **behavioral interpretation method** is the part of the interpretation method in which component behavior described in the behavioral description model are interpreted in terms of **behavioral expressions**.

However in some cases, the behavioral interpretation of components may have an impact on the architecture of the system in the semantic model. This is illustrated in figure 2.4 by the dotted oriented line that links the behavioral interpretation method to architectural expressions.

This may happen for example if the description model uses highly abstract architectural notation, i.e. notation that has no implied behavior. In this case, the architecture is degenerated to a structure which only describe inter-component wiring, and the behavior of every architectural component, including communication wires, has to be described by means of behavioral description. Such description model has the advantage to be very flexible, it allows the description of any type of components (there exist no pre-defined ones) and any type of inter-component communications. In this case, the designer may create his/her own type of components and her/his own type of inter-component communications. ObjecTime is an example of such description model. The interpretation of such description model may result in the production of extra components in the architecture.

For example, if LOTOS, which only provides synchronous (rendez-vous) inter-component interactions, is the semantic model, the interpretation of a communication wire (called bindings in ObjecTime) that has been defined, at the behavior level, as an asynchronous communication wire will result in the creation of an extra buffer (mailbox) process in the LOTOS architectural expression. Thus, as it is illustrated in figure 2.8, instead of having component C1 interacting with component C2 via the asynchronous wire *a* in the description model design, we have, in LOTOS semantic model, process C1 interacting (rendez-vous) on gate *a1* with a process Buffer which also interacts (also rendez-vous) on gate *a2* with process C2. Thus, the interaction semantics of the design has been preserved through the LOTOS interpretation, but an extra process has been created in the LOTOS design.

Figure 2.8 Example of architectural effect of behavior interpretation

2.3.2.5 Composition of Total Behavior

The **composition of total behavior method** is the part of the interpretation method in which the **total behavior** of the system is produced. This composition is conducted using a constructive bottom-up approach based on design tree representations. The total behavior is composed progressively from the behavioral expressions of the primitive components (leaves of the tree) to the global architectural expression of the system situated at the top of the tree. At each architectural node *N* of the design tree representation, the method consists of taking all descendant-nodes of *N* and compose them in the architectural expressions associated with *N*. An example of a composition of total behavior method for LOTOS is provided in §4.2.

2.3.2.6 Definition Steps

In order to define an interpretation method for a given description model and a given semantic model, we have to conduct the three following steps:

- 1°) choose a type of architectural expression and a type of behavioral expression (the choice of these type of expressions have to be governed by the type of semantic model towards which the interpretation method is oriented),
- 2°) define both an architectural interpretation method and a behavioral interpretation method,
- 3°) define a method for the composition of the total behavior.

Case Study: The Elevator System

An example of interpretation method definition is provided in §4. The example discussed the definition of a LOTOS interpretation method in relation with the elevator system MachineCharts design.

2.3.3 Semantic model

The **semantic model** constitutes the underlying model associated with a particular architecture model. In the semantic model, the whole design of the system (architecture and behavior) is expressed in terms of the semantic model. The resulting behavior of the whole system is called the **total behavior**.

The relation between design models and semantic models may be of the type many-to-many, i.e. several design models can use the same semantic model and a design model can be translated into several semantic models.

Case Study: The Elevator System

In the case study described in this thesis, the FDT LOTOS is used as a semantic model for the elevator system MachineCharts design.

2.4 Analysis Model

The term analysis model is used, in this project, to refer to any set of design analysis techniques that are used by the design method. These techniques may include: symbolic execution, simulation, model checking, formal verification, etc.

Example of analysis models in relation with existing design tools are: animation (simulation) and timelines in TimeBench, any LOTOS tools in EGL, and animation (simulation) in ObjecTime.

Chapter 3 A Framework for Formal Method Integration

As mentioned in the introduction (§1), the main objective of this project is to progress towards the definition of a framework for the integration of formal methods in the design process of distributed systems. In this chapter, we:

- 1°) illustrate how interpretation methods can be used to integrate formal methods in the design process of distributed system, and
- 2°) illustrate how interpretation methods can be used to integrate LOTOS in the design process of distributed systems.

We illustrate formal method integration in the context of the design method diagram previously presented in §2.2.

3.1 Formal Method Integration

This thesis proposes to integrate formal methods in the design process by means of interpretation methods that allow formal design interpretation. The problem is the following: designs are given using a description model X (possibly informal), but have to be interpreted in terms of a formal semantic model Y. A solution to this problem will allow the designer working with a design method X to have access to both the analysis model X and the formal analysis model of formal method Y (analysis model Y). Therefore, the addition of a formal semantic model Y to a design method X would improve the design process by providing the designer with a formal analysis model (the advantages of using formal methods are discussed in §1.2).

Figure 3.1 illustrates such an integration. We see from this figure that the designer who uses design method X may use, in addition to the analysis model X, a formal analysis

model provided by the environment of the formal method Y. Consequently, the design process associated with the design method X is improved by the addition of a formal analysis model. Also, we observe that the design of the system is still developed in the same description model. Thus, we say that such framework leads to a soft-integration of formal methods in the design process. Therefore, improvement can be made in distributed system design without having to change the whole design methodology.

In figure 3.1, the "Feedback to Architectural Model" arrow, which is labelled 2, indicates the flow of information which is send back to the design method X after the analysis has been performed in the analysis model of formal method Y. This may involve modification of the design if errors were discovered in the design by using the analysis model of formal method Y.

Legend:

- ① Interpretation Method
- ② Feedback to Architectural Model

Figure 3.1 Integration of formal method Y in design method X

Also, in order to define a multi-paradigm design process, we could generalize this approach and define a set of interpretation methods in which each interpretation method gives access to a different formal method (ultimately one for each different aspect, or viewpoint, of the system). Thus, we could analyse the different aspects of the system with appropriate analysis models.

3.2 LOTOS Integration

We may apply the framework defined in the previous section (§3.1) for the integration of LOTOS in the design process. In figure 3.2, a LOTOS interpretation method is illustrated in relation with a design method X. The figure is separated in two parts: the right hand side part illustrates a design method X, and the left hand side part illustrates formal environments: that is, LOTOS environment and Y-formalism environment, where Y-formalism stands for any formalism on which LOTOS specifications can be mapped. Such methods for the translation of LOTOS specifications into Y-formalism specifications already exist for: Petri-Nets ([Che92]), Galileo-Nets ([Mar88]), and Extended Finite State Machine (XFSM) ([Kar92]). In this way, LOTOS specifications constitute the entry to many formal methods related to the communication aspect of distributed system design.

Figure 3.2 Integration of LOTOS in design method X

We see from this figure that a designer working with the design method X does not have to change his/her methodology in order to benefit from LOTOS formal verification methods, which are part of the LOTOS analysis model, or Y-formalism analysis model. The interpretation method is responsible for the interpretation of designs, developed in the description model, in LOTOS terms. Therefore, the designer can use the analysis models included in her/his design method, but also he/she can use analysis models included in LOTOS methods and Y-formalism methods.

Case Study: The Elevator System

The use of LOTOS as a semantic model for the elevator system MachineCharts design will enable us to analyse the design using the LOTOS formal analysis model. In fact, the use of LOTOS as a semantic model will allow us to use the whole set of existing LOTOS tools for

the formal verification of the design. Also, the designer may still use the MachineCharts analysis model for design verification.

Thus, we do not have to change the MachineCharts design methodology to use LOTOS formal verification methods in MachineCharts designs.

Chapter 4 Definition of Generic Formal Interpretation Methods

In the previous chapter, we discussed the integration of formal methods in a given design method X by means of formal interpretation methods. In this chapter, we analyse the issues in defining formal interpretation methods.

The main objective of this chapter is to define, for a given semantic model (say Y'), a generic formal interpretation method which is sufficiently abstract to serve as a framework for the definition of formal interpretation methods between different types of description models and Y' .

As in the previous chapter, we first analyse the issues in terms of an abstract formal method, and then apply it for the definition of a LOTOS interpretation method.

4.1 Generic Formal Interpretation Methods

Before we discuss the issues in defining generic formal interpretation methods, we need to analyse the way design models are defined. In figure 4.1, we illustrate three abstract design models based on three different description models: A , B and C . Each of these description models has its own associated semantic model: description model A is associated with semantic model A' , description model B is associated with semantic model B' and description model C is associated with semantic model C' . Also, for each of these pairs (description model - semantic model), there is an interpretation method which has been defined to allow the interpretation of designs given using a description model X in terms of its associated semantic model X' . The interpretation methods are represented in figure 4.1 by $IM_{AA'}$, $IM_{BB'}$ and $IM_{CC'}$.

Legend:**IM:** Interpretation Method

Figure 4.1 Design models

However, in the general case, interpretation methods only exist between a given description model X and its associated semantic model X' ; there is no interpretation method defined to allow the interpretation of a description model X in terms of another semantic model, say Y' . In figure 4.1 for example, there is no interpretation method between description model A and semantic model B'.

Therefore, in order to integrate a given formal semantic model Y' in a design model (method) based on a description model X , we need to define an interpretation method which enables the interpretation of X in terms of the formal semantic model Y' . More generally, we may want to integrate formal semantic model Y' in several design models. Such a case is illustrated in figure 4.2 in which A, B and C are three different description models that we want to interpret in terms of the formal semantic model Y' . Interpretation methods $IM_{AY'}$, $IM_{BY'}$ and $IM_{CY'}$ are the ones that we need to define in order to integrate formal method Y' in design models A, B and C.

Legend:

IM: Interpretation Method

Figure 4.2 A generic interpretation method in relation with design models

In order to conduct such an integration of formal semantic model Y' in several different design models, we may take advantage of defining a generic interpretation for Y' . In this thesis, the term **generic method** refers to a method template in which some of sub-methods, and/or sub-parts, have been completely defined, while others have only been partially defined to allow redefinition in precise contexts. In this way, a generic interpretation method for Y' can be used for the definition of specific interpretation methods between description models and Y' .

Our objective, here, is to define a generic Y' interpretation method which provides a maximum of reusability for the definition of specific interpretation method between any description model X and the semantic model Y' . We recall that in order to define an interpretation method for a given description model and a given semantic model, we have to conduct the three following steps (§2.3.2.4):

- 1°) choose a type of architectural expression and a type of behavioral expression (the choice of these type of expressions have to be governed by the type of semantic model towards which the interpretation method is oriented),
- 2°) define both an architectural interpretation method and a behavioral interpretation method,
- 3°) define a method for the composition of the total behavior.

In these three steps, we observe that step 1° and 3° may be defined independently of the description model; their definitions may be completely determined by the semantic model Y' . Therefore, we say that step 1° and 3° constitute the constant part of a generic interpretation method associated with a semantic model Y' . Thus, in order to define an interpretation method between any given description model X and semantic model Y' (for which step 1° and 3° have already been defined), we only have to conduct step 2°, i.e. define an architectural interpretation method and a behavioral interpretation method. The definition of both interpretation methods has to be governed by the type of expressions (architectural and behavioral) used in the interpretation method.

4.2 A Generic LOTOS Interpretation Method

In order to define a generic interpretation method for LOTOS, we need to analyse LOTOS interpretation methods in relation with the three steps previously discussed. For this purpose, a LOTOS interpretation method is illustrated in figure 4.3 in relation with an abstract description model.

Figure 4.3 A generic LOTOS interpretation method

4.2.1 Definition of the Generic LOTOS Interpretation Method

1°) Architectural and Behavioral Expressions

In this generic LOTOS interpretation method, LOTOS expressions are used for both architectural and behavioral expressions.

2°) LOTOS Architectural and Behavioral Interpretation Methods

LOTOS Behavioral Interpretation Method

Since some methods already exist for the visual description and LOTOS interpretation of behaviors, we do not develop such a method in this thesis. Most of the existing methods for the visual description and interpretation of behaviors are based on state machines. For

example, [Lamouret91] and [Obaid91] defined such methods for the description and LOTOS interpretation of behavior.

LOTOS Architectural Interpretation Method

In the definition of the architectural interpretation method part of any generic formal interpretation method, we need to consider the fact that the formal architectural interpretation method must allow the interpretation of different types of architectural models, and therefore different type of architectural artifact, in terms of the chosen type of architectural expressions.

In relation with LOTOS, this means that any LOTOS architectural interpretation method must allow the interpretation of different architectural models in LOTOS terms. The definition of LOTOS architectural interpretation methods is the subject of the next chapter (§5). Issues for such a definition are discussed in details in that chapter.

3°) Composition of Total Behavior Method

Based on the fact that both the architectural and the behavioral expressions are LOTOS expressions, the composition of total behavior method is reduced to a straightforward composition of both LOTOS architectural expressions and LOTOS behavioral expressions. The composition of the LOTOS expressions (architectural and behavioral) is based on the design tree representation of the design using the constructive bottom-up approach described in §2.3.2.

Hereafter, a LOTOS specification corresponding to the design tree representation of figure 2.7 is given. In the composition of this specification, we consider that every node contained in figure 2.7 is described by means of LOTOS expressions (architectural and behavioral). The LOTOS specification given above is written in pseudo-LOTOS. The purpose here is to give the skeleton of a LOTOS specification which may be obtained from the tree representation of a design. The method informally described here by means of an example can be used as a general composition of total behavior method for LOTOS.

In the following specification, AE is used as an acronym for "architectural expression", and BE is used as an acronym for "behavioral expression". Thus, AES refers to the architectural expression that describes the internal architecture of component S, and in the same way

BEP refers to the behavioral expression that describes the internal behavior of primitive component **P**.

```
specification S :=  
  AEs  
where  
  process S1 := AES1  
    where  
      process S11 := AES11  
        where  
          process P111 := BEP111 endproc  
          process P112 := BEP112 endproc  
          process P113 := BEP113 endproc  
        endproc  
      process P12 := BEP12 endproc  
      process P13 := BEP13 endproc  
    endproc  
  process P2 := BEP2 endproc  
  process S3 := AES3  
    where  
      process P31 := BEP31 endproc  
      process P32 := BEP32 endproc  
    endproc  
endspec
```

Specification 4.1 Derivation of the skeleton of a LOTOS specification from the design tree representation

Case Study: The Elevator System

LOTOS interpretation method for MachineCharts

In §4.2, the definition of a generic LOTOS interpretation methods has been discussed. Now, we discuss the definition of a LOTOS interpretation method for MachineCharts design. The method is illustrated in figure CS4.1. Since step 1° (choose of both architectural and behavioral expressions) and 3° (definition of a total behavior composition method) of interpretation method definition are completely determined by the semantic model, the definition of a specific LOTOS interpretation method for MachineCharts designs only requires the description of step 2°, i.e. the definition of both architectural and behavioral interpretation method.

These internal interpretation methods (architectural and behavioral) must allow the interpretation of both architectural and behavioral MachineCharts designs in LOTOS terms. However, since LOTOS behavioral interpretation methods already exists, we concentrate, in this thesis, on the definition of a LOTOS architectural interpretation method for MachineCharts architectural designs. Also, we observe in figure CS4.1 that LOTOS behavioral interpretation of MachineCharts behavioral description has no effect on the LOTOS architecture, i.e. there is no dotted line between LOTOS behavioral interpretation method and LOTOS architectural expressions. That is because the behavioral description of MachineCharts can be directly interpreted in LOTOS terms using either state machine LOTOS interpretation ([Lamouret91], [Obaid91]) or sequential code LOTOS interpretation.

One of the two main issues in the definition of a LOTOS architectural interpretation method for MachineCharts architectural designs consist in the interpretation of rich and expressive architectural design artifacts in terms of abstract ones; MachineCharts has many different type of artifact for both architectural components and interactions while LOTOS only has one type of component (process) and one type of interaction (rendez-vous). Therefore, the LOTOS interpretation of a MachineCharts architectural component will not only result in the creation of a LOTOS process, but may also partially determine the behavior of the component (that is the case for MachineCharts mailbox components). In the same way, the LOTOS interpretation of a MachineCharts architectural interaction will not only result in the creation of a LOTOS interaction gate, but may also result in the creation of an extra LOTOS process (that is the case for MachineCharts jab (asynchronous) wire). This is illustrated in

figure CS4.1 by the dotted line. The other main issue in the definition of a LOTOS architectural interpretation method for MachineCharts architectural designs is related to the generation of LOTOS algebraic expressions from the 2-dimensional graph representation of MachineCharts design. In the next chapter (§5), these two issues are discussed in details and a solution for the definition of a LOTOS architectural interpretation method is presented.

Figure CS4.1 A LOTOS interpretation method for MachineCharts designs

Chapter 5 LOTOS Architectural Interpretation Methods

In this chapter, we present a solution for the definition of a LOTOS architectural interpretation methods based on the used of a particular type of intermediate model called LARG. We first give an overview of the solution in §5.1 and then, we describe the nature of the LARG intermediate model in §5.2. The LARG model is described in details in §6.

5.1 Solution Overview

As previously mentioned, the main objective of the LOTOS architectural interpretation method is to allow the generation of LOTOS expressions from different types of architectural descriptions. In order to achieve this goal, we propose to introduce an intermediate model (LARG) between the architectural models and the target architectural expression (see figure 5.1). This will allow the decomposition of the LOTOS architectural interpretation method into two distinct steps:

- 1°) the mapping of the architectural design on LARGs which is identified by "AM Method" (the AM method is described here above) in the figure , and
- 2°) the generation of LOTOS architectural expressions from LARGs which is identified by "LAEG Method" in the figure (the LAEG method is the subject of §8).

Legend:

AM: Architectural Mapping

LAEG: LOTOS Architectural Expression Generation

Figure 5.1 Solution for LOTOS architectural interpretation method

5.1.1 The AM Method

We see from figure 5.1 that the AM (Architectural Mapping) method is the one responsible for both the creation of LARGs from architectural design and the interpretation of the implied behavior of architectural artifacts. In fact, the AM method decomposes every architectural artifact of the design into an abstract LOTOS artifact (process or interaction gate) part and an implied behavior part. As previously shown (§2.3.2 figure 2.8), the implied behavior may also result in the creation of an extra LOTOS process. The LOTOS artifacts are sent to the LARG intermediate model (illustrated in figure 2.4 by a simple oriented line), while the implied behavior is sent to LOTOS behavioral expressions (illustrated in figure 2.4 by a dotted oriented line) where it can be composed together with the internal behavioral description of the component to give the internal behavior of the LOTOS process.

