UNU/IIST's Software Technology R&D in Africa, Asia, Eastern Europe, and Latin America

'Five' Years — A Personal View

Dines Bjørner

With Special Appendices by Colleagues

1992–1997 — Printed August 21, 1997

UNU/IIST Report No. 90 — IT/DTU Aug.97

UNU/IIST

UNU/IIST enables developing countries to attain self-reliance in software technology by: (i) their own development of high integrity computing systems, (ii) highest level post-graduate university teaching, (iii) international level research, and, through the above, (iv) use of as sophisticated software as reasonable.

UNU/IIST contributes through: (a) advanced, joint industry–university **advanced development projects** in which rigorous techniques supported by semantics-based tools are applied in case studies to software systems development, (b) own and joint university and academy institute **research** in which new techniques for *(1) application domain* and computing platform modelling, *(2) requirements capture,* and *(3) software design & programming* are being investigated, (c) advanced, post-graduate and post-doctoral level **courses** which typically teach **Design Calculi** oriented software development techniques, (d) **events** [panels, task forces, workshops and symposia], and (e) **dissemination**.

Application-wise, the **advanced development projects** presently focus on software to support large-scale infrastructure systems such as transport systems (railways, airlines, air traffic, etc.), manufacturing industries, public administration, telecommunications, etc., and are thus aligned with UN and International Aid System concerns. UNU/IIST is a leading software technology centre in the area of **infrastructure software development**.

UNU/IIST is also a leading research centre in the area of **Duration Calculi**, i.e. techniques applicable to *real-time, reactive, hybrid & safety critical systems.* The **research** projects parallel and support the advanced development projects.

At present, the technical focus of UNU/IIST in all of the above is on applying, teaching, researching, and disseminating **Design Calculi** oriented techniques and tools for trustworthy software development. UNU/IIST currently emphasises techniques that permit proper development steps and interfaces. UNU/IIST also endeavours to promulgate sound project and product management principles.

UNU/IIST's primary dissemination strategy is to act as a clearing house for reports from research and technology centres in industrial countries to industries and academic institutions in developing countries. At present more than 200 institutions worldwide contribute to UNU/IIST's report collection while UNU/IIST at the same time subscribes to more than 125 international scientific and technical journals. Information on reports received (and produced) and on journal articles is to be disseminated regularly to developing country centres — which are then free to order a reasonable number of report and article copies from UNU/IIST.

Dines Bjørner, Director — 2.7.1992–1.7.1997

UNU/IIST Reports are either $\mathcal{R}$esearch, $\mathcal{T}$echnical, $\mathcal{C}$ompendia or $\mathcal{A}$dministrative reports:

$\boxed{\mathcal{R}}$ Research Report • $\boxed{\mathcal{T}}$ Technical Report • $\boxed{\mathcal{C}}$ Compendium • $\boxed{\mathcal{A}}$ Administrative Report

P.O. Box 3058
Macau

UNU/IIST's Software Technology R&D in Africa, Asia, Eastern Europe, and Latin America

'Five' Years — A Personal View

Dines Bjørner

With Special Appendices by Colleagues

Abstract

UNU/IIST conducts joint R&D with more than 30 groups worldwide. We review this research (into real-time & hybrid systems, telecommunications and computing platform & systems integration). We also review the advanced software development projects. These latter center around software support for large-scale infrastructure systems. Throughout we review research into and application (methods) of design calculi (ie. formal methods).

The report represents the authors' rather personal view on the matter!

Dines Bjørner (DB) is (was) the first & founding Director of UNU/IIST and served UNU/IIST from 2. July 1992 to 1. July 1997 while on an extended five year leave of absence from his chair at the Department of Information Technology, The Technical University of Denmark (TUD), Bldg. 345, DK–2800 Lyngby, Denmark (E-Mail: db@it.dtu.dk; Fax: +45-45-88.45.30). DB is one of the original developers of VDM, while at the IBM Vienna Laboratory, 1973–1975, and later at TUD. DB co-founded, in 1979, Dansk Datamatik Centre (DDC, 1979–1989) which, under his initial leadership, developed formal semantics of and industrial strength compilers for CHILL and Ada. Later DB instigated the RAISE Project. Before all this DB was with IBM (1962–1975), at their Nordic Laboratories (Stockholm, Sweden), Research & Development Laboratories in San Jose (Calif., USA), and Vienna (Austria) — where he worked for and with Cai Kinberg, Gene Amdahl, John W. Backus, E. F. (Ted) Codd, Peter Lucas, the late Hans Bekić, Cliff Jones and many others. DB was a co-founder of VDM Europe and, as its chairman "converted" VDM Europe into FME: Formal Methods Europe.

Extended Abstract:

We outline projects of advanced development of infrastructure software systems and research into real-time and telecommunications systems. These R&D projects are pursued in a worldwide collaboration between UNU/IIST and a long list of academic and public and private sector software centres in developing countries on four continents. UNU/IIST is the UN Systems' only R&D + post-graduate centre for software technology,

We emphasize the methods used in research and in formal, verifiable software development: (i) the Duration Calculi for the design of (ii) *Real-time and Hybrid Systems*, (iii) formal versions of SDL for the design of (iv) *Telecommunications Systems*, and (v) RAISE: Rigorous Approach to Industrial Software Engineering for the research & exploratory development of (vi) *Software Support for Infrastructure Systems*.

We survey two main axes of methods for the development of software: (vi) the "trinity" stages of Domain Theory & Analysis (as a novel means for thoroughly understanding the universe of discourse of the application problem), Requirements Capture & Definition (as the means for describing precisely what properties are expected of the software), "followed" by Software Design; and (vii) the mapping of software development approaches onto specialized techniques and implementation tools for Problem Frames [62], such as *Translation, Control, Information, Workpiece, Transaction Clearing, Connection,* and other *Frames*.

We finally exemplify a large number of R&D projects. These span from research oriented studies to advanced, prototype software developments. The latter aim at such infrastructure application areas as: railways, air traffic, metropolitan transport, manufacturing industries, multi-lingual text processing, ministry of finance taxation, budget & treasury systems, external debt monitoring & control systems, financial services & industries (banks, securities, insurance, clearing, etc.), decision support systems for sustainable development, and enterprise systems for strategic, tactical & operation resource management.

The style of this presentation is rather relaxed.

With Special Appendices by Colleagues

Contents

1 Introduction **2**

1.1 "Case Story Examples" . . . 2

1.1.1 A Domain Example . . . 2

1.1.2 A Requirements Example . . . 2

1.1.3 A Software Development Example . . . 3

1.1.4 A 'Trilogy' of Development Stages . . . 4

1.2 Advanced Development . . . 4

1.2.1 Software Quality . . . 4

1.2.2 Infrastructures: Meeting the Needs of a Developing World . . . 5

1.2.3 Scalability . . . 5

1.3 Foundational Research . . . 6

1.3.1 A Discourse on 'Science & Technology' . . . 7

1.3.2 Duration Calculi — UNU/IIST, a World Centre of Research . . . 8

1.3.3 Advanced Infrastructure Development . . . 8

1.4 Technologies and Hypes . . . 8

1.4.1 On Object-Orientedness . . . 9

1.4.2 Hypes . . . 9

1.5 About UNU/IIST . . . 9

1.5.1 Charter . . . 9

1.5.2 The UNU/IIST Agenda . . . 10

1.5.3 More on UNU/IIST . . . 11

2 UNU/IIST's Design Calculi — Formal Methods **12**

3 UNU/IIST's R&D Projects **14**

3.1 The Applied Research Projects . . . 14

3.2 The Advanced Development Projects . . . 15

3.2.1 The "Trinity" of Development Stages . . . 15

3.2.2 Project Enumeration . . . 16

3.2.3 Application "Type Theories" . . . 20

4 The Overall Development Methods **22**

4.1 Methodological Concerns . . . 22

4.1.1 Stage & Stepwise Unfolding — The "Trinity" Revisited . . . 22

4.1.2 Model Experiments . . . 23

4.1.3 Problem Frame Foca . . . 25

4.2 Interplay between Informality & Formality . . . 26

5 Review & Evaluation **28**

5.1 Science & Technology: Some Comments . . . 28

5.2 International Collaboration: Some Comments . . . 30

5.3 UNU/IIST Now & in the Next Future . . . 32

5.3.1 UNU/IIST Strengths . . . 32

5.3.2 UNU/IIST Weaknesses . . . 34

6 Acknowledgements **38**
6.1 Scientific Staff 38
6.2 Fellows 38
6.3 Visitors 39
6.4 The UNU/IIST Board 39
6.5 Macau and Initial Donors 39

A Bibliographical Notes **41**

B About UNU/IIST **47**
B.1 The UNU System 47
B.2 Funds & Budget, Staff & Fellows 47
B.3 The UNU/IIST Charter 47
B.4 UNU/IIST's Programmatic Activities 48
B.5 The UNU/IIST Network 49

C The Design Calculi & Formal Methods **50**
C.1 DC: The Duration Calculi 50
Zhou Chao Chen, Dang Van Hung and Xu Qi Wen
C.2 SDL: Specification and Description Language 59
Cornelis Adam (Kees) Middelburg
C.3 RAISE: A Formal Software Engineering Method 63
Chris George
C.4 LSD: Logical Systems Development 66
C.4.1 Overview 66
C.4.2 VDM & RAISE 67
C.4.3 Stage & Stepwise Development 68
C.4.4 Michael Jackson's Problem Frames 70
C.4.5 Informality & Formality 87
C.5 Infrastructures — A Research Topic 88
C.5.1 Transport Infrastructures 88
C.5.2 Public Administration Infrastructures 89
C.5.3 An Infrastructure Research Agenda 89

D UNU/IIST's Research Projects **90**
D.1 DeTFoRS: Real-time & Hybrid Systems 90
Zhou Chao Chen and Xu Qi Wen
D.2 DesCaRTeS: Telecommunications Systems 95
Cornelis Adam (Kees) Middelburg
D.3 CaSIno: Platforms for Systems Implementation & Integration 99
Tomasz Janowski

E UNU/IIST's Software Technology Projects **101**
E.0.1 Models to be shown 101
E.0.2 Comments on the Models to be Developed 101
E.0.3 On "*Why are we showing You these Models?*" 102