However, because we now want to concentrate on the LARG model, and consequently with the part of the AM method which is linked to the LARG model, we will omit, in the sequel of this chapter, to draw the link (dotted line) between the AM method and LOTOS behavioral expressions. We concentrate on the path that goes from the architectural model to LOTOS architectural expressions, passing successively by the AM method, the LARG intermediate model and the LAEG method.

Case Study: The Elevator System

Definition of an AM method for MachineCharts architectural designs

The formal and complete definition of an AM method for MachineCharts designs, or any other description model, is not the subject of this thesis. It constitutes a research area for future works. However, in this chapter, we identified two main issues in relation with AM method definition:

- 1°) the interpretation of the implied behavior of architectural artifacts, and
- 2°) the generation of LARGs from architectural designs.

Thus, in this thesis, we do not define an AM method for MachineCharts architectural design. However, we, in the case study part of §6.1, give the result of the architectural mapping of the elevator system MachineCharts design (figure CS2.2) on LARGs, and we discuss the two issues described above in relation with this architectural mapping.

5.1.2 Generic LOTOS architectural interpretation method

In order to define a generic LOTOS architectural interpretation method, we need to define an intermediate model (LARG) into which different types of architectural models can be mapped, and from which LOTOS architectural expressions can be directly generated. In this way, the LARG intermediate model constitutes the angular stone of our LOTOS

architectural interpretation method. The introduction of the LARG intermediate model in architectural interpretation methods is illustrated in figure 5.2.

Legend:

AM: Architectural Mapping

LAEG: LOTOS Architectural Expression Generation

Figure 5.2 LARG intermediate model in relation with generic architectural models

In figure 5.2, we observe that the introduction of the intermediate model have effectively divided the LOTOS architectural interpretation method into two distinct parts:

- 1°) the definition of an AM method which allow the mapping of architectural designs on the intermediate model, these are identified by AM_A , AM_B and AM_C in the figure, and
- 2°) the definition of an LAEG method which allow the generation of a LOTOS architectural expressions from the LARGs.

Therefore, since for a given intermediate model the LAEG is always the same, the definition of a LOTOS architectural interpretation method, and consequently the definition of a generic LOTOS interpretation method, for a given description model is reduced to the definition of an AM method.

5.1.3 Application of the Generic LOTOS architectural interpretation method to EGL, MachineCharts and ObjecTime

In particular, we want to apply the intermediate model solution to EGL, MachineCharts and ObjecTime architectural models. This is illustrated by figure 5.3. We see from this figure that the development of the LOTOS intermediate model would enable the generation of LOTOS architectural expressions from any of the three architectural models (EGL, MachineCharts and ObjecTime). Therefore it would enable a soft integration of LOTOS into conventional design methods. To integrate LOTOS in these design methods we only have to define architectural mapping methods AM_{EGL} , AM_{MC} and AM_{OT} .

Legend:

AM: Architectural Mapping

LAEG: LOTOS Architectural Expression Generation

Figure 5.3 The LARG model in relation with EGL, MachineCharts and ObjecTime

In this thesis, we show that the definition of an architectural mapping method is possible for MachineCharts designs (AM_{MC}) (see case study section at the end of §6.1.2). Also, since EGL and LARG use the same type of architectural artifacts, i.e. processes and synchronous

interactions, the definition of such a method for EGL (AMEGL) is straightforward. However, since the LARG model is intended to model the static part of architectural designs, the definition of an architectural mapping method between description models like ObjecTime, in which the dynamic creation of components is possible, and LARGs seems to be a lot more complex. Some concepts used in ObjecTime, like "multiple part of" and dynamic creation of actor, can not be represented in LARGs, and thus LARGs can only partially represent ObjecTime designs. However, the method proposed here can still be used to analyse an ObjecTime design by parts, and then verify properties in specific part of the design. A discussion of the conceptual differences between LOTOS and ObjecTime (Telos) is provided in [Lam92].

5.2 Nature of the LARG Model

In the previous section (§5.1), we discussed the reasons for developing an intermediate model in order to define a LOTOS interpretation method for architectural designs. Before we present our intermediate model solution (LARG) for the generation of LOTOS expressions from architectural designs, we need to:

- analyse the required nature of the LARG model, and
- define a set of requirements which must be fulfilled by the LARG model.

As previously mentioned, the definition of the LARG must be governed by the two following statements:

- (1) the mapping of different architectural models into LARGs must be possible, and
- (2) the generation of LOTOS architectural expressions from LARGs must be possible.

We call statement (1) the "AM" statement, and statement (2) the "LAEG" statement. Now, we will analyse separately the requirements imposed by these two statements. The AM statement is concerned with the "universality" of the intermediate model, while the LAEG statement is concerned with the analysis and the linearization of the 2-dimensional representation of the architecture in order to generate LOTOS expressions.

5.2.1 AM Requirements

First, we analyse the requirements imposed by the AM statement. In this analysis, two different types of architectural artifacts have to be considered: the **architectural components** and the **component interactions**. The architectural requirements are the one that applied to the definition of the set of artifacts used in the intermediate model.

Since the intermediate model constitutes an intermediate stage between architectural models and LOTOS architectural expressions, it seems natural to define the set of intermediate model artifacts as being the set of LOTOS architectural artifacts.

Therefore, in relation with architectural components, since LOTOS only has one type of component (process), we must define the intermediate model such that it possesses only one type of component. This type of component must be sufficiently abstract to allow the mapping of different types of architectural components.

In relation with component interactions, we need to define an intermediate model which possesses every possible type of interactions that can be represented in LOTOS architectural expressions. These are:

- N-way interaction, and
- high level operators, i.e. parallel composition operators (either general or restricted), which include the interaction operator, the enable operator and the disable operator.

5.2.2 LAEG Requirements

We now analyse the requirements imposed by the LAEG statement. In the analysis of these requirements, we have to consider two different features that must be provided by the intermediate model:

- architecture analysis, and
- graph linearization facilities.

5.2.2.1 Architecture Analysis

In order to generate a LOTOS architectural expression from a given LARG, the second step consists in analyzing the LARG itself. This stage is essential for the generation of expressions for two main reasons.

- It provides ways of identifying errors in the LARG, and in this way this stage constitutes the first level of validation in the design process. Design errors in LARGs are discussed in §7.1.
- It permits to determine if the LARG is ambiguous or not, and if yes, which type of ambiguity it is. The different types of ambiguities are discussed in section §7.2. The identification of ambiguities is essential, because it constitutes an important parameter in the generation of algebraic expressions.

5.2.2 2 Graph Linearization

Then, the last step of the generation of LOTOS architectural expressions consists in the linearization of the 2-dimensional LARG (see figure 5.4). This stage is essential since LOTOS only possesses binary operators. Thus, the LARG model needs to have "graph rewriting" facilities for LARG linearization. An efficient way of doing the graph linearization consists in grouping architectural components in a binary manner (see definition of binary grouped LARGs in §6.1.7). This type of groupings may also be represented by means of binary tree representations of the LARGs. The binary grouping process must be defined in such a way that the interaction semantics of the design is preserved through LARG groupings. Also, the binary trees can be written as LOTOS architectural expressions.

An illustration of graph linearization by grouping is given in figure 5.4. In figure 5.4a, we have a LARG on which interaction gates synthesis and architecture analysis have been applied. Therefore, we consider that there is no architectural error in the LARG of figure 5.4a, according to §7.1. Then, figure 5.4b gives an equivalent LARG which has been linearized by successive groupings. The grouping sequence used in figure 5.4 has been arbitrarily chosen, and is only one of many possible ones. The visual notations used in this example are the one of the LARG model; they are described in §6.

Figure 5.4 Linearization by grouping

The tree representation of the LARG of figure 5.4b and its associated LOTOS architectural expression are given in figure 5.5. We see from these two figures that the generation of a LOTOS architectural expression from a linearized LARG is straightforward.

((P1[a, e] ||a|| P2[a, b]) ||b|| P3[b, c]) ||c, e|| (P4[c, d] ||d|| P5[d, e])

Figure 5.5 Tree representation and LOTOS architectural expression of the linearized LARG

5.2.3 Summary

In summary, the LARG must:

- possess abstract components on which different types of architectural components can be mapped,
- allow the representation of different type of component interactions , e.g. N-way interaction,

- possess high level operators (interaction operator),
- allow architecture analysis,
- enable the linearization of the architectural design representation.

The intermediate model used in our interpretation method is called the LARG model. LARG have been developed specifically for the purpose of this project. The model is described and discussed in the next chapter (§6).

Chapter 6 The LARG Model

In this chapter, we present the LARG (LOTOS Architectural Representation Graph) model. The LARG model has been developed, within the context of this research project, to serve as the intermediate model in the LOTOS architectural interpretation method. Its definition has been based on the set of requirements previously discussed in §5.2. In order to allow the interpretation of the greatest possible number of description models, the LARG model has been developed to be as abstract as possible. Thus, the LARG model possesses only one type of architectural component, called process, and one type of interaction which is rendez-vous interaction. Interactions between processes are realized by means of synchronization on gates. The LARG model allows the representation of N-way interactions (for $N \geq 1$) and possesses a high level interaction operator that corresponds to the LOTOS parallel operator. Also, because the generation of LOTOS architectural expressions must be allowed from LARGs, the LARG model has been developed in such a way that the LARG artifacts, i.e. processes and gates, can directly be mapped to LOTOS architectural constructs, i.e. LOTOS processes and LOTOS gates. Finally, for the purpose of LAEG, both a Grouping algorithm and an UnGrouping algorithm have been defined on LARGs.

This chapter is divided in two sections: in §6.1 we define LARG graphical notations, and in §6.2 we present the formal description of LARGs; the Grouping and the UnGrouping algorithms are defined in §6.2.4 and §6.2.5.

6.1 LARG Visual Notations

In this section, we describe the different types of LARG visual notation and give some of their semantic conventions.

6.1.1 LARG Components

Processes and Gates

Processes are visually represented by rectangles containing an identifier (e.g. P, P1, P2, ...) which uniquely identifies processes, called the **process identifier**. In LARG, interaction gates are simply called **gates** and they represent interaction points on which processes may interact. Then, a set of gates called **label set** is added at the top of process boxes (see figure 5.9) in a rounded rectangle. The label set of a process P_i contains the all the gates on which P_i may interact. Figure 6.1 illustrates an example of a simple LARG.

Figure 6.1 Example of a simple LARG

Gate Sets and Edges

Gate sets (GSs) correspond to parallel operators in LOTOS. They are visually represented by circles in which a set of gates appears (figure 6.2a) (select synchronization in LOTOS). We may also use an empty circle to express concurrency (figure 6.2b) (interleave operator in LOTOS). For any subset of N processes ($N \geq 2$) in the LARG, there exist a unique GS (possibly empty) which contains all the gates on which the different processes of the subset may interact together in an N -way interaction. Edges are used to link processes to GSs.

Figure 6.2 Interaction notations

N-Way Interactions

In LARG, the number of edges linked to a particular GS indicates the number of processes which participate in interaction on the gates contained in the GS. Thus, if there are N edges linked to an GS G_i , then all the gates contained in G_i are considered to be in an N -way interactions between the processes which are linked to G_i . For example, in figure 6.3, gate a is a 3-way interaction between $P1$, $P2$ and $P3$, and gates b , c and d are 2-way interaction.

Figure 6.3 Illustration of 2-way and 3-way interactions in LARG

Grouping

In LARG, dotted rectangles can be used to group processes, as parenthesis in algebraic expressions (figure 6.4b). We call **Groupings** the subsystems delimited by dotted rectangles. The semantics of GSs linked to groupings is the following: if a gate g is contained in an N -way interaction GS linked to a grouping in which g is an M -way interaction gate, then g is an $(N+M-1)$ -way interaction gate between the N processes linked to the GS containing g outside the grouping and the M processes linked to the GS containing g inside the grouping. For example in figure 6.4b, gate a is in the same LARG a 2-way interaction gate between $P1$ and the grouping, and 2-way interaction gate between $P2$ and $P3$ inside the grouping. Then, the semantics of GSs linked to groupings implies that a is a 3-way interaction gate between $P1$, $P2$ and $P3$. This can be verified by looking at the equivalent ungrouped LARG (figure 6.4a) where a is a 3-way interaction gate between $P1$, $P2$ and $P3$.

Definition: Binary Grouped LARG

We say that a given LARG is **binary grouped** iff all the processes of the LARG are grouped in such a way that every process, or grouping, is linked to one and only one GS. The relation between binary grouped LARGs and LOTOS expressions is one-to-one. For example, the LARG presented in figure 6.4b is binary grouped, but the one presented in figure 6.4a is not. The LOTOS expression corresponding to figure 6.4b is:

$P1[a, b, c] \parallel [a, b, c] (P2[a, b, d] \parallel [a, d] P3[a, c, d])$.

Figure 6.4 Example of grouping in LARG

Process Architectural Description

We illustrate the fact that a given LARG G corresponds to the architectural description of a process P by enclosing G in a process architectural description box. An example of **process architectural description box** is given in figure 6.5. The box is composed of two parts separated by a full line: the top part in which the process identifier (e.g. P in figure 6.5) and the label set (e.g. $\{a, b, c, d\}$ in figure 6.5) appears respectively on the left and the right hand sides, and the bottom part which is composed of both a hidden gate box, which optionally appears at the top left hand side corner under the process identifier (e.g. **hide a** in figure 6.5), and a LARG which corresponds to the architectural description of the process. Hidden gates in LARGs corresponds to LOTOS hidden gates; they constitute internal gates that are not offer to the environment of the process in which they

are defined. If there is no hidden gates in the architectural description, then the hidden gate box does not appear in the process architectural description box.

Figure 6.5 Process architectural description box

6.1.2 LARG Semantic Conventions

Gates interacting exclusively with the environment

In LARG, if a gate g is contained in the label set a process P_i , which is a process of P (i.e. $P_i \in P$), and P_i does not interact with any other process of P on gate g , then g is considered to be a gate on which P_i interacts exclusively with the environment of P , e.g. gate c in figure 6.1. In figure 6.1, P_1 has only one gate a on which it interacts with P_2 , but P_2 has, in addition to gate a , a gate c on which P_2 interacts exclusively with the environment.

However, note that, as in LOTOS, every non-hidden gate also requires an interaction with the environment. This means that every n -way interaction on a non-hidden gate g in a process P is in fact a $(n+m)$ -way interaction gate, where m is the number of interacting processes on g outside of P . If P does not interact with any other process, then g is a $(n+1)$ -way interaction gate between the n processes which interact on g inside of P and the environment. This distinction between gates on which processes interact exclusively with the environment and other ones is done for the purpose of LARG grouping (see definition of deterministic LARG §7.2.1)

Interleave Components

Also, we use the convention that if there is no common GS between any subset of processes, then these processes are considered to be executing in parallel (interleave operator in LOTOS). For example in figure 6.6, we consider that P1 and P3 are in parallel. Thus the result of the interpretation of figure 6.6 can be expressed by the LOTOS expression $P2[b, d] \parallel [b, d] (P1[b] \parallel P3[d])$. This expression is obtained by isolating P2 from P1 and P3 (i.e. by grouping P1 and P3).

Figure 6.6 Interleaved processes

Important Remark

We want to clarify a case that can be wrongly interpreted by some readers, especially those who are familiar with LOTOS-oriented representations, like GLOTOS. In figure 6.7, figure 6.7a should not be interpreted as a 3-way interaction between P1, P2 and P3 as it is illustrated in figure 6.7b, but it should rather be interpreted as a 2-way non-deterministic interaction-choice (see non-deterministic interaction-choice in §4.2.2) between P1 and either P2 or P3. It is important to remember that, as expressed by the interleave components convention, P2 and P3 are in parallel (interleave), and thus they do not interact together (figure 6.7c). Therefore, figure 6.7a must be interpreted as illustrated by the binary grouped LARG of figure 6.7d. In relation with LOTOS, figure 6.7a must be expressed by the following expression:

$P1[a] \parallel [a] (P2[a] \parallel P3[a])$.

Figure 6.7 2-way non-deterministic interaction-choice vs. 3-way interaction

Case Study: The Elevator System

The Elevator System LARG

In figures CS6.1, CS6.2 and CS6.3, a LARG version of the elevator system MachineCharts design (figure CS2.2) is given. The architectural mapping of the elevator system: MachineCharts design on LARGs is discussed in the last part of this "Case Study" box.

Figure CS6.1 The LARG representation of the elevator system MachineCharts design

In the elevator system LARG of figure CS6.1, the Building and the Car processes (Car 1, Car 2, and Car 3) correspond to primitive components and thus they are described by means of behavioral description. These behavioral descriptions are not given here. On the other hand, the Coordination process and the Car_SC processes are architecturally described in figures CS6.2 and CS6.3.

Also, we observe in figures CS6.2 and CS6.3 that the objects (Pending_Floor_Requests, All_Car_Status, Pending_Car_Requests and Car_Controller) that appear in the elevator system MachineCharts design do not appear in the elevator system LARGs. The reason for this is that MachineCharts objects may be represented in LOTOS using abstract data types (ACT ONE), and thus they do not appear in the LARGs which is only concerned with process interactions.

Figure CS6.2 The Car_SC subsystem LARG representation

Figure CS6.3 The Coordination subsystem LARG representation

In the LARGs of figure CS6.2 and CS6.3, the following processes correspond to primitive components: Report_cb, Rec_msg, Mailbox (Control), Engine (Control), Report_fb, Change_cs, Mailbox (Dispatch) and Engine (Dispatch). These are described by means of a LOTOS specification; the LOTOS specifications of primitive processes is not provided in this thesis.

Architectural Mapping

Architectural component mapping

From the elevator system LARG, we observe that the different architectural components of the MachineCharts design have all been mapped to processes. The Building and Coordination subsystems contained in the MachineCharts design have both been mapped into a single LARG process, while the set of Car subsystems and the set Car_SC subsystems have both been mapped into a group (LARG grouping) of LARG processes. For the purpose of this example the number of cars in the elevator system, and consequently in the Car and Car_SC subsystems, has been set to three. However, the same reasoning apply for different number of cars.

Component interaction mapping

In relation with component interactions, we observe that each MachineCharts wire is mapped to a gate in LARGs. However, while the fb, cs and msg wires have been mapped each to a single gate in the LARG, the cb wire has been mapped to a set of gates. In fact, we could have mapped the cb wire to a single gate and then constrained process interactions using extended gate identifier (e.g. cb!id) in such a way that only interaction between relevant processes (i.e. Car i_SC and Car i) are allowed. However, since we know, from the implicit semantics of the MachineCharts design, that for a given Car (tactical control) (e.g. Car 1) the only allowed interactions on the cb wire are the ones with its peer strategic control subsystem (Car 1_SC), we have chosen to express this fact clearly in the LARG using different gate names for interactions between different pairs of interacting processes. Thus, we use cb1 as the interaction gate between Car 1_SC and Car 1, and more generally cbi as the interaction gate between Car i_SC and Car i.

Architectural artifact implied behavior effects on process behavior

We identify two different types of effect that are created by architectural artifact implied behavior on process behavior: internal effect and external effect.

- **Internal effect**

In MachineCharts, the implied behavior of architectural artifacts have a direct effect on the artifact's behavior. This is what we call internal effect. For example in the elevator system MachineCharts design (figure CS2.2) the behavior of the mailbox component is partially determined by the implied behavior of the mailbox artifact. In this case, the behavior of the mailbox component is a composition of both the implied behavior of the mailbox artifact and the behavioral description of the mailbox.

In a general manner, we can say that, in any architectural interpretation method, the behavior of an architectural artifact is the result of the composition of both the implied behavior of the artifact and the behavioral description (possibly empty) of the artifact. Also, the implied behavior of different artifacts may have more or less effect on their own behavior depending on the type of artifact (expressive ones or abstract ones) used in the description model. In MachineCharts, since the artifacts used are very expressive ones, the architectural artifact implied behavior effect is considered to be important.

- **External effect**

In MachineCharts, the implied behavior of architectural artifacts may also have an effect on the behavior of an another component with which it interacts. In other words, if two architectural MachineCharts components interact together via a given wire, then depending on the type of port used by the called component, the behavior of the calling component will differ. This is what we call external effect.

By definition, we say that in a given interaction, the called component is the one to which the communication port is attached, and the calling one is the one located at the other end of the wire.

For example, in the elevator system MachineCharts design (figure CS2.2), we observe that there exists two different types of interaction: wire connected to button and wire connected to mailbox. For the two different types of interaction (button and wire) we give, in table CS6.1 (the table is placed right after this case study box), the skeleton of both the MachineCharts behavioral description and the LOTOS specification of the calling component. We see from the comparative table (table CS6.1) that the MachineCharts behavioral description of the Calling_Component is the same in both cases, while the Calling_Component LOTOS specifications are different. This difference in the LOTOS specification can be explained by the different semantics of the two types of port: button and mailbox (see "The MachineCharts architectural model" part of the case study box of §2.3.1.1). We recall that:

- buttons are unary operators that provide procedural abstractions, and
- mailboxes are asynchronous interface operators.

The scenario implied by a button is the following:

- Calling_Component : call (possibly send data)
- Called_Component : execute (possibly received input data), end (possibly deliver output data)
- Calling_Component : continue (possibly received resulting data)

The scenario implied by a mailbox is the following:

- Calling_Component : send, continue
- Called_Component : receive, read, continue

These scenarios are reflected in the LOTOS specifications (table CS6.1). We see, from the LOTOS specification given in this table, that in the first case, i.e. msg wire linked to button, the Calling_Component LOTOS process has to interact twice on the msg gate before it can continue. The first time to call the button (and possibly send the button input data), and the second time to synchronize with the end of the button execution (and possibly received output data from the button). In between these two successive interactions on gate msg, the Calling_Component is blocked. On the other hand, in the mailbox case, the Calling_Component LOTOS process only interacts once with the mailbox port to deliver the message (possibly including data).

Tree Representation of the elevator system LARG

From the elevator system LARG given in figures CS6.1, CS6.2 and CS6.3, we may produce the tree representation given in figure CS6.4. In this figure, the vertical dotted lines (e.g. the one between Car1 and Car) means the process at the top of the line is an instantiation of the one at the bottom. Also, all leaves represent primitive processes, except the ones for which a tree have been further developed, e.g. the Car_SC process which has been architecturally decomposed into the tree given below the main tree.

Figure CS6.4 Tree representation of the elevator system LARG

Then, we can deduce the following skeleton for the LOTOS specification of the elevator system.

specification Elevator_System :=

AE_{Elevator_System};

where

process Building[fb] := BE_{Building} endproc

process Coordination[fb, cs, msg] := hide fb_rep, new_cs in AE_{Coordination};

where

process Dispatch[fb_rep, new_cs, msg] := hide int in AE_{Dispatch};

where

process Engine_Dispatch[int, msg] := BE_{Engine_Dispatch} endproc

process Mailbox_Dispatch[fb_rep, new_cs, int] := BE_{Mailbox_Dispatch} endproc

endproc

process Report_fb[fb, fb_rep] := BE_{Report_fb} endproc

process Change_cs[cs, new_cs] := BE_{Change_cs} endproc

endproc

process Car1_SC[msg, cb1] := Car_SC[msg, cb1] endproc

process Car2_SC[msg, cb2] := Car_SC[msg, cb2] endproc

process Car3_SC[msg, cb3] := Car_SC[msg, cb3] endproc

process Car_SC[msg, cb] := hide cb_rep, msg_rec in AE_{Car_SC};

where

process Control[cb_rep, msg_rec] := hide int in AE_{Control};

where

process Engine_Cont[int] := BE_{Engine_Cont} endproc

process Mailbox_Cont[cb_rep, msg_rec, int] := BE_{Mailbox_Cont} endproc

endproc

process Report_cb[cb, cb_rep] := BE_{Report_cb} endproc

process Rec_msg[msg, msg_rec] := BE_{Rec_msg} endproc

endproc

```
process Car1[cb1, cs] := Car[cb1, cs] endproc  
process Car2[cb2, cs] := Car[cb2, cs] endproc  
process Car3[cb3, cs] := Car[cb3, cs] endproc  
process Car[cb, cs] := BECar endproc  
endspec
```

Specification CS6.1 The skeleton of the elevator system LOTOS specification

Therefore, in order to generate a complete LOTOS specification for the elevator system, we have to produce the different LOTOS architectural and behavioral expressions (respectively AE's and BE's).

Type of interaction	MachineCharts behavioral description	LOTOS Behavior
msg wire linked to button	engine Calling_Comp is ...; push(msg); ...; end Calling_Comp;	process Calling_Comp := ...; msg!x; msg?y: Msg_Returned; ...; endproc
msg wire linked to mailbox	engine Calling_Comp is ...; push(msg); ...; end Calling_Comp;	process Calling_Comp := ...; msg!x; ...; endproc

Table CS6.1 External effects of architectural artifact implied behavior on LOTOS specifications (explanation in the external effect section in page 68-69)

6.2 Formal Description of LARG

We define a LARG P as a 5-tuple $\langle P, G, E, L, H \rangle$ where:

$P = \{P_1, P_2, \dots, P_n\}$ is the set of sub-processes of P ,

$G = \{G_1, G_2, \dots, G_m\}$ is the set of gate sets (GSs) inside of P ,

$E = \{e_1, e_2, \dots, e_k\}$ is the set of edges which link processes P_i to gate sets G_j . We referred as (P_i, G_j) the edge that links process P_i to GS G_j .

$E \subseteq P \times G$.

$L = \{g_1, g_2, \dots, g_n\}$ is the set of labels, or gates, which are accessible for interaction by the environment of P , and

$H = \{hg_1, hg_2, \dots, hg_n\}$ is the set of hidden gates, i.e. gates which are not accessible by the environment.

Also, we obtain:

- the set of processes of P by using $P(P)$
- the set of GS of P by using $G(P)$
- the set of edges of P by using $E(P)$
- the set of labels of P by using $L(P)$
- the set of hidden gates of P by using $H(P)$

When we write $P = \langle P, G, E, L, H \rangle$, we mean that P is a process identifier such that $LARG(P) = \langle P, G, E, L, H \rangle$.

L and H

In LARGs, every process is recursively defined as a LARG. All primitive components P_i 's are defined as $\langle \emptyset, \emptyset, \emptyset, L(P_i), H(P_i) \rangle$, i.e. $LARG(P_i) = \langle \emptyset, \emptyset, \emptyset, L(P_i), H(P_i) \rangle$. Thus, $L(P_i)$ has to be explicitly given in the description of all primitive components. $H(P_i)$ is by default set to $\emptyset$, but it can be updated by explicitly adding gates to $H(P_i)$. Then, for every non-primitive (architectural) component P_i , $L(P_i)$ is recursively defined as:

$$L(P_i) = \bigcup_{P_{ij} \in P(P_i)} L(P_{ij}) - H(P_i)$$

In this thesis, the operator $\bigcup_{S_i \in S} S_i$ is used as the generalized union operator over elements of a set, and it means $\bigcup S_i$ for all $S_i \in S$. Also, in cases where a process P_i has subprocesses, we denote them P_{ij} .

6.2.1 Functions on LARGs

We define the following functions on LARGs:

LARG

$LARG(P_i)$ defines, for every process P_i contained in $P(P)$, the 5-tuple which defined the internal composition of P_i , and

$$\forall P_i \in P(P), LARG(P_i) = \langle P(P_i), G(P_i), E(P_i), L(P_i), H(P_i) \rangle$$

GSets

$GSets(P_i)$ defines, for every process P_i contained in $P(P)$, the set of GS linked to P_i , and

$$\forall P_i \in P(P), GSets(P_i) = \{G_j / (P_i, G_j) \in E\}$$

Procs

$Procs(G_i)$ defines, for every GS G_i contained in $G(P)$, the set of processes linked to G_i , and

$$\forall G_i \in G(P), Procs(G_i) = \{P_j / (P_j, G_i) \in E\}$$

Int (Internal gates of P_i in P)

$Int(P_i)$ defines, for every P_i contained in $P(P)$ ($P_i \in P(P)$), the set of gates of P_i ($Int(P_i) \subseteq L(P_i)$) on which P_i interacts with other processes P_j in r . We call gates contained in such sets "internal gates of P_i in P ", and

$$\forall P_i \in P(P), Int(P_i) = \bigcup_{G_j \in GSets(P_i)} G_j$$

Env (Environmental gates of P_i in P)

$Env(P_i)$ defines, for every P_i contained in $P(P)$ ($P_i \in P(P)$), the set of gates of P_i ($Env(P_i) \subseteq L(P_i)$) on which P_i interacts with no other processes P_j in P ($i \neq j$), i.e. P_i interacts exclusively with the environment of P . We call gates contained in such sets "environmental gates of P_i in P ", and

$$\forall P_i \in P(P), Env(P_i) = L(P_i) - Int(P_i)$$

RealProcs

Also in LARGs, the process identifier Gr_i is used as a special process identifier for groupings. Gr_i 's are considered to be processes in the formal description of an LARG, i.e. they are defined as a 5-tuple $\langle P, G, E, L, H \rangle$, but they do not constitute real processes. The process identifier Gr_i is only used for the purpose of internal formal description of LARGs. Thus, we define the function *RealProcs*. This function returns the set of real processes contained in an LARG.

$$RealProcs(P) = P(P) - (\bigcup_{Gr_i \in P(P)} Gr_i) \cup (\bigcup_{Gr_i \in P(P)} RealProcs(Gr_i))$$

In the case where P is an ungrouped LARG,

$$RealProcs(P) = P(P).$$

Cardinality

Also, we define the cardinality function $|S|$ which gives the number of elements contained in the set S . For example, $|P(P_i)|$ gives the number of processes contained in P_i .

6.2.2 Example of LARG Formal Description

In figure 6.8, a 3-component LARG is illustrated. The formal description of this LARG is given by the 5-tuple:

$P = \langle P, G, E, L, H \rangle$, where

$P = \{P_1, P_2, P_3\}$,

$G = \{G_1, G_2, G_3\}$,

$E = \{(P_1, G_1), (P_1, G_3), (P_2, G_1), (P_2, G_2), (P_2, G_3), (P_3, G_2), (P_3, G_3)\}$,

$L = \{a, b, c, d\}$, and

$H = \emptyset$

where

$G_1 = \{b\}$,

$G_2 = \{d\}$, and

$G_3 = \{c\}$

Figure 6.8 LARG example

Also, the computation of the different functions defined on LARGs give the following results:

LARG

$LARG(P_1) = \langle \emptyset, \emptyset, \emptyset, \{a, b, c\}, \emptyset \rangle$,

$LARG(P_2) = \langle \emptyset, \emptyset, \emptyset, \{b, c, d\}, \emptyset \rangle$,

$LARG(P_3) = \langle \emptyset, \emptyset, \emptyset, \{c, d\}, \emptyset \rangle$.

GSets

$GSets(P_1) = \{G_1, G_3\}$

$$GSets(P_2) = \{G_1, G_2, G_3\}$$
$$GSets(P_1) = \{G_2, G_3\}$$
Procs
$$Procs(G_1) = \{P_1, P_2\}$$
$$Procs(G_2) = \{P_2, P_3\}$$
$$Procs(G_3) = \{P_1, P_2, P_3\}$$
Int
$$Int(P_1) = \{b, c\}$$
$$Int(P_2) = \{b, c, d\}$$
$$Int(P_3) = \{c, d\}$$
Env
$$Env(P_1) = \{a\}$$
$$Env(P_2) = \emptyset$$
$$Env(P_3) = \emptyset$$
RealProcs
$$RealProcs(P) = P(P) = \{P_1, P_2, P_3\}$$

Case Study: The Elevator System

Formal Description of the elevator system LARG

The formal description of the elevator system LARG illustrated in figures CS6.1, CS6.2 and CS6.3 is given as follows. Every component which is not defined by means of a LARG here is considered to be a primitive component, and therefore is described by means of a behavioral description.

The_Elevator_System =

$\langle \{ \text{Coordination, GrCar_SC, GrCar, Building} \}, \{ G_1, G_2, G_3, G_4 \},$
 $\{ (\text{Coordination, } G_1), (\text{Coordination, } G_2), (\text{Coordination, } G_3), (\text{Gr1, } G_2), (\text{Gr1, } G_4),$
 $(\text{Gr2, } G_3), (\text{Gr2, } G_4), (\text{Building, } G_1) \}, \{ \text{fb, cs, msg, cb1, cb2, cb3} \}, \emptyset \rangle$

where

$G_1 = \{ \text{fb} \}$

$G_2 = \{ \text{cs} \}$

$G_3 = \{ \text{msg} \}$

$G_4 = \{ \text{cb1, cb2, cb3} \}$

$\text{Gr1} = \langle \{ \text{Car1, Car2, Car3} \}, \emptyset, \emptyset, \{ \text{cs, cb1, cb2, cb3} \}, \emptyset \rangle,$

$\text{Gr2} = \langle \{ \text{Car1_SC, Car2_SC, Car3_SC} \}, \emptyset, \emptyset, \{ \text{msg, cb1, cb2, cb3} \}, \emptyset \rangle,$

$\text{Car_SC} = \langle \{ \text{Control, Report_cb, Rec_msg} \}, \{ G_1, G_2 \}, \{ (\text{Control, } G_1), (\text{Control, } G_2),$
 $(\text{Report_cb, } G_1), (\text{Rec_msg, } G_2) \}, \{ \text{msg, cb} \}, \{ \text{cb_rep, msg_rec} \} \rangle$

where

$G_1 = \{ \text{cb_rep} \},$

$G_2 = \{ \text{msg_rec} \}$

$\text{Control} = \langle \{ \text{Mailbox, Engine} \}, \{ G_1 \}, \{ (\text{Mailbox, } G_1), (\text{Engine, } G_1) \}, \{ \text{msg_rec, cb_rep} \}, \{ \text{int} \} \rangle$

where

$G_1 = \{ \text{int} \}$

$\text{Coordination} = \langle \{ \text{Dispatch, Report_fb, Change_cs} \}, \{ G_1, G_2 \}, \{ (\text{Dispatch, } G_1),$
 $(\text{Dispatch, } G_2), (\text{Report_fb, } G_1), (\text{Change_cs, } G_2) \}, \{ \text{fb, cs, msg} \}, \{ \text{fb_rep, new_cs} \} \rangle$

where

$G_1 = \{ \text{fb_rep} \}$

$G_2 = \{ \text{new_cs} \}$

Dispatch = $\langle \{ \text{Mailbox, Engine} \}, \{ G_1 \}, \{ (\text{Mailbox}, G_1), (\text{Engine}, G_1) \}, \{ \text{msg. new_cs, fb_rep} \}, \{ \text{int} \} \rangle$
 where
 $G_1 = \{ \text{int} \}$

6.2.3 Partition

In this thesis, the partition function Part is defined in the conventional way as follows:

$$\text{Part}(P(P)) = \{ (\text{Part}_1, \text{Part}_2, \dots, \text{Part}_n) / \text{Part}_i \subseteq P(P) \wedge \forall i, j (i \neq j) \text{Part}_i \cap \text{Part}_j = \emptyset \wedge \bigcup_{i=1}^n \text{Part}_i = P(P) \}.$$

6.2.4 Grouping

Formally, the grouping function on LARGs is defined as follows:

Group : LARG $\times$ Part $\rightarrow$ LARG

Group (P, Part) = P'

where P and P' are two LARGs and Part is a partition of P(P), i.e.

Part $\in$ Part(P(P)).

The grouping algorithm defines P' from P and Part in a constructive way. In order to construct P' from P, we first need to sort the elements Part_i of Part such that the elements of Part which contains more than one process will be distinguished from the ones which contain only one process. All processes contained in an element Part_i $\in$ Part such that |Part_i| = 1 are considered as processes in P'. Then, for every Part_i $\in$ Part such that |Part_i| > 1, we construct a LARG Gr_i, and then we consider Gr_i as a process in P' (i.e. Gr_i $\in$ P(P')). The LARG of Gr_i is constructed from the processes of Part_i, i.e. P(Gr_i) = Part_i. The construction of the communication links within Gr_i, i.e. GSs and edges which connect the processes of Gr_i, must preserve the interaction semantic of the former LARG P in which the processes of Gr_i are involved. This means that if G_j is a n-way interaction GS in P (i.e. G_j $\in$ G(P) and |Procs(G_j)| = n) to which m (m $\leq$ n) processes of P(Gr_i) are connected (i.e. P(Gr_i) $\subseteq$ Procs(G_j) and |P(Gr_i)| = m), then all gates contained in G_j must be contained in

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

the m -way interaction GS between the m processes of $P(Gr_i)$ connected to G_j in P . The set of all other processes of P' (i.e. $P_j \in P(P')$ for all j , and $Gr_k \in P(P')$ for all $k \neq i$) constitutes the environment of Gr_i and may interact with Gr_i on any gate $g \in L(Gr_i)$. Then, the last phase of the algorithm consists of constructing P' from $P(P')$ which has been defined in the first phase. As in the previous phase, we must preserve the interaction semantics of P while constructing P' .