E.1 RaCoSy: Railway Computing Systems 103
E.1.1 Railway System Scope 103
E.1.2 Elements of a Domain Theory 104
E.1.3 Requirements: A "Running Map" System &c. 106
E.1.4 Software: A "Running Map" System &c. 106
E.1.5 Discussion 107
E.2 ATC'2000: Air Traffic 108
E.2.1 Air Space 108
E.2.2 Timetables 109
E.2.3 Traffic 110
E.2.4 Scheduling 111
E.2.5 Discussion 112
E.3 MiTraS: Minsk Metropolitan Transport System 115
E.3.1 Towards A Domain Theory 115
E.3.2 Towards Requirements 117
E.3.3 Towards An Information Technology 120
E.3.4 Discussion 121
E.4 General Transportation Systems 122
E.4.1 A Graph Model of Transport 122
E.4.2 A Stepwise Unfolded — Can/Cannot Do Transport Model 130
Tomasz Janowski
E.4.3 Towards A Theory of Transport Systems 134
Nikolai S. Nikitchenko
E.5 MI²CI: A Manufacturing Industry Systems 141
Tomasz Janowski
E.5.1 Introduction 141
E.5.2 Domain Modelling and Analysis, An Example 142
E.5.3 Conclusions 146
E.6 MultiScript: Multi-Lingual Script System 148
Richard Moore and Myatav Erdenechimeg
E.6.1 Basic Concepts 148
E.6.2 Background 149
E.6.3 A Domain Theory 150
E.6.4 A Formal Model of Multi-lingual Documents 152
E.6.5 On Requirements 153
E.6.6 On Software 154
E.6.7 Discussion 154
E.7 ABC'2000: 155
Richard Moore
E.7.1 Airline Business: Informal Analysis 155
E.7.2 Operational View: Airline Resources 155
E.7.3 Operational View: Airline Products 157
E.7.4 A Formal Model of Aircraft and Flights 157
E.7.5 Discussion 159
E.8 DiMulTS: A Philippine Radio Telephone System 160

Chris George
E.8.1 Introduction 160
E.8.2 Initial specification 161
E.8.3 First development step 162
E.8.4 Developing the network 163
E.8.5 Introducing distribution 163
E.8.6 Verification 164
E.8.7 Conclusion 164
E.9 MoFIT: Vietnam Ministry of Finance IT 165
Chris George
E.9.1 Introduction 165
E.9.2 Taxation accounting 165
E.9.3 Taxation system structure 166
E.9.4 Security 167
E.9.5 Policy 168
E.9.6 Budget and treasury systems 168
E.9.7 External aid and loans 168
E.9.8 Integration 168
E.10 EdMaCS: External Debt System 170
Chris George
E.10.1 Introduction 170
E.10.2 Accounts 170
E.10.3 Loans 172
E.11 RuBanCS: Financial Service Industry 174
E.11.1 The RuBanCS Project: UNU/IIST + ISP/RAS + FoRuBa 174
E.11.2 "What is a Bank?" 174
E.11.3 Overview: Institutions & Services 174
E.11.4 Banking 176
E.11.5 Securities Services 178
E.11.6 Transaction Clearing 181
E.11.7 Discussion 182
E.12 DSSfSD: Software for Sustainable Development 183
E.12.1 Towards A Domain Theory 183
E.12.2 Towards Requirements 186
E.12.3 Towards Software 187
E.12.4 Discussion 189
E.13 ReMaCS: Resource Monitoring & Control System 190
E.13.1 Allocation & Scheduling 190
E.13.2 Production 190
E.13.3 Order Processing 191
E.13.4 Performance & Estimates 191
E.13.5 Strategics 192
E.13.6 Tactics 192
E.13.7 Operations 193
E.13.8 Discussion 193

F A Development Analogy: The Inverted Pendulum **194**
F.1 The Analogy 194
F.2 Some Domain Analysis Calculations 197

G UNU/IIST Reports & Notes 1992–1997 **199**
G.1 UNU/IIST Reports 199
G.2 UNU/IIST Notes 208

Syntactic Structure — and Reading Guide

This report is structured as a main report, sections 1 — 5, supported by a large number of special appendix sections B (starting page 47) — F (starting page 194).

The report is intended to be read "through browsing or sampling": *a little bit here, a little bit there!*

Special appendix sections C.1 (starting page 50) — E.13 (starting page 190) provide cursory "glimpses" into various research topics and into advanced, application specific development subjects.

I thank my colleagues for providing significant input to, and in many cases the entire text of many of the special appendix sections.

Motivation

This report is written as a personal account of one facet of UNU/IIST over her first almost five years. That facet concerns the research and advanced development that took place in those years under my direction.

It is written for a broad audience: Peers of UNU/IIST, including fellows of and colleagues of staff of UNU/IIST; staff of funding agencies; partners around the world; etc.

Semantic Structure

The report starts out as a 'popular science' account. That part is intended to put the development paradigm of Domain Theory, Requirements Capture and Software Design in context. Appendix F brings an analogue to this paradigm. It is taken from Control Theory.

We then treat the concept of Design Calculi (also known as Formal Methods) in section 1.3, section 2 and appendix C.1–.3, and survey the research and the advanced development projects in section 1.2, section 3 and appendices D and E. Details of the overall advanced development approach is covered in section 4 and appendix C.4

Finally we comment extensively on our experience, successes and failures, in section 5.

UNU/IIST is briefly covered in section 1.5 and appendix B.

Appendix A brings a bibliography of those reports etc. directly referenced in the text. Appendix G lists all present UNU/IIST Reports and Notes (sections G.1, respectively G.2).

1 Introduction

1.1 "Case Story Examples"

The "case story" provides a background for one of UNU/IIST's advanced development projects — see item 3 (page 17 section 3.2.2) as well as special appendix section E.3 (starting page 115). It illustrates our Domain – Requirements – Design approach to software development.

1.1.1 A Domain Example

You are waiting for a bus. It is raining. Slush November. You're going home after a long day at the office. After the bus journey follows a train ride and then another bus trip. At your end station the bus is not on time. And at the end of your last bus trip the taxi rank is empty. You walk the last mile or more home.

We say that this example reflects aspects of the Domain of transports — as it is expressed in an indicative mood [62] by asserting some reality, some fact.

Establishing a Domain Theory and Analysing this theory is considered a prerequisite for any meaningful software design. First we must know 'the laws' of the application domain. We must know *what there is* — independent of, that is: *before computing!*

1.1.2 A Requirements Example

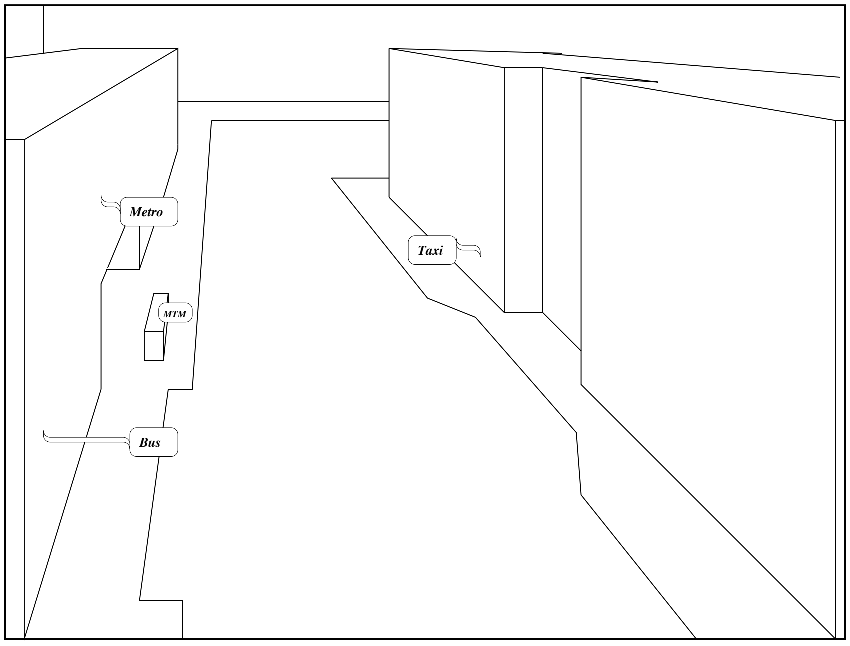
At your office you can either take a bus, a taxi or a metro. Cf. figure 1 on the facing page.

So you would like to inquire as to a best route to some other metropolitan address. You'd hope that one these new $_3$MTM*'s ("multi–media metropolitan transport machines") would be installed. Then you could be advised on combined bus, subway and taxi trips. You could buy combined tickets. And even ordering the taxi — which will know when, approximately, you will arrive at the rendez vous!*

We say that this example reflects aspects of possible Requirements to the domain of transports — as it is expressed in an optative mood [62] by expressing some wish.

Establishing, i.e. Capturing a Requirements Definition and Analysing it is likewise considered a prerequisite for any meaningful software design. After understanding 'laws' of the application domain, we must then know *what is wished* — independent of *how it is computed!*

Figure 1: Transport Service Offerings



1.1.3 A Software Development Example

Our last example is that of the software engineer: *She has to adapt an existing software system by extending it with new functionalities.* Let us, to keep in line with the above examples, assume that — *the system already provides some transport inquiry & command facilities and is to be extended with further such.* Maybe the first, existing facilities covered trains and busses, and maybe the new facilities are to incorporate taxis. *Now she would probably find that the software architecture and program organisation is such as to basically necessitate a complete rewrite of most software! Therefore she would wish that some project manager had mandated that the existing software should have been implemented in (for example) a homomorphic manner (perhaps using some object-oriented principle). Then it would embody all the train and bus concepts of the* domain *and* requirements *in a clearly identified way. It would therefore facilitate the addition of new, the extensions to existing, and (however rarely) the deletion of old descriptions. These are text parts of the* domain, *the* requirements, *and the various* software, *including* code *descriptions.*

We say that this example both reflects aspects of overall Development and Software Design — as it is expressed in an imperative mood [62] by conveying a directive to be followed in Development & Design.

Certain 'meta-requirements' must be put to the structure and contents of [i] Domain Theories,

[ii] Requirements Definitions, and [iii] Software Designs, in order for Development[1] to proceed in a professional manner.

1.1.4 A 'Trilogy' of Development Stages

Thus UNU/IIST pursues advanced software development projects in a staged manner:

- Domain Descriptions
- Requirements Descriptions
- Software Descriptions

We shall have much to say about this 'triptych'[2] of descriptions throughout this report. For the time being we invite readers to study special appendix section F on page 194.

1.2 Advanced Development

At UNU/IIST, in Macau, near Hong Kong and GuangZhou (Canton, South China), researchers in close collaboration with fellows from all over the developing world are working on the software aspects of information technology problems like those outlined above — a variety extending far beyond most peoples' imagination!