Also, the partition $Part$ such that $|Part| = 1$, i.e. the case where all processes are grouped together in a unique grouping, constitutes a special partition which leads to a special grouping. In this case,

$$LARG(P') = \langle \{Gr_1\}, \emptyset, \emptyset, L(P), H(P) \rangle, \text{ and} \\ LARG(Gr_1) = \langle P(P), G(P), E(P), L(P), \emptyset \rangle.$$

This special partition is checked at the very beginning of the grouping algorithm. If this case occurs, P' is defined in the straightforward manner previously described and the algorithm terminates.

Also, for the purpose of the application of the algorithm, all groupings present in P (i.e. all $Gr_i \in P(P)$) are considered to be processes in the application of the grouping algorithm to construct P' from P . In fact, all $Gr_i \in P(P)$ can be relabelled as P_{n+i} , i.e. $P(P) = \{P_1, P_2, \dots, P_n, G_1, \dots, G_m\}$ can be transformed into $P(P) = \{P_1, P_2, \dots, P_n, P_{n+1}, \dots, P_{n+m}\}$ for the purpose of the application of the algorithm. Thus, we do not consider the internal composition of $Gr_i \in P(P)$, i.e. $LARG(Gr_i)$, in the application of the grouping algorithm.

6.2.4.1 The Grouping Algorithm

begin

$LARG(P') = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle;$ {initialization of P' }

if $|Part| = 1$ **then begin**

$LARG(Gr_1) = \langle P(P), G(P), E(P), L(P), \emptyset \rangle;$

$LARG(P') = \langle \{Gr_1\}, \emptyset, \emptyset, L(P), H(P) \rangle;$

end

else begin

{Phase 1:

We construct the set $P(P')$, and for each $Part_i$ such that $|Part_i| > 1$, we construct the set $P(Gr_i)$. As previously mentioned, if a partition $Part_i$ contained more than one

process, a grouping identifier Gr_i will be place in the set $P(P')$ and an LARG Gr_i will be defined in the second phase. Then, the set $P(Gr_i) = Part_i$

for each $Part_i \in Part$ **do**

if $|Part_i| = 1$ **then**

$P(P') := P(P') \cup Part_i$

else $\{|Part_i| > 1\}$ **begin**

$P(P') := P(P') \cup Gr_i$;

$P(Gr_i) := Part_i$;

end;

{Phase 2:

We construct, for each $Gr_i \in P(P')$, the LARG of Gr_i . We recall that we already obtained $P(Gr_i)$ in the previous phase. }

for each subset S of $P(Gr_i)$ **do begin**

if $|S| > 1$ **then**

for each $G_j \in G(P)$ **do**

if $Procs(G_j) \cap P(Gr_i) = S$ **then**

$G_S := G_S \cup G_j$;

if $G_S \neq \emptyset$ **then begin** $\{G_S$ is the ISI-way interaction GS between the processes of $S\}$

$G(Gr_i) := G(Gr_i) \cup G_S$;

for all $P_k \in S$ **do**

$E(Gr_i) := E(Gr_i) \cup (P_k, G_S)$;

end;

end;

{Phase 3:

We now construct the complete LARG of P' . We recall that we already obtained $P(P')$ in the first phase of the algorithm. }

for each subset S of $P(P')$ **do begin**

if $|S| > 1$ **do**

for each $G_j \in G(P)$ **do**

if $Procs(G_j) \sqsubseteq P(P') = S$ **do** {see definition of $\sqsubseteq$ below (§6.2.4.2)}

$G_S := G_S \cup G_j$;

if $G_S \neq \emptyset$ **then** $\{G_S$ is the ISI-way interaction GS between the processes of S , which in this case are partition (or LARG)) **begin**

```

     $G(P') := G(P') \cup G_S;$ 
    for all  $P_k \in S$  do
         $E(P') := E(P') \cup (P_k, G_S);$ 
    end;
end;
end;
end.

```

6.2.4.2 Definition of generalized intersection operator $\triangle$:

For all $P_i \in P(P)$

$P_i \in (Procs(G_j) \triangle P(P))$ iff $P_i \in (Procs(G_j))$

for all $Gr_j \in P(P)$

$Gr_j \in (Procs(G_j) \triangle P(P))$ iff $P(Gr_j) \cap Procs(G_j) \neq \emptyset$

6.2.4.3 Example of Application of the Grouping Algorithm

An example of application of the grouping algorithm is given in figure 6.9. In this figure, two LARGs, P (figure 6.9a) and P' (figure 6.9b), are given. P is an ungrouped LARG, while P' is an equivalent grouped version of P . P' is obtained from P by applying the grouping algorithm with the partition:

$Part = \{ \{P_1\}, \{P_2, P_3, P_4\}, \{P_5, P_6\} \}.$

Figure 6.9 Application of the grouping algorithm

The formal descriptions of P and P' are (we do not give the formal description of the primitive processes (components) of P ($P_1, P_2, \dots, P_6$)):

$$P = \langle P, G, E, L, H \rangle$$

$$P = \{P_1, P_2, P_3, P_4, P_5, P_6\},$$

$$G = \{G_1, G_2, G_3, G_4, G_5, G_6, G_7\},$$

$$E = \{(P_1, G_1), (P_1, G_2), (P_2, G_1), (P_2, G_3), (P_2, G_4), (P_3, G_2), (P_3, G_3), (P_3, G_5), \\ (P_3, G_6), (P_4, G_4), (P_4, G_5), (P_5, G_6), (P_5, G_7), (P_6, G_6), (P_6, G_7)\},$$

$$L = \{a, b, c, d, e, f, g\}, \text{ and}$$

$$H = \emptyset$$

$$P' = \langle P', G', E', L', H' \rangle$$

$$P' = \{P_1, Gr_1, Gr_2\},$$

$$G' = \{G_1, G_2\},$$

$$E' = \{(P_1, G_1), (Gr_1, G_1), (Gr_1, G_2), (Gr_2, G_2)\},$$

$$L' = \{a, b, c, d, e, f, g\}, \text{ and}$$

$$H' = \emptyset$$

$$Gr_1 = \langle P(Gr_1), G(Gr_1), E(Gr_1), L(Gr_1), H(Gr_1) \rangle$$

$$P(Gr_1) = \{P_2, P_3, P_4\}.$$

$$G(Gr_1) = \{G_{11}, G_{12}, G_{13}\}.$$

$$E(Gr_1) = \{(P_2, G_{11}), (P_2, G_{12}), (P_3, G_{11}), (P_3, G_{13}), (P_4, G_{12}), (P_4, G_{13})\}.$$

$$L(Gr_1) = \{a, b, c, d, e, f\}, \text{ and}$$

$$H(Gr_1) = \emptyset$$

$$Gr_2 = \langle P(Gr_2), G(Gr_2), E(Gr_2), L(Gr_2), H(Gr_2) \rangle$$

$$P(Gr_2) = \{P_5, P_6\},$$

$$G(Gr_2) = \{G_{21}\},$$

$$E(Gr_2) = \{(P_5, G_{21}), (P_6, G_{21})\},$$

$$L(Gr_2) = \{f, g\}, \text{ and}$$

$$H(Gr_2) = \emptyset$$

Also,

$$RealProcs(P') = RealProcs(P) = P(P) = \{P_1, P_2, P_3, P_4, P_5, P_6\}$$

Case Study: The Elevator System

Application the Grouping Algorithm

Grouping may serve for example to group all strategic control entities together in the elevator system (ES) LARG. Therefore, we can transform the LARG ES of figure CS6.1 into the LARG ES' of figure CS6.6 using the following grouping:

$$Group(ES, (\{Building\}, \{Gr_1\}, \{Coordination, Gr_2\})) .$$

6.2.5 UnGrouping

Formally, the ungrouping function on LARGs is defined as follows:

$\text{UnGroup} : \text{LARG} \times \text{Grouping} \rightarrow \text{LARG}$

$\text{UnGroup} (P, Gr_i) = P'$

where P and P' are two LARGs and Gr_i is a grouping in P (i.e. $Gr_i \in P(P)$).

The ungrouping algorithm defines P' from P and Gr_i in a constructive way. The algorithm first defines $P(P')$ from $P(P)$ by replacing Gr_i by $P(Gr_i)$ in $P(P')$, i.e. $P(P') = P(P) - Gr_i \cup P(Gr_i)$. It then transforms every GS of P which is linked to Gr_i (i.e. every $G_j \in P(P)$ such that $Gr_i \in Procs(G_j)$), and every internal GS of Gr_i (i.e. every $G_j \in P(Gr_i)$) in the following way: if a gate g is an N -way interaction gate on which Gr_i interacts in P and an M -way interaction gate inside of Gr_i , then g becomes an $(N+M-1)$ -way interaction gate in P' and all processes which were linked to g , either outside or inside of Gr_i , in P are linked to the GS containing g in P' . All other GSs and edges, the ones not linked to Gr_i , remain unchanged in P' .

In prior to the application of the algorithm previously described, we check the number of processes contained in $P(Gr_i)$. We distinguish two cases: $|P(Gr_i)| = 1$ and $|P(Gr_i)| > 1$. If the number of processes is equal to one, the ungrouping algorithm performs like the identity and we defined P' such that $LARG(P') = LARG(P)$. However, if the number of processes is superior than one, the algorithm is applied and P' is defined.

6.2.5.1 The UnGrouping Algorithm

begin

If $|P(Gr_i)| = 1$ **then begin**

$P(P') = P(P) - \{Gr_i\} \cup P(Gr_i);$

$G(P') = G(P);$

$E(P') = E(P);$

for all $G_j \in GSets(Gr_i)$ **do**

$E(P') = E(P') - \{(Gr_i, G_j)\} \cup \{(P(Gr_i), G_j)\};$

$L(P') = L(P);$

$H(P') = H(P);$

else if $|P(Gr_i)| = |RealProcs(P)|$ **then begin**

$LARG(P') := LARG(Gr_i);$

$H(P') = H(P);$

end

else $\{|P(Gr_i)| \neq 1 \text{ and } |P(Gr_i)| \neq |RealProcs(P)|\}$ **begin**

 {We define the set of processes $P(P')$.}

$P(P') := P(P) - \{Gr_i\} \cup P(Gr_i);$

```

{ We define the new GSs of P' }
for each pair (Gi, Gj) such that ((Gi ∈ G(Gri)) or (Gi ∈ Env(Pi) for all Pi ∈
Gri)) and (Gj ∈ G(P)) and Gri ∈ Procs(Gj) { Gj is connected to Gri } do begin
  Gs := Gi ∩ Gj;
  if Gs ≠ ∅ then begin
    { we add Gs to G(P') }
    G(P') := G(P') ∪ Gs;
    { we define the new linkage to Gs }
    for all Pk ∈ (Procs(Gi) ∪ Procs(Gj)) do
      E(P') := E(P') ∪ (Pk, Gs);
    end;
  end;
end;
{ We finally rewrite the unchanged GSs and edges from P to P' }
for each Gi ∈ G(P) such that Gri ∉ Procs(Gj) do begin
  G(P') := G(P') ∪ Gi;
  for all Pk ∈ Procs(Gi) do
    E(P') := E(P') ∪ (Pk, Gi);
  end;
  L(P') := L(P);
  H(P') := H(P);
end;
end.

```

6.2.5.2 Example of Application of the UnGrouping Algorithm

In figure 6.10, an example of application of the UnGrouping algorithm is given. In figure 6.10a, we have the grouped LARG of figure 6.9b. Figure 6.10b gives the LARG P' resulting from the application of the function UnGroup(P, Gr₂).

Figure 6.10 Application of the ungrouping algorithm

The formal description of P'' is given by the following tuple:

$$P'' = \langle P'', G'', E'', L'', H'' \rangle$$

$$P'' = \{P_1, Gr_1, P_5, P_6\},$$

$$G'' = \{G_1, G_2, G_3\},$$

$$E'' = \{(P_1, G_1), (Gr_1, G_1), (Gr_1, G_2), (P_5, G_2), (P_5, G_3), (P_6, G_2), (P_6, G_3)\},$$

$$L'' = \{a, b, c, d, e, f, g\}, \text{ and}$$

$$H'' = \emptyset.$$

The formal description of P' and Gr_2 are given in §6.2.4.3. We observe that $P(P'')$ contains the same processes as $P(P')$ with the exception that Gr_2 has been replaced by P_5 and P_6 which correspond to $P(Gr_2)$. Also, the only GS that has been modified from P' to P'' is the one linked to Gr_2 in P' , i.e. G_2 of P' . It has been replaced by both G_2 and G_3 in P'' . Finally, we observe the newly created GSs have been linked to processes and that the linkage in P'' respects the interaction semantic of P' ; for example, we observe that gate f is in P' a 3-way interaction which involves Gr_1 and both processes of Gr_2 (i.e. P_5 and P_6) and that gate f is still a 3-way interaction between the same processes in P'' .

Also, we may verify that P'' could have been obtained directly from P by applying $Group(P, Part)$ by choosing $Part = \{\{P_1\}, \{P_2, P_3, P_4\}, \{P_5\}, \{P_6\}\}$. Therefore, we can prove that the application of the Grouping and UnGrouping algorithms provides correctness

preserving transformations (CPTs) on LARG P. However, some partitions may violate the interaction semantic of an LARG. These may occur if the LARG is non-deterministic and they will be discussed in the next chapter.

Case Study: The Elevator System

Application the UnGrouping Algorithm

The UnGrouping algorithm may be used to flatten the LARG ES given in figure CS6.1. To obtain the totally ungrouped version of the LARG ES, we need to apply the UnGrouping algorithm twice to ES as follows:

$\text{UnGroup}(\text{UnGroup}(\text{ES}, \text{Gr1}), \text{Gr2}).$

The result of the two successive applications of the UnGrouping algorithm is given in figure CS6.7. We observe that the application of the UnGrouping algorithm preserves the interaction semantics of LARGs.

Figure CS6.7 The ungrouped version of the elevator system LARG

6.3 Discussion

LARG constitutes a type of graph in which all possible types of interactions are represented by distinct GSs. For example, figure 6.11 gives the LARG for a 3-process architecture (figure 6.11a) and a 4-process architecture (figure 6.11b). We say that LARGs are interaction complete in opposition with complete graphs in such that they do not only illustrate interactions between any pair of processes, as it is the case in complete graphs, but they illustrate any types of interactions between any N-tuples of processes by distinct GS in

the graph. Thus, LARGs provide direct representations of N-way interactions ($N \geq 2$) in the graphs. For example, LARGs possess distinct GSs for the 3-way interaction involving P_i , P_j and P_k , and the 2-way interaction involving P_i and P_j . Every processes participating in a particular interaction is linked to the GS corresponding to this interaction. We think that the AR of any distributed system can be mapped into a LARG.

Figure 6.11 General 3-component and 4-component LARGs

In LARGs, the maximum number of GSs is given by the number of process subsets that contains more than one process; i.e.

$$\text{the number of GS in a LARG} \leq 2^n - n - 1$$

We observe that this number grows exponentially (2^n). However, in practice, the fact that empty GSs are usually not drawn in LARGs, and that they are not considered in their formal description, reduces the number of GSs. As previously mentioned, the non-existence of a common GS between any subset of n processes is interpreted as an empty GS between these n processes. Therefore, the number of links in LARGs is usually relatively small, and thus does not constitute an obstacle to the use of LARGs.

In fact, most of the existing design description models (MachineCharts, ObjecTime, ...) only allow 2-way interactions, which means that LARGs generated from these types of design models will have empty m -way interaction GSs (for $m > 2$). In this case, the

analysis made on LARG will be limited to 2-way interaction GSs and the number of GSs is reduced to $n*(n-1)/2$.

Also, since LARGs use different GSs for different interactions, their use facilitates the analysis of ARs. Thus, LARGs can be used as a validation and analysis framework for architectural designs (see §7). LARGs enable the detection of design errors and non-determinism.

Chapter 7 Issues in Formal Interpretation of LARGs

In this chapter, we discuss two important issues related to the formal interpretation of LARGs using LOTOS:

- 1^o) design errors (§7.1), and
- 2^o) groupings in both deterministic and non-deterministic LARGs (§7.2).

7.1 Design Errors

Design errors detection constitutes a crucial issue in system design. Design errors must be detected at the design stage of system development. Therefore design errors detection must be part of our formal interpretation method. However, because our method is only concerned with architectural behavior, the errors detected by the method are architectural errors. We identify three types of architectural errors: **partially defined gate errors** (§7.1.1), **undefined gate errors** (§7.1.2) and **ambiguously defined gate errors** (§7.1.3). If a gate g of the LARG is neither partially defined nor undefined, we say that g is **completely defined**, and if the same gate g is completely defined and unambiguously defined, we say that g is **correctly defined**. For every type of errors presented in this section, we discuss the possible solutions for error correction.

In this thesis, we define a **architectural deadlock** as a possible deadlock caused by the architectural organization of the system. An architectural deadlock can either be direct, as in the case of partially defined gate errors, or caused by grouping side effects, as in the case of undefined gate errors. We say that a architectural deadlock is caused by a **grouping side effect**, if the architectural deadlock only comes as the result of the grouping process. Examples of architectural deadlocks are provided in figure 7.1 (direct architectural deadlock) and figure 7.2 (architectural deadlock caused by grouping side effect).

7.1.1 Partially Defined Gates

Partially defined gate errors occur when N ($N \geq 2$) components must interact together on a certain gate g which is defined in the label set of some processes and not defined in others. Formally, we say that a gate $g \in G_i$, where $Procs(G_i) = \{P_i, P_j, \dots, P_n\}$, is a partially defined gate iff there exists $k, l \in \{i, j, \dots, n\}$ such that $g \in L(P_k)$ and $g \notin L(P_l)$. Therefore, each and every gate g contained in a GS G_i must be contained in the label set of all processes linked to G_i .

There exists two possible corrections for partially defined gate errors:

- 1) remove the partially defined gate g from the GS G_i , or
- 2) add the gate g , which is only partially defined, to the label set of every process linked to G_i such that g becomes completely defined.

For example in figure 7.1, gate a is considered to be a partially defined gate error. That is because $P1$ and $P2$ must interact on gate a , and a is a gate of $P1$ but not a gate of $P2$. In this case, there is a possible deadlock in process $P1$ when process $P1$ arrives at the point where it must interact with $P2$ on gate a .