1.2.1 Software Quality

Their concern is quality: that whatever software eventually accrues is correct; meets end-users expectation; is dependable: accessible, reliable and safe. That the software systems can be easily extended to meet new requirements; etc. But what is meant by 'correct', by 'meeting end users expectations', etc.? UNU/IIST helps secure, properly and professionally, that answers to these questions become criteria that are met by the software. Therefore, UNU/IIST, her small scientific staff and its many fellows, research and deploy a variety of Design Calculi, also known as 'formal methods' (although that is not a most fitting label). They likewise research and deploy a variety of accompanying methods. Each method is specifically 'geared' to the Problem Frame [62] at hand. (We shall later in this report specifically discuss these two issues: Design Calculi and Problem Frames.)

[1] Software Development is here used a term that comprises all three stages [i–ii–iii]

[2] Tripartite, trilogy, etc. are here used as synonyms.

1.2.2 Infrastructures: Meeting the Needs of a Developing World

UNU/IIST is expected to help meet the software technology needs of developing countries. Therefore UNU/IIST and her fellows focus on such application domains as can perhaps best be characterised as Infrastructures.

According to the World Bank, 'infrastructure' is an umbrella term for many activities referred to as 'social overhead capital' by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spill-overs from users to non-users). We take a more technical view, and see infrastructures as concerned with supporting other systems or activities. Software for infrastructures is likely to be distributed and concerned in particular with supporting communication of data, people and/or materials. Hence issues of openness, timeliness, security, lack of corruption and resilience are often important.

Examples of infrastructures, or, more precisely, infrastructure components, are: *transport systems (railways, air traffic, airlines, roads, etc.), manufacturing industry, telecommunications, the financial industry and its services (banking, insurance, securities, clearing, etc.), [multi-lingual] document processing (creation, editing, formatting, etc.), ministry of finance administration: taxation & budget, health care, education, power,* etc. Later sections and special appendix sections, will list, respectively highlight, several "ambitious" projects dealing with such "large scale systems".

UNU/IIST considers her emphasis on *Software Support for Infrastructures* and her Research into the *Infrastructure Concept* to be a distinguishing, unique feature.

We refer also to section C.5 on page 88.

1.2.3 Scalability

How is it that UNU/IIST and her fellows, working on many projects — in groups of usually one scientific staff and one to three or four fellows — are able to "tackle" such seemingly large systems? The answer lies in the strict approach taken in all of UNU/IIST's advanced development projects:

- A 'Trinity' of Stages of Development:

  Before (iii) Developing the Software using Design Calculi, precise (ii) Requirements are established, also using Design Calculi. But before this, a proper (i) Theory of the application Domain is established and Analysed again using Design Calculi.

- Use of Abstraction and Mathematics:

  Abstraction is science & engineerings most important way of capturing the essence of matter before delving into details.

And, as in other responsible engineering disciplines, mathematics is a tool for expressing abstraction. It is applied, wherever reasonable, to complement informal descriptions. This mathematics is usually in the form of one formal description language (system: language + proof system) or another.

Constructing, as in other traditional engineering disciplines, formal models, allow developers to analyse proposed systems long before costly implementations. Models also allow us to predict desirable or undesirable properties, to compute necessary resource requirements, etc.

- Careful "Mixture" of Informal & Formal Descriptions:

  The world: reality, is informal. Communication with clients — those desiring (procuring) the software — must necessarily be in terms of colloquial (national, natural, but precise) language. Every stage and step of development results in (soft, "non-technological") descriptions. Therefore careful attention to the use of both informal and formal language is important.

  Informal descriptions lets the casual reader understand, but formal descriptions gives us highest attainable — but not necessarily ultimate — trust in what the descriptions purport to convey.

- Stepwise Refinement:

  Above we mentioned the need for abstraction. Abstraction captures the essence before proceeding to necessary details. Refinement is a (set of) technique(s) for — as well as the actions of — evolving, orderly and with greatest confidence, from abstractions to efficiently executable code.

- Problem Frame Orientation:

  Techniques for constructing compilers are usually very different from techniques for producing information systems, and they are all different from techniques for producing control systems (viz.: for train monitoring & control), etc. Michael Jackson's [62] concept of problem frames is therefore applied.

In various sections, and in the special appendix sections, this report will illustrate several of the above issues.

1.3 Foundational Research

But before we cover, in more detail, the advanced software development projects and their construction methods, we shall first describe another facet of UNU/IIST: her research activities.

1.3.1 A Discourse on 'Science & Technology'

Although the acronym UNU/IIST refers to Software Technology, UNU/IIST practices R&D in the spectrum between and including *Science & Technology.* To paraphrase UNU/IIST's Board Chairman 1995–1996, Dr. Gilles Kahn (INRIA, France): *Scientific results "lasts forever", whereas technology results lasts 3–5 years! Scientific results can be used for free, whereas technology products comes at a price (from US $ 99 well into the thousands!). But there are clear "paths" from one to the other.* To continue a line of example, albeit not one pursued at UNU/IIST, let us elaborate further: *In computer science, researchers deal, for example, with automata theory and formal languages. Software engineers derive from these theories automated lexical scanner and syntax error correcting parser generators. In computer science researchers deal with foundations of denotational (algebraic, etc.) semantics. Software engineers derive from these theories semantics of programming languages, etc. Finally software engineers construct varieties of compilers*[3] *using their first "derivations" — and technology has been made!*

UNU/IIST propagates to her fellows, while researching the issue, increased awareness and understanding of the "paths" between science and technology: from one to the other, and vice versa. An example of the latter is briefly outlined in special appendix section D.3 (starting page 99).

Our view can be summarised:

- **In General:**
  - There is Science and there is Technology. We only refer to such Technology which relates (somehow) to Science: reflects scientific results, etc.
  - Engineering is the process of "mapping" science into technology — and sometimes vice-versa.
  - Scientists are primarily concerned with Science.
  - Technologists are primarily concerned with Technologies.
  - Engineers use mathematics, technicians do (and need) not.
  - Technologists are almost like technicians: do not bother about science, but unlike technicians they were formerly acting as engineers do.
- **Specifically:**
  - Computing Science is the study of laws of programming and of existing information technological artifacts (incl. software) — in its widest interpretation.
  - Software Technology is software: constructed artifacts.
  - Computing Scientists — as at UNU/IIST — thus studies how to construct software and such existing (OMG[4]) artifacts as CORBA[5], etc.

[3] Examples are: Optimising, or Fast Compilation, or Interactive, Diagnostics Compilers.
[4] OMG: Object Management Group
[5] CORBA: Common Object Request Broker Architecture — an OMG artifact.

- At UNU/IIST the academic staff combine in them being computing scientists, software engineers and technologists.
- Technology pushes the market, and the market pulls (or desires) Technology. The market is not interested in science or in engineering. Computer scientists and software engineers owe their job to Information Technology.
- Computer Scientists studies the foundations of computing. UNU/IIST staff and visitors occassionally make discursions into computer science.

1.3.2 Duration Calculi — UNU/IIST, a World Centre of Research

Sections 2 on page 12 and 3.1 on page 14 outline UNU/IIST's efforts in the area of research: *science & technology!*

Suffice it here to emphasize that UNU/IIST, in it first four-to-five years of existence, has become a leading research centre in the highly crucial (European for 'Americanese' "hot") research area of *real-time, safety critical and hybrid systems.* UNU/IIST's research includes the design calculi of Durations — formal systems (language & proof systems) appropriate for the research area.

1.3.3 Advanced Infrastructure Development

— UNU/IIST, a Leading Centre of Experimental Research

This document also outlines UNU/IIST's efforts in the area of exploratory, experimental and prototype development. UNU/IIST's many advanced and feasibility development projects focus on infrastructures. UNU/IIST is a clear leader in the world of computing science & software engineering research in this area.

1.4 Technologies and Hypes

UNU/IIST's advanced development projects, like ellipses, all have two roots: one in the center of the application Domain, the other in the combined Science & Technology development approach. We do not primarily start with a Technology and then go on looking around for suitable uses of this Technology — although we do study object-oriented technologies such as put forward by the OMG[6] (viz.: CORBA[7]). See section D.3 on page 99.

[6]OMG: Object Management Group

[7]CORBA: Common Object Request Broker Architecture

1.4.1 On Object-Orientedness

Using, as we currently do, the RAISE Method (*Science*) & Tool-set (*Technology*), we need not initially bother about current Object-Oriented (OO) Technologies. This is so since all of the nice features they may offer are already in RAISE &c.: The *Schemes, Classes* and *Subtypes* of RSL (the RAISE Specification (& design) Language) richly cover modules and type inheritance of current OO-languages. The Concurrency (ccs/CSP–like) features of RSL similarly covers OO-Languages' message mechanisms.

The problem with current OO Methods, as we see it, is that they do not allow for (i) Abstraction, (ii) Decomposition, and (iii) Reasoning. Also, it seems to us, but we may be wrong, so we are looking at it, OO Methods are rather tightly "wound up" with specific kind of applications. And, to us, the world is not object-oriented! But: there is a lot of wisdom put into many of the current OO-ideas, especially when applied to structuring software designs (as in Program Organisation), so all is not that negative!

1.4.2 Hypes

At UNU/IIST we find it relatively easy to stay clear of ever eclipsing hypes: they are usually dictated from commercial sides and do not represent profound *Science to/from Technology* phenomena, and there are so many exciting real *Science & Technology* challenges.

We witness, often with sadness, how many computing science department researchers (and lecturers) are "dragged" (and "drag" students) around, always after, thus always "behind", technologies, rather than dictating, through their research (and lectures) — and with some background in science (i.e. theories) — next generation, real technologies.

We find it very sad when we observe software engineering university curricula that are based on almost no scientific content. Even technology-oriented courses need have also a center in theories. Temporary "appeasement" of so-called business & industry demands for vocationally trained, rather than academically educated, B.Sc.'s and M.Sc.'s represents a serious "knee-fall"!

1.5 About UNU/IIST

UNU/IIST is the UN System's only Software Technology Research, Advanced Development and Post-Graduate & Post-Doctoral "Training" Center.

1.5.1 Charter

UNU/IIST is here to help make sure that more and more developing countries, in their progress away from "developing" status, are becoming increasingly self-reliant wrt. software technology:

- Software Technology Procurement:

  It is one of UNU/IIST's obligations to help make sure that public & private institutions, in developing countries, increasingly become able to procure adequate quality software, and, through that ability, obtain adequate quality computing systems (i.e., including hardware). These institutions are infrastructure institutions: transport, manufacturing, telecommunications, public administration, financial industry, health care systems, etc.