Figure 7.1 Partially defined gate error

7.1.2 Undefined Gates

Undefined gate errors occur in a LARG when there exists a GS G_i containing a gate g ($g \in G_i$) such that g is not defined in any of the label set of the N processes which are linked to G_i . Formally, we say that a gate $g \in G_i$, where $Procs(G_i) = \{P_i, P_j, \dots, P_n\}$, is an undefined gate iff $g \notin L(P_k), \forall k \in \{i, j, \dots, n\}$.

However, unlike partially defined gate errors, undefined gate errors do cause immediate error in the system; errors will only occur as a possible consequence of grouping. This is what we call a **grouping side effect**. If not detected, undefined gates can possibly cause

architectural deadlock (figure 7.2) by grouping side effect. The same type of corrections as the ones defined for partially defined gate error can be applied to undefined gate errors. However, in most cases, the correction will consist in removing undefined gates from the gate set (correction type 1).

Figure 7.2 illustrates such a case. In figure 7.2a, we have a LARG in which c is a gate between $P3$ and the environment. However, c is also contained in the gate set between $P1$ and $P2$ even if c is neither a gate of $P1$ nor a gate of $P2$. Hence, c constitutes, with respect to our definition, an undefined gate error. Therefore, if this error is not detected by the interpretation method, grouping side effects will cause architectural deadlock. This is illustrated in figure 7.2b in which $P3$ must interact with $P1$ on gate c , but c is not a gate of $P1$.

Figure 7.2 Undefined gate error

7.1.3 Ambiguously Defined Gates

There exist ambiguously defined gate errors in a LARG when a gate name g is used to represent both N -way interactions and M -way interactions ($N \neq M$). Ambiguously defined gate errors are different from the two previous types of errors in such that they are purely conceptual, or semantic. Unlike the architectural deadlock errors and undefined gate errors, they do not cause deadlock. They result from an ambiguous description, or conception, of the system. In order to have an unambiguous description of a system, we need to associate a unique type of interaction (N -way interaction vs. M -way interaction, $M \neq N$) to every gate. Therefore, in order to correctly specify the system, this type of errors needs to be detected and corrected. The correction, for this type of errors, implies the renaming of one of the two gates which have the same name. This type of correction is not restrictive, with

respect to the design process, since the semantics of an N-way interaction and an M-way interaction is clearly different.

Figure 7.3 illustrates an ambiguous gate interaction error correction. In figure 7.3a, we have a LARG in which gate *c* constitutes an ambiguous gate interaction error. Gate *c* is, in this LARG, a 3-way interaction between P1, P2 and P3 and also a 2-way interaction between P4 and P5. In figure 7.3b however, we give a corrected version of figure 7.3a where the name of the 2-way interaction between P4 and P5 has been replaced by *c'*.

Figure 7.3 Correction of ambiguously defined gate error

Case Study: The Elevator System

Design errors in relation with MachineCharts

In MachineCharts, the set of interaction wires for a given component is defined as the set of all wires linked to the component. Thus, each wire *g* is defined in set of interaction wires of every component connected to gate *g*, and hence all the gates in the LARGs generated from MachineCharts designs are completely defined. Also, because MachineCharts only provides 2-way interaction, gates are unambiguously defined. Hence, MachineCharts only provides correctly defined interactions.

However, since the generation of LOTOS architectural expressions from LARGs requires error free LARGs, it may be important to validate the LARGs in relation with design errors to make sure that no error has been introduced in the LARGs by the architectural mapping method.

7.2 Determinism vs. Non-Determinism

In general, several different AEs may be generated from a given LARG. These different AEs result from different groupings of the former LARG. For this reason, we say that LARGs, and more generally speaking ARs, are ambiguous with respect to the generation of AEs. In particular, because LOTOS only possesses binary infix operators, we face this problem when using LOTOS as a formal semantic model. In analyzing the different architectural expressions generated from a given LARG, we observed that for certain types of LARGs, all LOTOS architectural expressions generated from a given LARG are equivalent, while for other types of LARGs, LOTOS architectural expressions generated from a given LARG are not all equivalent. We also observed that for the second type of LARGs (the ones for which the LOTOS architectural expressions are not all equivalent) some generated AEs are correct with respect to the former LARG, while others are not. We call this problem the **LAEG problem**. The resolution of this problem constitutes a critical issue for the definition of the LAEG method.

After analyzing the different types of LARGs in relation with the LAEG problem, we came to the conclusion that the former type of LARGs corresponds to **deterministic** LARGs (§7.2.1), and the second type corresponds to **non-deterministic** LARGs (§7.2.2). Therefore, in dealing with LARGs, it is crucial to identify non-deterministic ones so that an LARG is not wrongly interpreted by setting some illegal groupings, i.e. groupings that do not preserve the interaction semantic of the LARG.

In this section, we analyse the LAEG problem in relation with both deterministic and non-deterministic LARGs.

7.2.1 Deterministic LARGs

Definition

Conventionally, deterministic LARGs (ARs) are defined as being LARGs in which the set of processes that participate in any given interaction can be determined simply by the knowledge of the gate name, i.e. LARGs in which all GSs and all process environments ($Env(P_i)$) are mutually disjoint. In order to distinguish this definition from ours, we call this type of LARGs **totally deterministic**. However, because the goal of LAEG is to linearize a given LARG into a binary grouped equivalent one by successive grouping transformations and because groupings only involve internal GS transformations (groupings are not concerned with $Env(P_i)$), we, for the purpose of the LAEG analysis, generalize the definition of deterministic LARGs.

Therefore, we call **deterministic** any LARG in which the set of processes that participate in any given internal interaction can be determined simply by the knowledge of the gate name, i.e. any LARG in which all GSs are mutually disjoint. This definition of deterministic LARG, in opposition to the conventional one, does not require that process environments ($Env(P_i)$) are mutually disjoint. Formally, an LARG P is deterministic iff

$$G_i \cap G_j = \emptyset \text{ for all } G_i \in G(P) \text{ and } G_j \in G(P) (i \neq j).$$

An example of deterministic LARG, as defined in this thesis, is given in figure 7.4. We see from this figure that both process P1 and P2 can interact with the environment on gate e . However, this LARG is considered to be deterministic.

Figure 7.4 Example of deterministic AR

Groupings in Deterministic LARGs

In deterministic ARs, all possible groupings, and generated AEs (binary groupings), are equivalent and correct with respect to the former LARG. For this reason, we say that the type of ambiguity occurring in deterministic LARGs is only **syntactic ambiguity**.

We distinguish two types of deterministic LARGs: **linear LARGs** and **non-linear LARGs**.

Linear LARGs

We say that a given LARG P is **linear** iff

$$\begin{aligned} &\forall G_i \in G(P), |Procs(G_i)| = 2, \text{ and} \\ &\exists P_i, P_j \in P(P) \ (i \neq j) \text{ such that } |GSets(P_i)| = 1 \text{ and } |GSets(P_j)| = 1, \text{ and then} \\ &\forall P_k \in P(P) \ (k \neq i, j), |GSets(P_k)| = 2. \end{aligned}$$

Also, if the LARG does not contain any GS, i.e. $|G(P)| = 0$, then it is considered to be linear.

In linear LARGs (figure 7.5a), syntactic ambiguity can be resolved by grammar associativity rules. Parallel operators are assumed to be right associative in LOTOS. Thus, LOTOS architectural expressions can be directly generated from linear LARGs. Unfortunately, we have, in the general case, non-linear LARGs. Figure 7.5b and figure 7.5c illustrates two cases of non-linear LARGs. In non-linear LARGs, the left associative rule is not sufficient to remove syntactic ambiguity. Therefore, there is a need for using grouping to eliminate this type of ambiguity.

Figure 7.5 Linear vs. non linear LARGs

Example

Figures 7.6 and 7.7 together illustrate groupings in deterministic LARGs. Figure 7.6 illustrates the same LARG as the one given in figure 7.5.

Figure 7.6 General 3-component deterministic LARG

The application of grouping on the LARG of figure 7.6 may lead to any of the three binary grouped LARGs illustrated in figure 7.7. The LOTOS architectural expressions associated with these three LARGs are equivalent (strong bisimulation). Therefore, in LARG, syntactic ambiguity can be removed using groupings. Any of the three expressions given below are possible LOTOS AEs for the LARG of figure 7.6. The choice that is made may have some impact on the architecture or the performance of the final design of the system.

Figure 7.7 Three different groupings.

The LOTOS AEs corresponding to these three groupings of figure 7.7 are:

- (a) $P1[a,b,c,e] \mid [a,b,c] \mid (P2[a,b,d,e] \mid [a,d] \mid P3[a,c,d])$
- (b) $P2[a,b,d,e] \mid [a,b,d] \mid (P1[a,b,c,e] \mid [a,c] \mid P3[a,c,d])$
- (c) $P3[a,c,d] \mid [a,c,d] \mid (P1[a,b,c,e] \mid [a,b] \mid P2[a,b,d,e])$

7.2.2 Non-Deterministic LARGs

Definition

In opposition to deterministic LARGs previously defined, we call **non-deterministic** any LARG in which the set of processes which participate in a given internal interaction can not be determined simply by the knowledge of the gate name. We identify non-determinism, in

a given LARG, when there exist a gate g which is such that g is contained in more than one GS. Formally, we identify non-determinism in a given LARG P iff

$\exists g$ such that $g \in G_i$ and $g \in G_j$ for some $G_i \in G(P)$ and $G_j \in G(P)$ ($i \neq j$).

An example of non deterministic LARG is given in figure 7.8. We observe that in this LARG, gate b is contained in two different gate sets.

Figure 7.8 Example of non-deterministic LARG

Groupings in Non-Deterministic ARs

In the case of non-deterministic LARGs, the generation of AEs from a given LARG leads to non-equivalent AEs. In fact, some AEs are correct with respect to the interaction semantic of the former LARG, while others are not. The incorrect AEs result from illegal groupings. For this reason, we say that the type of ambiguity occurring in non-deterministic LARGs is **semantic ambiguity**. The purpose of this section (§7.2.2), is to analyse possible groupings in non-deterministic LARGs in order to define an LAEG method which only generates correct AEs. We called this problem the "LAEG problem".

We know that LAEG is only possible for certain LARGs, it is not possible in general; we can not generate a LOTOS AE from any arbitrary topology of interacting processes (components). An example of LARG from which the generation of LOTOS AE is not possible is given in figure 7.9. Any grouping in this LARG will change the semantic of the 2-way interaction gate a for a 3-way interaction one, which does not reflect the semantic of the former system. In such a case, we say that there exist no possible LAEG for the LARG. More formally, we say that there exist no possible LAEG for a given LARG iff the level of non-determinism in the LARG is such that it can not be remove without affecting the

interaction semantic of the system, i.e. any sequence of groupings in the LARG eventually leads to a transformation of the interaction semantic of the system. In this thesis, we describe different types of grouping that can be used to removed non-determinism in LARGs (see §7.2.2.1 and §7.2.2.2).

Figure 7.9 No LAEG solution LARG

The LAEG problem has also been discussed in Vigder's thesis (called the wiring problem in [Vigder92]). In [Vigder92], the solution adopted is the "default process" solution; it simulates the behavior of the system by creating artificial processes. However, this solution does not preserve the architecture of the former system. In this project, one of the main reasons for the generation of LOTOS AEs is the possibility to apply LOTOS CPTs (Correctness Preserving Transformations) on LARGs. Therefore, the architecture of the system constitutes a main concern, and thus "the default process" solution can not applied in our method. We need to find a solution that preserves the architecture of the system.

In this thesis, we identify two particular types of non-determinism that are allowed in design models (see §7.4), and from which LAEG is possible under some conditions: **non-deterministic parallel-interactions** (§7.2.2.1) and **non-deterministic interaction-choice** (§7.2.2.2).

Since they are allowed in design models, solutions for these types of non-determinism is important for the definition of the LAEG method. However, even if we have solutions for certain types of LARGs in which these types of non-determinism exist, the combination of them in a single LARG may lead to a situation where LAEG is impossible. Therefore, for the two types of non-determinism mentioned above, we give both examples of a LARG from which LAEG is possible and one from which LAEG is not possible. Also, in cases where LAEG is not possible, we discuss possible solutions.

7.2.2.1 Non-deterministic parallel-interactions

Definition

We say that a gate g is the source of non-deterministic parallel-interactions in an LARG P , iff g is contained in more than one GSs in P and every process P_i which possesses gate g is linked to one and only one GS containing g . In this definition, g can be of any types of gates, i.e. N -way interaction for any $N \geq 1$. This type of non-determinism has been identified by Bolognesi (called ambiguity in [Bolognesi 90]).

Non-deterministic parallel-interactions are considered to be design errors in most design models (MachineCharts, ObjecTime). However, because some design tools allow such non-determinism (e.g. EGL) and because they are allowed in LOTOS, it is important to allow non-deterministic parallel-interactions in our method. Therefore, our method must provide a formal way to identify non-deterministic parallel-interactions in LARGs and a way to generate LOTOS expressions from such LARGs.

LAEG solutions

We now define two different types of grouping that can be used to remove non-deterministic parallel-interactions in LARGs. Also, we illustrate three different examples of LARGs in which there exists non-deterministic parallel-interactions. For these examples we discuss possible LAEG solutions.

1- Interacting process grouping

In interacting process grouping, we group together all processes which are linked to the same GS containing the non-deterministic gate. In this case, there is only one possible grouping. Interacting process grouping is illustrated in example 1 (figure 7.10b).

2- Process subset decomposition grouping

In process subset decomposition grouping, we group in N process subsets all processes which participate in the N -way non-deterministic interaction gate. In fact, each subset of processes must contained exactly one process linked to each of the GS which contains the non-deterministic gate. The process subset decomposition must be such that the cartesian

product of the N subsets must contain all N -tuples of interacting processes existing in the former LARG. If however the cartesian product of the subsets contains more tuples of interacting processes than there exists in the former LARG, extended gate identifiers are used to reduced the number of interactions between processes on the non-deterministic gate. In the case of process subset decomposition, there may exist many possible groupings, however decomposition for which the cartesian product contains all the tuples of interacting processes of the former LARG is correct and may be used. Process subset decomposition grouping is the most general and flexible type of grouping that can be used to eliminate non-determinism in LARGs. It can be applied to any types of non-determinism. In fact, process subset decomposition grouping seems to be the only type of grouping that can be used to remove non-determinism in the case of arbitrary non-deterministic gates, i.e. non-deterministic gates that are the source of neither non-deterministic parallel-interactions nor non-deterministic interaction-choices. If process subset decomposition grouping does not allow the elimination of non-determinism in a given LARG, then there is no LAEG solution for this LARG. Process subset decomposition grouping is illustrated in both example 1 (figure 7.11) and example 2 (figure 7.12).

Example 1 Interacting process grouping

Figure 7.10 illustrates a case of non-deterministic parallel-interactions. Figure 7.10a gives a LARG in which gate a constitutes the source of a non-deterministic parallel-interactions. In this LARG, we observe that either P1 can interact with P2 on gate a or P3 can interact with P4 on the same gate. We see from figure 7.10b and 7.10c, where two different groupings of the same LARG are given, that in this case different groupings are not equivalent. In figure 7.10c, for example, P1 can interact with P4 on gate a , while this is not possible in the grouping of figure 7.10b.

The LARG of figure 7.10b is the correct interpretation of figure 7.10a. We say that figure 7.10b is equivalent to figure 7.10a. Figure 7.10c corresponds to an incorrect interpretation of figure 7.10a.

Figure 7.10 Interacting process grouping

However if we add extended gate identifiers, the grouping of figure 7.10c becomes equivalent to the LARG of figure 7.10a, and also equivalent to figure 7.10b. In the LARG of figure 7.10c, we may add two different extended gate identifiers for gate *a* to distinguish between interactions on *a* involving P1 and P2 from the one involving P3 and P4. Then, the interaction on gate *a* will not only require that processes are ready to interact on *a*, but will also require the matching of extended gate identifiers between the processes which interact on *a*. For example, gate *a* may be replaced by *a'1* in P1 and P2, and by *a'2* in P3 and P4. Therefore, if gate *a* occurs, P1 and P4 can not interact together anymore since their extended gate identifiers do not match (P1 is *a'1* and P4 is *a'2*). The resulting LARG is given in figure 7.11.

Figure 7.11 Addition of Extended gate identifier in LARGs

Example 2 Process subset decomposition grouping

In the following example (figure 7.12), we give a LARG for which interacting process grouping does allow to eliminate non-determinism. In this LARG, gates a and b are both sources of non-deterministic parallel-interactions. Trying to remove non-determinism on gate a using interacting process grouping leads us to the LARG of figure 7.12b. We know that for this type of LARG there is no LAEG solution. The result of the elimination of non-determinism on gate b using interacting process grouping leads to a similar LARG for which there is no LAEG solution. However, in this case, we may apply process subset decomposition grouping and obtain the LARG of figure 7.12c which constitutes a correct interpretation of the former LARG (figure 7.12a) and for which LAEG is possible.

Figure 7.12 Process subset decomposition grouping

Example 3 Hybrid grouping solution

In some cases, any solution for LAEG involves the use of extended gate identifiers. For example in figure 7.13, if we do not use extended gate identifiers, neither 7.13b nor 7.13c correctly expressed the architectural behavior of figure 7.13a. In figure 7.13b interactions involving P1 and P4 on gate *b* are allowed while they are not in figure 7.13a, and figure 7.13c allows interactions involving P1 and P4 on gate *a* while they are not in figure 7.13a. In this case, neither of the two groupings is a correct interpretation of figure 7.13a.

Figure 7.13 LARG without "pure" grouping solution

The solution consists in adding two different extended gate identifiers to gate *b* to distinguish between interactions on *b* involving P1 and P3 from the one involving P2 and P4. In this case, the binary grouped LARG of 7.13b becomes a correct solution for the former LARG (figure 7.13a). Then the interaction on gate *b* will not only require *b*, but will also require the matching of the extended gate identifiers. For example, gate *b* may be replaced by *b/1* in P1 and P3, and by *b/2* in P2 and P4. Therefore if the gate *b* occurs, P1 and P4 can not interact together anymore since their extended gate identifier do not match (P1 is *b/1* and P4 is *b/2*). Similarly, the binary grouped LARG of 7.13c becomes a correct solution if we apply the same technique to gate *a*. The modified versions of the LARGs of figure 7.13b and 7.13c are given in figure 7.14.

Figure 7.14 Hybrid grouping solution

Conclusion

In order to generate LOTOS expressions from LARGs, we need to formally identify cases of non-deterministic parallel-interactions. In such cases, if a gate g is the source of a non-deterministic parallel-interactions, we may try to apply interacting process grouping on g . If this lead to a LARG from which there is no LAEG solution, then the only possible solution for LAEG consist in finding a process subset decomposition grouping which allows to remove the non-determinism on g . If there exists no such process subset decomposition grouping, then there is no solution for LAEG.

7.2.2.2 Non-deterministic interaction-choice

Definition

We say that a gate g is the source of non-deterministic interaction-choice in a LARG P , iff g is contained in more than one GSs in P and every GS containing g is linked on one side to a constant set of processes, called the **root process set** of the non-deterministic interaction-choice, and on the other side to distinct processes, i.e. processes which are linked to only one GS containing g , called the **choice process set** of the non-deterministic interaction-choice. Thus, every process which possesses gate g is either linked to every GS containing g or to one and only one GS containing g . In this definition, g can be of any types of interactions, i.e. N -way interaction for any $N \geq 1$. Also, because the number of processes linked to GS containing g is the same for every GS (otherwise g is ambiguously defined)

and because the number of processes contained in the root set is also constant, the number of choice processes linked to GS containing g has to be the same for every GS.

In relation with design models, non-deterministic interaction-choices are considered to be design errors in some design models (e.g. ObjecTime), and allowed in others (e.g. MachineCharts, EGL). Also, they are allowed in LOTOS. Thus, it is important to allow non-deterministic interaction-choices in our method. Therefore, our method must provide a formal way to identify non-deterministic interaction-choices in a LARG and a way to generate LOTOS expressions from such LARGs.

For 2-way interactions, we say that a gate g is the source of non-deterministic interaction-choice in a LARG P , iff g is contained in more than one GS in P and every GS containing g is linked on one side to a unique process P_r , called the **root process** of the non-deterministic interaction-choice, and on the other side to a **choice process** P_j ($j \neq r$), i.e. P_j is linked to only one GS containing g . In other words, every GS containing g is linked on one side to P_r and on the other side to an another process P_j such that P_j is linked to only one GS containing g (P_j is the choice process).

LAEG solution

For non-deterministic interaction-choices, we define only one type of grouping to eliminate non-determinism. This type of grouping is simply called **non-deterministic interaction-choice grouping**. Also, we present three examples of LARGs in which there exists non-deterministic interaction-choices. In the first two examples, the non-determinism can be removed in the LARG using non-deterministic interaction-choice grouping, while in the third example non-deterministic interaction-choice grouping is not a solution. In fact, for this third example, there exists no LOTOS solution.

Non-deterministic interaction-choice grouping

In non-deterministic interaction-choice grouping, we group together all choice processes, i.e. all processes contained in the choice process set. Thus, if g is an $(N+M)$ -way non-deterministic interaction-choice gate where N is the number of root processes of g and M is the number of choice processes for each GS containing g , the application of non-deterministic interaction-choice grouping will create a new grouping in which g is an M -way non-deterministic parallel-interaction gate. If M is equal to one, we say that non-

deterministic interaction-choice grouping eliminates non-determinism. However, if M is greater than one, the application of non-deterministic interaction-choice grouping eliminates non-deterministic interaction-choice from the LARG, but it introduces non-deterministic parallel-interactions, i.e. the application of non-deterministic interaction-choice grouping transforms a non-deterministic interaction-choice gate into a non-deterministic parallel-interaction gate. In this case, the non-deterministic parallel-interaction gate can be removed by applying one of the type of grouping defined in §7.2.2.1, i.e. interacting process grouping or process subset decomposition grouping. Such an example is illustrated in example 2.

Example 1 2-way non-deterministic parallel-interactions choice grouping

Figure 7.15a illustrates a case of non-deterministic interaction-choice. In this LARG, gate a is the non-deterministic interaction-choice gate. We observe that $P1$ can interact with either $P2$ or $P3$ on the 2-way interaction gate a . We also observe that $P2$ and $P3$ are in parallel (interleaving). Therefore, in order to have interactions on gate a , we need to have $P1$ ready to interact on a and either $P2$ or $P3$ also ready to interact on a .

Figure 7.15 2-way non-deterministic parallel-interactions choice grouping

We see from figure 7.15b and 7.15c, where two different groupings of figure 7.15a are given, that, in the case of non-deterministic interaction-choice, different groupings are not equivalent. In the first case a is a 2-way interaction while in the second case a is a 3-way interaction. Thus, we see that, in order to generate LOTOS expressions from LARGs, we need to formally define a method which allows the identification and the manipulation of non-deterministic interaction-choices. In this example, the LARG of figure 7.15b is the correct interpretation of figure 7.15a, and figure 7.15c corresponds to an incorrect interpretation of figure 7.15a.

Also, we observe that in this case, the application of non-deterministic interaction choice grouping allows to remove non-determinism in the LARG.

Example 2 4-way non-deterministic interaction-choice grouping

In figure 7.16, we give an example of a non-deterministic interaction-choice LARG for which non-deterministic interaction-choice grouping does not eliminate non-determinism, but it rather transforms it into non-deterministic parallel-interactions. In figure 7.16a, an ungrouped non-deterministic interaction-choice LARG is given. The application of grouping on this LARG transforms the non-deterministic interaction-choice LARG into a non-deterministic parallel-interaction LARG (figure 7.16b). Then, as mentioned earlier, on one the two grouping techniques defined for non-deterministic parallel-interactions can be applied. The results of the application of these two grouping techniques are given in figure 7.16c (interacting process grouping) and 7.16d (process subset decomposition grouping).

Figure 7.16 4-way non-deterministic parallel-interactions choice grouping

Example 3 Impossible non-deterministic interaction-choice grouping

We saw from the previous examples how non-deterministic interaction-choice grouping can be used to remove non-determinism from LARGs in which non-determinism is of the type non-deterministic interaction-choice. However, the combination of non-deterministic interaction-choice gates in a given LARG may lead to a LARG for which LAEG is not possible. Figure 7.17 illustrates such an example. In this example, it is impossible to group processes in a binary group manner such that the interaction semantic of the resulting AR is equivalent to the former one. Therefore, because LOTOS AEs can only be generated from binary grouped LARGs, it is not possible, with the existing LOTOS operators, to generate a LOTOS AE from this type of ARs.

Figure 7.17 Impossible non-deterministic parallel-interactions choice grouping

Conclusion

Thus, in order to generate LOTOS expressions from LARGs, we need to formally identify cases of non-deterministic interaction-choices. In such cases, if g is the gate involved in non-deterministic interaction-choices, we must apply non-deterministic interaction-choice grouping. These must constitute first priority groupings.

7.2.3 Discussion (Possible Solutions to LAEG problem)

In order to define the LAEG method, we either need to find a general solution or to define a set of partial solutions to the LAEG problem. Some solutions have already been proposed in the previous section (§7.2.2), but no solution covers all possible cases. We now review the previously proposed solutions and discuss a general solution that can be applied when others can not.

First, we saw that if the LARG is linear, we can easily eliminate the syntactic ambiguity using the LOTOS left associative rule. Unfortunately in the general case, LARGs are non-linear. Thus, the associative rule is not sufficient for LAEG.

In the case of non-linear LARGs, the solution consists of using successive groupings to obtain a binary grouped LARGs. If the LARG is deterministic, any grouping is correct with respect to the interaction semantic of the former LARG. If the LARG is non-deterministic, then we have to find a grouping sequence that allow to remove the non-determinism in the LARG. For this purpose, we defined three different grouping techniques. These are: interacting process grouping and process subset decomposition grouping for non-deterministic parallel-interaction gates, and non-deterministic interaction-choice grouping for non-deterministic interaction-choice gates. If the non-determinism is of the general type, then only process subset decomposition grouping can be applied.

However, we saw that in cases where non-determinism (or semantic ambiguity) can not be removed using the different grouping techniques previously defined in this thesis (as it is the case in figure 7.17), the generation of a LOTOS architectural expression is not possible, i.e. we can not directly generate a LOTOS architectural expression from the LARG. To solve this problem, we propose two different solutions:

- 1- Conduct gate relabelling on non-deterministic gates in order to obtain a deterministic LARG from a non-deterministic one; in this case, if an non-deterministic gate g is contained in N GSs, then g will be replaced by a different gate identifier ($g_1, g_2, \dots, g_n$) in each GS in which g is contained. However, this solution has a direct impact on the behavioral description of processes which synchronize on non-deterministic gates in such a way that we will have to conduct the same gate relabelling in the behavioral description of implicated processes. Also, if a process P_i is part of the root of an non-deterministic interaction-choice on a gate g , we will have to replace each occurrence of g in the behavioral description of P_i by: $g_1 [] g_2 [] \dots [] g_n$ (where n is the number of choices of the non-deterministic interaction-choice g).
- 2- Define a general interaction operator Op that can be used to describe any arbitrary LARG P , i.e. an operator which may be used to describe any topology of interacting processes. For this purpose we define the following operator:

$$Op(L_1, L_2, \dots, L_n), \text{ and } L_i = \langle G_i, Procs(G_i) \rangle \quad G_i \in G(P).$$

In fact, a link L_i is defined for each $G_i \in G(P)$, and thus each GS G_i of a LARG P is described by a distinct L_i in the operator Op . Then, every time an action g occurs, we construct the set of links SL ($SL \subseteq \{L_1, L_2, \dots, L_n\}$) which contains every link L_i ($L_i = \langle G_i, Procs(G_i) \rangle$) such that:

- 1°) G_i contains g , and
- 2°) every P_i which is linked to G_i is ready to execute g .

Then, one SL_i is chosen in a non-deterministic manner and force all processes linked to G_i to interact on g .

In other words, if an action g occurs, we choose in a non-deterministic manner a link L_k in the set of all link in which g is contained and such that all processes linked to the GS are ready to interact on g , and then we force all processes contained $Procs(G_k)$ to interact on g .

Case Study: The Elevator System

Non-determinism in relation with MachineCharts

In relation with non-determinism, MachineCharts allows critical non-deterministic interaction-choices by means of multiple wires linked to a given port. However, because wire identifiers are unique, MachineCharts does not allow non-deterministic parallel-interactions.

In the elevator system LARG, we observe in figure CS6.6 (the ungrouped LARG of the elevator system) that there exist non-deterministic interaction-choices on both gate cs and gate msg . However, we also observe, in figure CS6.1, that the non-determinism on these gates can be removed by grouping respectively all Car processes together and all Car_SC processes together.

7.4 Relation Between these Issues and Description Models

In this section, we discuss the existence of design errors (as defined in §7.1) and non-determinism (as defined in §7.2) in relation with EGL and ObjecTime. Such a discussion for MachineCharts has already been provided in the "Case Study" boxes of this chapter.

7.4.1 Relation with EGL

EGL does not conduct any validation nor analysis on ARs. It uses LOTOS tools as both validation and analysis frameworks. Therefore, any kind of errors or non-determinism are allowed at the design stage in EGL, errors and non-determinism have to be identified by LOTOS tools (e.g. ELUDO (ISLA)) at the compilation stage.

7.4.2 Relation with ObjecTime

ObjecTime defines actor (component) interactions in terms of bindings and ports. In order to represent an interaction between two components, a port is added to both components and a binding is draw to link the two ports together. A common and unique identifier is then associated with the ports and the binding. Then, the set of interactions on which an actor can interact is defined as the set of ports owned by the actor. Thus, in ObjecTime interactions are completely defined. Also, because ObjecTime only provides 2-way interactions, interactions are unambiguously defined, and hence correctly defined.

Moreover, the relation between ports and bindings is one-to-one, i.e. a binding can only be connected to one port and a port can only be the source of one binding. Therefore, in relation with non-determinism, ObjecTime does not allow neither non-deterministic parallel-interactions nor non-deterministic interaction-choices. Non-deterministic parallel-interactions is not allowed because interaction identifiers are unique, and non-deterministic interaction-choices is not allowed because the relation between ports and bindings is one-to-one.

Chapter 8 The LAEG Method

In this chapter, we present the LAEG method (§8.1) and some of its possible applications (§8.2).

8.1 Description of the LAEG method

In this section, we present the LAEG method itself. As previously discussed, the method aims at generating LOTOS AEs from LARGs. Then, in order to generate LOTOS AEs from LARGs, we first need to conduct LARG analysis. As discussed in the §7, LARG analysis must include design error analysis and determinism analysis. In this section, we integrate the analysis of §7 in a global LARG analysis method (§8.1.2). The result of the analysis phase must terminate with an error free deterministic LARG. Therefore, the responsibility of the analysis phase is to correct design errors and remove non-determinism in the former LARG. Then, the second phase consists in generating a binary grouped LARG which is equivalent to the former one. The generation of LOTOS AE phase involves successive applications of the grouping algorithm, in fact the algorithm is applied until the resulting LARG is binary grouped.

Therefore, the method is conducted in two distinct phases:

- 1°) LARG analysis, and
- 2°) generation of LOTOS AEs.

We now present the LAEG algorithm (§8.1.1) and describe the two LAEG phases (§8.1.2 and §8.1.3).

8.1.1 LAEG Algorithm

```
procedure LAEG;  
var  
P: LARG;  
deterministic : Boolean;  
{The Analysis procedure serve to detect architectural errors and non-determinism in the  
LARG P. It also conducts error correction phases and a non-determinism elimination phase.  
If the input LARG is one for which there exists a grouping sequence to eliminate non-  
determinism, then the result of the analysis procedure is an error free and deterministic  
LARG and the boolean variable Deterministic is set to true. Otherwise, the boolean variable  
Deterministic is set to false.}  
procedure Analysis(var P: LARG; var Deterministic: Boolean);  
var  
error : Boolean;  
erroneous_gate : Gate_Name;  
non-determ_gates : Set_of_Non-determ_Gates;  
begin  
  repeat  
    error := false;  
    AnalysisPhase1(P, error, erroneous_gate);  
    if error then CorrectError(P, erroneous_gate);  
  until not(error);  
  repeat  
    error := false;  
    AnalysisPhase2(P, error, erroneous_gate);  
    if error then CorrectError(P, erroneous_gate);  
  until not(error);  
  deterministic := true;  
  AnalysisPhase3(P, deterministic, non-determ_gates);  
  if not(deterministic) then  
    FindGroupingSeq(P, non-determ_gates, deterministic);  
end;  
begin  
  Analysis(P, deterministic);
```

```

{Generation of LOTOS AE phase (described in §8.1.3)}
if deterministic then
    BinaryGrouping(P)
else
    UseAlternativeSolution;    (there exists no LOTOS solution)
end.

```

8.1.2 LARG Analysis Method

In LARG, we conduct three phases of analysis which allow the detection of both architectural errors and non-determinism. The two first phases allow architectural error detection, while the third one allows the detection of architectural non-determinism. If the LARG is non-deterministic, the third phase also allows to determine the type of non-determinism (non-deterministic parallel-interactions, non-deterministic interaction-choice or general non-determinism). The three analysis phases must be conducted successively in the given order. Also, they have been defined in such a way that the second phase can only start after the analyzed LARG has proven to be error free with respect to the first phase, and similarly the third phase can only start after the analyzed LARG has proven to be error free with respect to the second phase.

Analysis Phase 1) Partially Defined Gate Errors and Undefined Gate Errors

We verify that each gate which is contained in a GS G_i is also contained in the label set of all processes which are linked to the G_i , i.e.

$$Gates(G_i) \subseteq (L(P_1) \cap L(P_j) \cap \dots \cap L(P_n)) , \text{ where } GSets(G_i) = \{P_i, P_j, \dots, P_n\}$$

Solution:

All gates that do not satisfy Condition 1 are either partially defined gate errors or undefined gate errors, and thus they have to be corrected. For a given partially defined gate error or undefined gate error, the correction may be either to remove the gate from all GSs in which the gate appear or to add the gate to the label set of all relevant components.

Analysis Phase 2) Ambiguously Defined Gate Errors

We verify that there exists no gate g such that g is both an N -way interaction gate and an M -way interaction gate (for $N \neq M$) in P :

$$\left(\bigcup_{P_i \in P(P)} Env(P_i) \right) \cap \left(\bigcup_{G_i \in G^2} G_i \right) \cap \dots \cap \left(\bigcup_{G_i \in G^n} G_i \right) = \emptyset.$$

where $G^k = \{G_i \in G(P) \mid |Procs(G_i)| = k\}$ and $|P(P)| = n$

Solution:

However, if a gate g does not satisfy Condition 2, we need to rename the gate g in such a way that the condition is satisfied (as mentioned in §7.1.3). This condition does not constitute a restriction in the design process since the functionality of N -way and M -way interactions is clearly different and therefore the renaming of one gate will help to destroy confusion in design interpretation. The result of this analysis phase is an error free LARG.

Analysis Phase 3) Non-Determinism Analysis

We check whether the LARG is deterministic or not. For this purpose, we need to check if all GSs are mutually disjoint. However, since we know at this point that there is no ambiguously defined gate, i.e. for all $N, M \geq 2$ ($N \neq M$) N -way interaction gates are distinct from M -way interaction gates, the search for non-deterministic gates is reduce to GSs involved in the same type of interactions, i.e. N -way interactions for every given $N \geq 2$. Formally, we must check that:

$$\forall n \geq 2, \forall G_i, G_j \in G(P) \ (G_i \neq G_j) \text{ such that } |Procs(G_i)| = |Procs(G_j)| = n, \\ G_i \cap G_j = \emptyset.$$

If the condition is not satisfied, we need further investigation to determine the type of non-determinism (non-deterministic parallel-interactions, non-deterministic interaction-choice or general non-determinism). Then, if g is the non-deterministic gate and if $G(P) \cap \{g\} = \{G_1, G_2, \dots, G_n\}$, we determinate the type of non-determinism by using the two following conditions.

1- Non-deterministic parallel-interactions

We say that the non-deterministic gate g is a non-deterministic parallel-interaction gate iff

$$\forall G_i, G_j \in \{G_1, G_2, \dots, G_n\} \ (G_i \neq G_j), \\ Procs(G_i) \cap Procs(G_j) = \emptyset.$$

2- Non-deterministic interaction-choice

We say that the non-deterministic gate g is a non-deterministic interaction-choice gate iff

$$\bigcap_{G_i \in \{G_1, G_2, \dots, G_n\}} Procs(G_i) \neq \emptyset, \text{ and}$$

$$\text{Root process set} = \bigcap_{G_i \in \{G_1, G_2, \dots, G_n\}} Procs(G_i)$$

$$\text{Choice process set} = \left(\bigcup_{G_i \in \{G_1, G_2, \dots, G_n\}} Procs(G_i) \right) \setminus \text{Root process set}.$$

Then, if the non-determinism is neither of the type non-deterministic parallel-interactions nor of the type non-deterministic interaction-choice, we deduce that the non-determinism is a **general non-determinism**.