  UNU/IIST therefore focus on propagating awareness of and techniques for application domain understanding and requirements capture. Target of this dissemination are local universities, research institutes and infrastructure institutions.

- Research, Development & Education:

  It is a fundamental aim of UNU/IIST to help ensure that local professionals increasingly take over the tasks of procurement and, when need be and opportunity arises, to develop parts of the software technology themselves. In this way many developing countries could be freed from its previous "uneasy" reliance on "expert" consultants. These are anyway, and usually designated by international aid agencies such as the UNDP, The World Bank, EBRD (European Bank for Reconstruction & Development), ADB (Asian Development Bank), IDB (Inter-American Development Bank), etc. These consultants and the software providers are from, i.e. are seen as representing the industrial world!

  Several developing countries are now emerging who can show good cause why it makes more sense for them to do own development. Eventually they become both sufficient scale domestic users, and exporters to the international market, of software for large scale infrastructures.

  UNU/IIST is therefore active in "training" industrial software developers and university staff in advanced methods for software construction and in transferring such courses for local institutions to propagate further. For local universities to better select appropriate design calculi UNU/IIST is also active in "training" researchers in "front-of-the-wave" research topics and techniques. Many fellows have had their research reports — researched and written while at UNU/IIST — published at strictly censured international conferences. UNU/IIST funds their participation at these conferences: in the US, Europe and Asia (— wherever the need be!).

1.5.2 The UNU/IIST Agenda

UNU/IIST works with university software technology departments, academy & government (sectorial) research institutes, and with private & public sector IT and IT-user institutions provided they are willing to adopt, adapt to and propagate the UNU/IIST Agenda:

- **The UNU/IIST Agenda:**

To research and develop software according to the principles of:

1. Design Calculi;
2. Software Support for Infrastructures,
   Design Techniques for Real-time and
   Telecommunications Systems;
3. Problem Frames;
   and the "tripartite division of labour":
4. Development $\approx$ Domains $\oplus$ Requirements $\oplus$ Design.

This report is about that Agenda!

Our fellows come primarily from such institutions which can convince UNU/IIST of their sincerity & seriousness in, and ability to follow up with respect to this Agenda.

1.5.3 More on UNU/IIST

A special appendix section B (starting page 47) gives a (further) background for UNU/IIST.

2 UNU/IIST's Design Calculi — Formal Methods

By a 'Design Calculus' we understand: (i) a formal language (formal syntax and semantics), (ii) a proof system — which may not necessarily be tool supported, although that is highly desirable, and (iii) a method of use. The latter includes rules for stepwise refinement (that is: "transformation" from abstract to concrete, execution efficient, forms).

Three Design Calculi and one "meta" Method are currently at the focus of UNU/IIST's research and advanced development activities:

1. DC — The Duration Calculi:

   Time is a "tricky" notion. It sometimes is understood more concretely than need be. The result is that temporal properties of domains or required software usually are expressed in an unnecessarily complicated manner. Therefore it is similarly difficult to reason about temporal properties.

   The ability to ensure highest possible confidence in time dependent software systems is seen as a crucial factor for the trustworthy deployment of many new software applications. Examples of such systems abound: (i) software for (life saving) medical instrument control; (ii) software for transport systems (incl. automobiles, trains, aircrafts, ships, etc.); (iii) software for process control (petrochemical plants [incl. oil refineries], robots, machining equipment, etc.), etc.

   A family of Duration Calculi has initially been promulgated by Zhou Chao Chen. Each special Duration Calculus is "geared" to a specific class of real-time, or safety critical, or hybrid systems, etc., problems.

   UNU/IIST is a leading centre, of the several research groups, worldwide, which further develops the Duration Calculi.

   See special appendix section C.1 (starting page 50) for issues of the Design Calculi [34].

2. SDL — Systems Description Language:

   In the telecommunications field, SDL (Specification and Description Language) is widely used. The first version of SDL became a recommendation of the ITU (International Telecommunication Union) in 1976. Since it has been extended several times. It originated from an informal graphical description technique already commonly used in the telecommunications field at the time of the first computer controlled telephone switches. Unlike most existing design calculi, SDL is intuitively comprehensible to its user community – in particular in its graphical form. Therefore, it will presumably still be used for a long time.

   The current practice in software development using SDL is such that: (i) it can not be ascertained to a reasonably high degree of precision that requirements will be met and (ii) it can only be established to a very low degree of certainty, not during but after its development, that software satisfies its specification. The intrinsic highly reactive and distributed nature of prospective telecommunication services and the increasing complexity of telecommunications systems will become a compelling reason to change the current practice.

This requires: (i) a dramatically simplified version of SDL with a mathematically precise semantics for it; (ii) means for advanced analysis of SDL descriptions, enabling better grounded validation of specifications; (iii) proof rules for SDL, enabling design steps made using SDL to be justified by formal verification. These topics are currently addresses at UNU/IIST.

See special appendix section C.2 (starting page 59) for issues of SDL [44].

3. RAISE — Rigorous Approach to Industrial Software Engineering:

   RAISE stands for a formal method for documenting the development of software. RSL stands for the RAISE Specification (and design) Language. The RAISE Method & RSL is supported by a comprehensive set of RAISE Tools. Besides syntactic and other support, the RAISE Proof Assistant supports users in developing proofs about properties of RSL documents.