Solution:

In prior to search such a grouping sequence, we first need to detect all non-deterministic gates. Then, the solution to non-deterministic LARGs consist in finding a grouping sequence that allows to remove the non-determinism from LARGs, i.e. remove the non-determinism on all non-deterministic gates. For a given LARG, the result of this analysis phase leads to either a deterministic LARG or to the conclusion that there exists no LOTOS AE for this LARG.

8.1.3 Binary Grouping Method

Binary grouping method

As previously discussed, the generation of LOTOS AEs from LARGs involves successive applications of the grouping algorithm, this method is called the **binary grouping (BinaryGrouping) method**.

The BinaryGrouping method is a recursive one. It consists, first, in partitioning the set of processes of a LARG P in two subsets ($S1$ and $S2$), not necessarily equal considering the

number of processes in each of them, and second, in applying $\text{Grouping}(P, \{S1, S2\})$ (defined in §6.2.4). Then, the method is recursively applied to both newly created sub-LARGs ($Gr1$ and $Gr2$) until the LARG is binary grouped. Formally, we define the binary grouping method (BinaryGrouping) as follow.

BinaryGrouping(P)

begin

$\text{Partition}(P(P))$ {in two subsets $S1$ and $S2$ }

$\text{Grouping}(P, \{S1, S2\})$ {this creates two interacting LARGs, $Gr1$ and $Gr2$, in P }

if $|P(Gr1)| > 2$ **then** $\text{BinaryGrouping}(Gr1)$

if $|P(Gr2)| > 2$ **then** $\text{BinaryGrouping}(Gr2)$

end.

Partition method

Because the number of LOTOS AEs generated from a given LARG is in general greater than one, we may need to choose a particular expression, based on some criteria. These partitioning criteria entirely defines the partition method. Therefore, in order to define the partition method, we first need to define partitioning criteria. For example, we may want to integrate performance criteria in the partition method. In order to do so, we would need to define some metrics on architectures. However, metrics are not included in this thesis. They constitute a research area for future research.

In this thesis, we suggest an half-cut partitioning solution for the partition method, i.e. at each partition step of the BinaryGrouping method, the set of processes is partitioned in two subsets $S1$ and $S2$ that contains respectively $\lceil n/2 \rceil$ and $n - \lceil n/2 \rceil$ processes, where $\lceil n \rceil$ is the smallest integer greater or equal to n . This solution has the advantage to be optimal with respect to the number of recursive call of the BinaryGrouping method.

Number of possible different binary groupings

We denote by $|\text{BinaryGrouping}(P^n)|$ the number of the possible different binary groupings in a given n -process LARG P . Also, we say that any 1-process LARG or 2-process LARG is binary grouped, and thus we have $|\text{BinaryGrouping}(P^1)| = |\text{BinaryGrouping}(P^2)| = 1$.

For a 3-process LARG, the only possibility consists in isolating one process (3 possibilities) and group the remaining two. Thus, since $|BinaryGrouping(P^2)| = 1$, we have $|BinaryGrouping(P^3)| = 3 \times 1 = 3$. With a 4-process LARG, we can either isolate one process ($C^4_1 = 4$ possibilities) and group the remaining three ($|BinaryGrouping(P^3)| = 3$), or we can group the processes two by two ($C^4_2 / 2 = 3$ possibilities). In the case of a 1-3 grouping, we have to recursively apply the binary grouping method to the 3-process LARG. Thus, $|BinaryGrouping(P^4)| = 4 \times 3 + 3 \times 1 = 15$. In such case, we need to divide C^4_2 by 2, because in the set of combinations of the form $C^n_{n/2}$, there exists, for each combination C , a combination C' (where C' is complement of C) such that C and C' are considered to be two different combinations, but they in fact lead to the same grouping. For example, grouping $P1$ and $P2$, or grouping $P3$ and $P4$ in a 4-process LARG lead to the same resulting LARG. Therefore, in order to obtain the number of different groupings, we have to divide $C^n_{n/2}$ by 2.

Figure 8.1 illustrates possible groupings in a 4-process LARG. It illustrates a general 4-process LARG in which only 2-way interactions are allowed (figure 8.1a) together with two possible groupings: a 2-2 grouping (figure 8.1b) and a 1-3 grouping (figure 8.1c). We observe that the LARG of figure 8.1b is binary grouped, while the one of figure 8.1c is not. Therefore, in order to generate LOTOS AEs, we need to apply the BinaryGrouping method to the 3-process grouping in figure 8.1c.

Figure 8.1 Different groupings in a 4-process LARG

In generalizing this function, we obtain the following recursive function that may serve to calculate the number of possible binary groupings in an N -process LARG (for a lack of space, we use BinGroup for BinaryGrouping in the writing of the following expressions):

$$|\text{BinGroup}(P^n)| = \begin{cases} \bullet & 1 & \text{if } n = 1, 2 \\ \bullet & \sum_{i=1}^{(n-1)/2} C_i^n \times |\text{BinGroup}(P^i)| \times |\text{BinGroup}(P^{n-i})| & \text{if } n \text{ is an odd number } > 2 \\ \bullet & \sum_{i=1}^{(n/2)-1} C_i^n \times |\text{BinGroup}(P^i)| \times |\text{BinGroup}(P^{n-i})| + (C_{n/2}^n / 2) \times (|\text{BinGroup}(P^{n/2})|)^2 & \text{if } n \text{ is an even number } > 2 \end{cases}$$

However, the BinaryGrouping method can only be applied to deterministic LARG. Therefore, as previously discussed, if the LARG is non-deterministic, we first have to remove the non-determinism before the binary grouping method can be applied. In cases where the level of non-determinism is not too high, we can remove it by imposing first

priority groupings (see §7.2.2). In other cases, i.e. cases where there exist no grouping sequence that allows to remove the non-determinism, there is no solution for the generation of LOTOS expressions. Therefore, the presence of non-determinism in the LARG will lead to a reduction of the number of processes to which the grouping function is applied, and hence will reduce the number of possible LOTOS AEs. For example, in figure 7.10, the presence of non-determinism imposes a 2-by-2 first priority grouping, that is P1-P2 and P3-P4, and thus there is only one possibility for the first grouping, and the number of all possible groupings is then equal to: $1 \times |\text{BinaryGrouping}(P^2)|^2 = 1$. In figure 6.10, the presence of non-deterministic interaction-choice imposes a 1-2 first priority grouping which results in the isolation of P1 and the grouping of P1-P2, and thus, in this case, the number of possible groupings is also equal to 1 ($1 \times |\text{BinaryGrouping}(P^1)| = 1$).

8.2 Possible Applications of the LAEG Method

We now discuss some possible applications of the LAEG method in the design process of distributed systems.

8.2.1 LOTOS Architectural Interpretation

As discussed in §4, the solution proposed for the definition of a LOTOS interpretation method requires the definition of both LOTOS architectural interpretation method and LOTOS behavioral interpretation method. Since LOTOS behavioral interpretation methods based on state machines already exist, we, in this thesis, aimed at developing a LOTOS architectural interpretation method. In §5, we discussed and set up a list of requirements for such a method. Then, the LARG model, and its associated LAEG method, has been defined under this list of requirements to provide an intermediate model solution for the definition of the LOTOS architectural interpretation method. The LARG model has been defined to allow the mapping of different types of architectural models. On the other hand, the LAEG method provides both a LARG analysis method, which allows the detection of both architectural errors and non-determinism in LARGs, and a binary grouping method, which allows the generation of LOTOS architectural expressions from LARGs.

Therefore, in order to completely define the LOTOS architectural interpretation method (see figure 5.1), we now only have to define an "Architectural Mapping (AM)" method. As discussed in §5, the relation between AM methods and architectural models is one-to-one.

Therefore, we need to define a different AM method for each design model from which we want to generate LOTOS AEs.

However, if we apply arbitrary process grouping for the purpose of LAEG, we may end up with groupings that are illogical with respect to the former architecture, e.g. a Buffer process, which takes place between two processes P1 and P2 which interact asynchronously via the Buffer process, can be grouped with a process P3 which has no relation with the Buffer process. This example is illustrated in figure 8.2. Some solutions may be defined to correct this problem, but since they are particular to description models, we do not propose any in this thesis. Any solution to this problem may be defined as partitioning criterion and placed in the Partition method.

Figure 8.2 Illogic LAEG grouping in LARG

Case Study: The Elevator System

LAEG from the elevator system design

In order to produce the complete LOTOS specification for the elevator system, we need to replace all AE's and BE's in the skeleton of the elevator system specification, given in specification CS6.1, by real LOTOS expressions. In this way, our LAEG method is used to generate LOTOS architectural expressions from LARGs. Again, LOTOS behavioral expressions (BE's) can be obtained using existing LOTOS behavioral interpretation method.

In this example, we generate architectural expressions from linear LARGs. We recall that LOTOS architectural expressions can be directly generated from linear LARGs. To linearize the Elevator_System LARG, we use the grouping given in figure CS6.6. All other LARG (Car_SC, Control, Coordination, Dispatch) are already linear. The architectural expressions corresponding to these LARGs are the following.

```

AEElevator_System := Building[fb]
                        |[fb]|
                        ( Coordination[fb, cs, msg]
                          |[msg]|
                          (Car1_SC[msg, cb1] ||| Car2_SC[msg, cb2] ||| Car3_SC[msg, cb3])
                        )
                        |[cs, cb1, cb2, cb3]|
                        (Car1[cs, cb1] ||| Car2[cs, cb2] ||| Car3[cs, cb3])

AECar_SC :=         Report_cb[cb, cb_rep]
                        |[cb_rep]|
                        Control[msg_rec, cb_rep]
                        |[msg_rec]|
                        Rec_msg[msg, msg_rec]

AEControl :=         Mailbox_Cont[cb_rep, msg_rec, int]
                        |[int]|
                        Engine_Cont[int]

AECoordination :=   Report_fb[fb, fb_rep]
                        |[fb_rep]|
                        Dispatch[msg, fb_rep, new_cs]
                        |[new_cs]|
                        Change_cs[cs, new_cs]

AEDispatch :=       Mailbox_Dispatch[fb_rep, new_cs, int]
                        |[int]|
                        Engine_Dispatch[int, msg]

```

8.2.2 Architecture Refactoring

Refactoring of architectures constitutes an important aspect of distributed system design. For example, refactoring may be used to explore different architectures at early stages of the design process. It may also be used later in the design process to modify an existing architecture to match implementation requirements or performance criteria. The goal of architecture refactoring is to transform an existing architecture into a different equivalent one. Therefore, architecture refactoring involves component (process in LARGs) grouping and ungrouping.

Since grouping and ungrouping algorithms have been defined on LARGs, these may be used to refactor architectures. For example, figure 6.8 and figure 6.9 illustrates such architecture refactoring.

However, the LAEG method only allows architecture refactoring in the LARG model. Thus, in order to use the LAEG method for architecture refactoring in other description models, we need to define inverse architectural mapping between the LARG model and other description models. This is illustrated in figure 8.3. This figure illustrates how LOTOS can be used to support architecture refactoring in a design model X. In order to apply refactoring R on an architecture A given in a description model X, we need to conduct the three following phases:

- 1°) map architecture A on its corresponding LARG P using an AM function,
- 2°) apply LARG refactoring R' on LARG P to obtain LARG P', where R' is the refactoring in the LARG domain which corresponds to refactoring R in the description model X domain, and
- 3°) use inverse mapping to derive architecture A' from LARG P'.

Therefore in order to allow the use of the LAEG method to support architecture refactoring in description models, since we already defined solutions for phases 1° and 2°, we only need to find a solution for phase 3°. Unfortunately, the definition of inverse mapping between the LARG model and description models does not have simple solution for every type of description models. Such a solution seems easy to define in the case of EGL

because the LARG model and EGL use similar architectural artifacts, but a lot more difficult to define in the case of MachineCharts, because the mapping between types of MachineCharts architectural artifacts and LARG artifacts is many-to-one, and thus the definition of an inverse mapping function is impossible. However, we think that some solutions can be found to allow the support of architecture refactoring with the LAEG method in description models such as MachineCharts.

The definition of particular solutions for inverse mapping still constitutes a topic for future research.

Figure 8.3 Using the LAEG method for architecture refactoring

Case Study: The Elevator System

Refactoring of the elevator system

In the case of the elevator system, for implementation concerns related to the physical location of processes in the real (implemented) system, we may want to transform the former elevator system LARG (figure CS6.1) in such a way that every Car_i process and Car_i_SC process are grouped together. For this purpose, we use both the UnGrouping and the Grouping algorithms to obtain the LARG figure CS8.1 from the former LARG (figure CS6.1).

8.2.3 LOTOS Correctness Preserving Transformation

A modified version of the LAEG method may also be used as a LOTOS architectural CPT (Correctness Preserving Transformation), i.e. CPT that can be applied to LOTOS AEs (LOTOS expressions in which processes are composed by means of parallel operators). In this case, the problem is the following: we have a LOTOS architectural expression E_{init} that we want to transform into a different LOTOS architectural expression E_{final} , where only the skeleton of E_{final} is given, i.e. a LOTOS architectural expression in which the composition of the GSs contained in the different parallel operators is left undefined. The processes of both architectural expressions are the same but ordered and grouped differently. Then, the goal of the CPT consist in finding all the GSs on which the different processes and groupings will interact in E_{final} . To illustrate this problem, we may consider the following example:

$$E_{init} = (P1[a, c] \parallel [a] \parallel P2[a, d]) \parallel [c, d] \parallel (P3[b, c] \parallel [b] \parallel P4[b, d])$$

$$E_{final} = (P1[a, c] \parallel [?] \parallel P3[b, c]) \parallel [?] \parallel (P2[a, d] \parallel [?] \parallel P4[b, d])$$

CPT Method

In order to use the LAEG method as a LOTOS CPT, we first need to generate the LARG P_{init} associated with E_{init} . The second step consists in flattening the LARG. The flattening method consist in successive application of the UnGrouping algorithm, the algorithm is applied until there exist no more groupings in the LARG. The result of the flattening is an ungrouped LARG P_{UnGr} which is equivalent to P_{init} . Then, we conduct the analysis phase on the LARG P_{UnGr} . In relation with the analysis method defined in §8.1 (for which an Analysis procedure have been defined in §8.1.1), the analysis phase conducted here integrally includes the two first phases of the analysis method defined in §8.1.1, i.e. detection and correction of architectural errors. Finally, we apply the grouping sequence impose by E_{final} (E_{final} imposes the grouping sequence that must be applied on the ungrouped LARG to obtain P_{final}). If the sequence is correct with respect to the interaction semantic of the ungrouped LARG, it transforms the ungrouped LARG into a binary grouped LARG P_{final} which is equivalent to P_{init} (E_{init}). If the grouping sequence is not correct, then there is no possible transformation between E_{init} and E_{final} .

Since in deterministic LARG any grouping is possible and semantically correct, if the ungrouped LARG associated with E_{init} is deterministic, then any transformation on E_{init} constitutes a CPT. However, if the ungrouped LARG associated with E_{init} is non-deterministic, CPT may not exist, and in this case the application of the grouping sequence may result in an error.

Hereafter, an algorithm is given for the CPT method.

CPT Algorithm**begin** Generate_LARG_from_LOTOSAE(E_{init}, P_{init}); Flatten(P_{init}, PUnGr);

Analyse(PUnGr);

 Apply_Grouping_Seq(PUnGr, P_{final}, error); **If not**(error) **then** Generate_LOTOSAE_from_LARG(P_{final}, E_{final}) **else** {there is no possible transformation between E_{init} and E_{final}};**end.****Example of Application of the CPT method**

As an example, we apply the CPT method to the two LOTOS AEs previously given.

E_{init} = (P1[a, c] || a || P2[a, d]) || c, d || (P3[b, c] || b || P4[b, d]), and

E_{final} = (P1[a, c] || ? || P3[b, c]) || ? || (P2[a, d] || ? || P4[b, d])

The result of the application of the CPT method with the two previous LOTOS AEs is illustrated in figure 8.4.

$$(P1[a, c] \parallel [a] P2[a, d]) \parallel [c, d] (P3[b, c] \parallel [b] P4[b, d])$$

↓ Generate_LARG_from LOTOSAE

↓ Flattening(P_{init})

↓ Grouping($P_{unGr}, ((P1, P3), (P2, P4))$)

↓ Generate_LOTOSAE_from_LARG

$$(P1[a, c] \parallel [c] P3[b, c]) \parallel [a, b] (P2[a, d] \parallel [d] P4[b, d])$$

Figure 8.4 Application of the CPT method

Chapter 9 Conclusions and Future Work

In this chapter, we briefly summarize the content of the thesis, highlight its main contributions, and draw up a list of topics for future research.

9.1 Thesis Summary

In the first part of this thesis, which includes chapters 2, 3 and 4, we discussed the definition of a conceptual framework for formal method integration in the distributed system design process. For this purpose, we first conducted an analysis of the distributed system design process and defined the context of the thesis. Then, in §3 and §4, we took steps towards the definition of the conceptual framework by defining the concept of generic formal interpretation method for design. This concept of generic formal interpretation method have also been applied for the definition of a generic LOTOS interpretation method. The analysis of generic formal interpretation methods in relation with LOTOS has shown the need for the definition of a LOTOS architectural interpretation method. For this purpose, we, in §5, proposed to introduce an intermediate model (the LARG model) between architectural designs and LOTOS architectural expressions. This has resulted in the decomposition of the LOTOS architectural method in two submethods: an AM method and the LAEG method. However, while the LAEG method constitutes an important part of this thesis, AM methods are not defined in this thesis; their definition still constitutes a topic for future research. Finally, we, in §5, analysed the requirements for the definition of LARG model.

In the second part of the thesis, which includes chapters 6, 7 and 8, we concentrated on the definition of a generic architectural interpretation method for LOTOS. We first defined the LARG model (§6) which serves as an intermediate model between architectural designs and LOTOS architectural expressions. Then, in §7, we discussed and defined solutions for two

important issues in relation with the generation of LOTOS architectural expressions from LARGs: design errors, and deterministic/non-deterministic LARG groupings. Finally, we defined the LAEG method (§8) which allows the generation of LOTOS architectural expressions from LARGs. In addition, different possible applications of the LAEG method have been discussed in §8.

Finally, the elevator system problem has been integrated all along the thesis as a case study to relate the different methods defined to concrete distributed system design. The development of the elevator system problem has also allowed to show that the LAEG method can be used for the generation of LOTOS architectural expressions from concrete designs.

9.2 Contributions

Conceptual framework for the integration of formal methods in the distributed system design process

The integration of FDTs in the distributed system design process would enable one to conduct formal verification on system designs. In this thesis, we took steps towards such an integration by setting the stage for the definition of a general framework for the integration of FDTs in the distributed system design process. The framework is based on formal interpretation methods. Such an approach offers the advantage to lead to a soft integration of FDTs in the design process.

A generic LOTOS interpretation method

A generic LOTOS interpretation method has been discussed. For this purpose, we defined both a "composition of total behavior" method and a generic LOTOS architectural interpretation method. The generic LOTOS architectural interpretation method is based on the LARG model and the LAEG method. In order to use the generic LOTOS architectural interpretation method with a given description model, one only has to provide an AM method that allows the mapping of the description model on the LARG model.

Also, this generic LOTOS interpretation method has been used to generate a LOTOS specification from the elevator system MachineCharts design. However, the resulting LOTOS specification remained incomplete in this thesis. In order to generate a complete LOTOS specification, we would need to integrate a LOTOS behavioral interpretation method in the LOTOS interpretation method. As mentioned, such a method already exists for the LOTOS interpretation of state machines.

The LARG model

The LARG model has been developed, within the context of this research project, to serve as the intermediate model in the LOTOS architectural interpretation method. In order to allow the interpretation of the greatest possible number of description models, the LARG model has been developed to be as abstract as possible. Thus, the LARG model possesses only one type of architectural component, called process, and one type of interaction which is rendez-vous interaction. The LARG model allows the representation of N-way interactions (for $N \geq 1$) and possesses a high level interaction operator that corresponds to the LOTOS parallel operator. Also, for the purpose of LAEG, both a Grouping algorithm, and an UnGrouping algorithm have been defined on LARGs.

LARG Analysis

LARG analysis (§7) provides the basis for the definition of the LAEG method. In this thesis, we defined analytical methods for identifying both design errors and non-determinism. In relation with design errors, we suggested appropriate corrections. In relation with non-determinism, we defined different grouping techniques that can be used to remove it in LARGs. However, in cases where the use of these techniques is not sufficient

to remove non-determinism, we conclude that it is not possible to remove it, and the generation of LOTOS architectural expressions is not possible. For this purpose, we proposed the definition of a general interaction operator (Op in §7.2.3) that can be used to describe any topology of interacting processes. Also, we described the semantic of this general interaction operator.

The LAEG method

We defined the LAEG method to allow the generation LOTOS AEs from LARGs. The contribution of the method itself is significant since:

- it provides a validation framework for architectural designs,
- it allows the formal interpretation of non-deterministic architectures, and
- it allows the generation of the complete set of equivalent LOTOS AEs for a given LARG.

As discussed in §8.2, the LAEG method may have other applications such as architecture refactoring and LOTOS architectural CPT. In relation with architecture refactoring, the definition of inverse mapping methods still need to be defined before the LAEG method can be used to support architecture refactoring in description models.

9.3 Future Research

- Implement of the LAEG method in a tool (e.g. EGL)
- Define partitioning criteria which will enable the choice of one particular expression in the set of all possible LOTOS expression for a given LARG (e.g. metrics for LOTOS architectural expressions)
- Define architectural mapping functions between different design models and the LARG model
- Define inverse mapping functions, or methods, to allow the use of the LAEG method to support architecture refactoring in description models
- Apply the conceptual framework define in this thesis to formalize the Timethread design process defined in [Buh92]

List of References

- [Boc89] G.v. Bochmann, O. Bellal. "Test Result Analysis in Respect to Formal Specifications". *Proceedings of 2nd Int. Workshop on Protocol Test Systems*. Berlin, Oct. 1989, pp.272-294.
- [Boc90] G. v. Bochmann, Q. Gao, C. Wu. "Distributed Implementation of LOTOS". S.T. Vuong (ed.) *Formal Description Techniques, II*. P: 133-146, North-Holland, 1990.
- [Bol89a] T. Bolognesi, E. Brinksma. "Introduction to the ISO Specification Language LOTOS". P.H.J. van Eijk, C.A. Vissers, and M. Diaz (eds.). *The Formal Description Technique LOTOS*, P: 23-73. North-Holland, 1989.
- [Bol89b] T. Bolognesi, M. Caneve. "Equivalence Verification: Theory, Algorithms and a Tool". P.H.J. van Eijk, C.A. Vissers, and M. Diaz (eds.). *The Formal Description Technique LOTOS*. P: 303-326, North-Holland, 1989.
- [Bol90] T. Bolognesi. "A Graphical Composition Theorem for Networks of LOTOS processes". *Proceedings of the 10th International Conference on Distributed Computing Systems*, IEEE Computer Society Press, 1990.
- [Bou91] R. Boumezbeur, L. Logrippo. *Design, Specification and Validation of Telephony Systems in LOTOS*, Master Thesis TR-91-30, Departement of Computer Science, University of Ottawa, Canada, 1991.
- [Bri91] E. Brinska. "What is the Method in Formal Methods?". *Proceeding of FORTE'91, Fourth International Conference on: FORMAL DESCRIPTION TECHNIQUES*, P: 31-52, Sydney, 1991
- [Buh90] R.J.A. Buhr, *Practical Visual Techniques in System Design (with Applications to Ada)*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

- [Buh92a] R.J.A. Buhr, G.M. Karam, C.M. Woodside, R. Casselman, G. Frank, H. Scott and Don Bailey. "TimeBench: A CAD tool for real-time system design". *Proceeding of the 2nd International Symposium on Environments and Tools for Ada (SETA92)*. Washington D.C., USA, 1992.
- [Buh92b] R.J.A. Buhr, R.S. Casselman, "Architectures with Pictures", In *Proceedings of OOPSLA'92*, P: 466-483, ACM/SIGPLAN, Vancouver, Canada, 1992.
- [Buh93] R.J.A. Buhr. *Designing with Timethreads for Systems with Distributed Control: A one-day introduction to the essence*. Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1991.
- [Buh93b] R.J.A. Buhr. *Pictures that play: Design Notations for real time & distributed systems*. Technical Report SCE-93-04, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1993.
- [Che90] T.Y. Cheung, Y.C. Ye, X. Ye, C.Q. Wang. "UO-GLOTOS: A Syntax/System for Representing". Editing, and Translating Graphical LOTOS. S.T. Vuong (ed.) *Formal Description Techniques, II*. P: 31-41, North-Holland, 1990.
- [CPT92] *Catalogue of LOTOS Correctness Preserving Transformations*. Lo/WP1/T1.2/N0045/V03, T. Bolognesi (Ed), Lotosphere Project (ESPRIT 2304), 1992.
- [Eij90] P. van Eijk, H. Kremer, M. van Sinderen. "On the use of specification styles for automated protocol implementation from LOTOS to C". L. Logrippo, R.L. Probert, H. Ural (eds.) *Protocol Specification, Testing, and Verification, X*. P: 153-164, North-Holland, 1990.
- [Fac89] M. Faci, L. Logrippo and B. Stéprien. *Formal Specifications of Telephone Systems in LOTOS*. Technical Report TR-89-07, Computer Science Department, University of Ottawa, Ottawa, 1989.
- [Fan90] A. Fantechi, S. Gnesi, G. Ristori. "Compositional logic semantics for LOTOS". L. Logrippo, R.L. Probert, H. Ural (eds.) *Protocol Specification, Testing, and Verification, X*. North-Holland, 1990.

-
- [Gal91a] S. Gallouzi, L. Logrippo, A. Obaid. "A Hoare style proof system for LOTOS". J. Quemada, J. Manas, E. Vazquez (eds.) *Formal Description Techniques, III*. North-Holland, 1991.
- [Gal91b] S. Gallouzi, L. Logrippo, A. Obaid. "An Expressive Trace Theory for LOTOS". B. Pehrson, B. Jonsson, J. Parrow (eds.) *Protocol Specification, Testing, and Verification, XI*. North-Holland.
- [Gar90] H. Garavel, J. Sifakis. "Compilation and Verification of LOTOS Specifications". L. Logrippo, R.L. Probert, H. Ural (eds.) *Protocol Specification, Testing, and Verification, X*. P: 379-394, North-Holland, 1990.
- [Gol91] P. Goldstein, M. Vigder and R.J.A. Buhr. *Case Study in LOTOS*. Technical Report SCE-91-42, Department of Systems and Computer Engineering, Carleton University, Ottawa, 1991.
- [Gui88] R. Guillemot, M. Haj-Hussein, L. Logrippo. "Executing Large LOTOS Specifications". S. Aggarwal, K. Sabnani (eds.) *Protocol Specification, Testing, and Verification, VIII*. P: 399-410, North-Holland, 1989.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [ISO89] International Organisation for Standardisation. *Information Processing Systems - Open Systems Interconnection - LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. ISO 8807, TC97/SC21, 1989.
- [ISO89b] ISO/IEC JTC1/SC21 N3253. G-LOTOS: A Graphical Syntax for LOTOS (1989).
- [Kar90] G. Karjoth. "A Compilation of Algebraic Processes Based on Extended-Action Derivation". *Proceeding of FORTE 90 : Third International Conference on Formal Description Techniques*, P: 125-138, Madrid, Spain, 1990.
- [Kar88] G. Karjoth. "Implementing Process Algebra Specifications by State Machines". *Protocol Specification, Testing and Verification VIII*. P: 47-60, North-Holland, 1988.

- [Lam91] S. Lamouret, M. Vigder, R.J.A. Buhr. *Semi-Automated Translation of State Machine into LOTOS Specifications and Processes*. Technical Report SCE-91-34, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1991.
- [Lam92] S. Lamouret. *Study of the links between Telos and LOTOS in relation with TimeThreads*. To be published as a Technical Report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1992.
- [Led90] G. Leduc. *On the role of Implementation Relations in the Design of Distributed Systems using LOTOS*. Faculté des Sciences Appliquées, Université de Liège, Liège, 1990.
- [Log90] L. Logrippo. "The University of Ottawa LOTOS Toolkit". J. Quemada, J. Manas, E. Vazquez (eds.) *Formal Description Techniques, III*. North-Holland, 1991.
- [Log91] L. Logrippo, M. Faci and M. Haj-Hussein. *An Introduction to LOTOS: Learning by Examples*, Technical Reports TR-91-21, Departement of Computer Science, University of Ottawa, Ottawa, 1991.
- [Log92] L. Logrippo, M. Faci, M. Haj-Hussein. *An Introduction to LOTOS: Learning by Examples*. Technical Report TR-91-21, Departement of Computer Science, University of Ottawa, Ottawa, Canada, 1991.
- [Man89] J.A. Manas, T. de Miguel, H. van Thienen. "The Implementation of a Specification Language for OSI Systems". P.H.J. van Eijk, C.A. Vissers, and M. Diaz (eds.). *The Formal Description Technique LOTOS*. P: 409-421, North-Holland, 1989.
- [Mar88] S. Marchena, G. Leon. *Transformation from LOTOS Specs to Galileo Nets*. L. Logrippo, R.L. Probert, H. Ural (eds.) *Protocol Specification, Testing, and Verification, X*. P: 217-230, Ottawa, Canada, 1990.
- [Mil89] R. Milner. *Communication and concurrency*. Prentice-Hall International Series in Computer Science, 1989.

- [Mil82] R. Milner. "A Complete Inference System for a class of Regular Behaviours". *Journal of Computer and System Sciences* 28, 1984.
- [Nom90] S. Nomura, T. Hasegawa, T. Takizuka. "A LOTOS Compiler and Process Synchronization Manager". L. Logrippo, R.L. Probert, H. Ural (eds.) *Protocol Specification, Testing, and Verification, X*. P: 165-184, North-Holland, 1990.
- [Oba90] A. Obaid, G. Guével and R. Saumure. *EGL: Un système de conception des protocoles de communication*. Rapport de Recherche RR 90/10-01, Université du Québec à Hull, Hull, Canada, 1990.
- [Ohm90] K. Ohmaki, K. Takahashi, K. Futatsugi. "A LOTOS Simulator in OBJ". J. Quemada, J. Manas, E. Vazquez (eds.) *Formal Description Techniques, III*. North-Holland, 1991.
- [Pir90] L.F. Pires and W.L. de Souza. *Step-wise Refinement Design Example Using LOTOS*. unknown, 1990.
- [Que88] J. Quemada, S. Pavon, A. Fernandez. "Transforming LOTOS specifications with LOLA, The Parametrized Expansion". K.J. Turner (ed.) *Formal Description Techniques*. P: 45-54, North-Holland, 1989.
- [Sel92] B. Selic, G. Gullekson, J. McGee, I. Engelberg, "ROOM: An Object Oriented Methodology for Developing Real-Time Systems", *Proceedings of CASE'92, Fifth International Workshop on Computer-Aided Software Engineering*. P: 230-240, 1992.
- [Sho92] J. Schot. *The role of Architectural Semantics in the formal approach of Distributed Systems Design*. Ph. D. Thesis, University of Twente, The Netherlands, 1990.
- [Tur93] K.J. Turner. "An Engineering Approach to Formal Methods". A. Danthine, G. Leduc, P. Wolper (eds.) *Protocol Specification, Testing, and Verification, VII*. P: 165-184, North-Holland, 1990.

-
- [Vig91] M. Vigder and R.J.A. Buhr. *Using LOTOS in a Design Environment..* Technical Report SCE-91-28, Department of Systems and Computer Engineering, Carleton University, Ottawa, 1991. (also in *Proceeding of FORTE'91, Fourth International Conference on: FORMAL DESCRIPTION TECHNIQUES*, P: 1-14, Sydney, 1991)
- [Vig92] M. Vigder. *Integrating Formal Techniques into the Design of Concurrent Systems*, Ph.D. Thesis OCIEE-92-03, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1992.
- [Vis89] C.A. Vissers, G. Scollo and M. van Sinderen. "Specification Styles in Distributed Systems Design and Verification". *Theoretical Computer Science* 89. Elsevier, 1991, p: 179-206.
- [Vis90] C.A Vissers, G. Scollo, M. van Sinderen and E. Briksma. *On the use of Specification Styles in the Design of Distributed Systems*. Unpublished manuscript, 1990.

Appendix A Description of the elevator system problem

A computer-controlled elevator system with N cars and M floors is to be installed in a building. The problem is to design the architecture of the strategic control software, allowing for the possibility that the hardware architecture may be either a single computer, or a computer for each car plus a central computer. The regime of operation of concern to this problem begins with all cars idle on some arbitrary initial floor; during this regime, cars are either in motion, visiting a floor, or idle at the last floor visited.

The elevator system outside the computer consists of cars, elevator shafts, hoist cables and winches, motors to drive the winches, doors on both the shafts at each floor and the cars, electromechanical controls, and sensors to give information on elevator position and movement, and buttons to request service. Each car has M buttons, 1 for each floor, to request a car to visit that floor. Each floor has 2 buttons (except bottom and top, which only have one each), one to request a car going up and one to request a car going down.

Strategic vs. Tactical Control

Figure CS1.1 Factoring of the Elevator system problem

Strategic Control

- The strategic control software to be designed monitors the status of cars and of requests for cars to visit floors, asks idle cars to start moving up or down, and asks moving cars to stop at floors. We distinguish the strategic control software architecture from the strategic control algorithm. The algorithm determines the order of servicing requests, e.g. to ensure that all requests for cars from floors must be serviced eventually, with all floors given equal priority. The architecture is only concerned with the algorithm to the extent that it is able to provide a framework that can support any algorithm. The algorithm is outside the scope of this problem.
- Strategic control is also concerned with other matters outside our scope, such as taking cars temporarily out of service.

Tactical Control

- Outside the scope of this problem is design of tactical control, which is concerned with fine details like interpreting raw input from sensors (buttons, car position sensors, etc.), controlling lights associated with buttons, opening elevator doors in a safe manner as a car

is approaches and stops at a floor, keeping doors open for a suitable period, setting the direction of rotation of motors for cable winches, and so forth. The particular mix of low-level software and traditional electromechanical controls needed to achieve tactical control is not our concern.

The details of tactical control are assumed to be hidden inside $N+1$ black boxes, one for each car and one for the building as a whole. The black box for a particular car encapsulates tactical control aspects that are uniquely associated with particular cars, such as reading raw input from car buttons and issuing motor commands. The black box for the building as a whole encapsulates tactical control aspects that are not uniquely associated with particular cars, such as reading raw input from floor buttons. From the perspective of the strategic control software, these black boxes are the elevator system. "Outputs" from the strategic control software are actually commands to these black boxes. "Inputs" to the strategic control software are actually commands to it from these black boxes. How these commands are realized is an open issue.

Figure CS1.2 Elevator Problem: Strategic Control Outputs & Inputs

Details of Strategic Control Outputs & Inputs

ServeFloor(floor):

This output, issued to the tactical black box for a particular car, causes the car to service the designated floor. Floor service consists of stopping at a floor, opening the doors, holding them open for a period, and then closing them automatically. The ServeFloor output is ignored unless the car is moving between floors with the designated floor as the next floor and is within the stopping window (see Inputs), or is already stopped at the designated floor. Thus the output can only be used at the last possible moment to ask a car to serve a floor; it cannot be used in advance to queue service requests. In the case where the car is already at the designated floor, the output simply causes the doors to open and close as above.

Move (direction):

This output, issued to the tactical black box for a particular car, causes a stationary car to move up or down. The car will continue to move until a ServeFloor output is issued. The Move output is ignored if the elevator is moving or if it can go no further in the designated direction. There is no Stop output, because stopping is folded into the ServeFloor output.

FloorButton (floor, direction):

This input comes from the tactical black box for the building as a whole. It provides the information that service in the particular direction is requested from a particular floor.

CarButton (car, floor):

This input comes from the tactical black box for a particular car. It provides the information that service to a particular floor is requested by a person inside a particular car.

MovingReady (car, direction, floor):

This input comes from the tactical black box for a particular car. It is derived from sensors above and below each floor in each shaft that indicate a car moving in a particular direction is within stopping range of a particular floor. There is a time window after this input occurs during which a ServeFloor output can be acted upon for that car and floor; otherwise the car keeps moving. The strategic control software uses this input to trigger decisions on floor service.

StoppedReady (floor):

This input comes from the tactical black box for a particular car. It indicates that the car is stopped idle on the floor. The strategic control software uses this input to trigger decisions on whether cars should remain where they are or move elsewhere.

END

1 9-0 7-9 4

FIN