   See special appendix section C.3 (starting page 63) for issues of RAISE [76, 77].

4. LSD — Logical Systems Development:

   By a method we understand: *a set of principles for selecting & applying tools & techniques in order efficiently to achieve and efficient artifact.* Here the artifacts are descriptions at various levels of abstraction: from Domain Theories, via Requirements Definitions, Software Architectures, and Program Organisations, to stepwise developed Software Designs and finally Code.

   We take the view that software consists of: (i) all these *Descriptions*, of (ii) [all] the Development Tools, of (iii) descriptions of the specific method applied, of (iv) *User, Installation & Maintenance Manuals*, and (v) a few other things!

   A method for developing, using and maintaining software is therefore, as in other engineering practices, a serious matter: highly intricate, and with many stages and steps of selection and application principles, and of techniques and tools.

   We have already hinted at a number of facets of the software development ("meta") method researched, developed and applied at UNU/IIST, but refer to special appendix section C.4 (starting page 66) for further issues of this "meta" method. We use the prefix "meta" since such 'methods' as B, VDM and RAISE, also use the characterising term 'method' — albeit in a rather more narrow sense.

   Roughly, but perhaps not too diplomatically, expressed: These other 'methods' really do not provide stage-by-stage and step-by-step principles, techniques and tools. Instead they focus on the use of formal specification & design languages and proof systems.

   See special appendix section C.4 (starting page 66) for issues of LSD [16].

3 UNU/IIST's R&D Projects

The three major design calculi mentioned in section 2 (starting page 12) are researched and applied in a number of R&D projects. Each of these projects are joint between UNU/IIST staff and Fellows from leading R&D and end user software application institutions in developing countries.

3.1 The Applied Research Projects

The applied research projects are:

1. DeTFoRS — Design Techniques for Real-time & Hybrid Systems: App.: D.1

   Since the start of UNU/IIST, the group led by Zhou Chao Chen has contributed significantly to the further development and applications of the Duration Calculi, under the project Design Technique for Real-time Systems (DeTFoRS).

   The work can be roughly summarised from three aspects:

   - Logical Investigation:
     A family of real time models have been formalised in Duration Calculi, such as that of event, dependability, finite divergence, super dense computation and infinite intervals.
   - Design Methodology:
     Methods of specification, verification and refinement of real-time and hybrid systems have been investigated, along with many case studies, such as the deadline driven scheduling algorithm, real time protocols, such as inverted pendulum, water tanks and a robot trajectory tracking algorithm.
   - Tools:
     Model checking methods based on linear and integer programming, as well as proof assistant based on PVS theorem proving system have been investigated.

   See special appendix section D.1 (starting page 90) on this topic [33].

2. DesCaRTeS — Design Calculi & Research for Telecommunications Systems: App.: D.2

   UNU/IIST's research programme DesCaRTeS (Design Calculi and Research for Telecommunications Systems) is focusing attention on formal techniques, and tools in support of them, to complement SDL (Specification and Description Language) for more rigorous approaches to software development in telecommunications.

   The primary aims of this programme are: (i) to enhance the possibilities for advanced analysis of SDL descriptions, thus enabling better grounded validation of specifications; (ii) to turn SDL into a full-fledged design calculus, thus enabling design steps made using SDL to be justified by formal verification. The research topics of the programme include:

(i) theoretical underpinning of the communication and timing mechanisms in SDL; (ii) underlying models for the semantics of SDL; (iii) logics to analyse properties of systems described in SDL; (iv) proof rules for SDL.

See special appendix section D.2 on page 95 on this topic [45].

3. CaSIno — CAtegories for Systems INtegratiOn: App.: D.3

   This research project examines the problem of integrating heterogeneous applications: written in different languages, "running" on incompatible machines, using protocols that cannot "talk to each other". It also looks at how the problems created in this environment are tackled within increasingly accepted (ODP[8]) distributed platforms like e.g. the Common Object Request Broker Architecture (CORBA[9]) and how the new problems arise due to the lack of any semantic constraints in CORBA's (and any other [to us] known) Interface Definition Language (IDL).

   See special appendix section D.3 on page 99 on this topic [47].

4. LSD — Logical Systems Design: App.: C.4

   This ("background") research examines problems of Domain Theories, Requirements Capture, Software Architecture and Program Organisation, of the relation between successive pairs of these, and investigates the concept of Infrastructures.

3.2 The Advanced Development Projects

3.2.1 The "Trinity" of Development Stages

UNU/IIST projects practice a three-staged development paradigm:

1. Domain Theory & Analysis (incl. Validation):

   Software development first requires a thorough understanding, through Domain Analysis, of a much wider spectrum of the application domain than may, at first sight, be warranted by requirements of the initial target software.

   Thus all of the projects, listed and briefly explained in section 3.2.2 on the following page, first build and experiment with Domain Theories. Typically we spend up towards half, and in a few cases far more, of our staff and fellow resources on developing informal and formal Domain descriptions.

2. Requirements Capture, Definition & Validation + Verification:

   Then, after perhaps six or more months (typically 12–18 man-months) of effort, informal and formal target Requirements are elucidated. They are carefully developed from Domain

[8] ODP: Open Distributed Processing

[9] CORBA is part of the OMG (Object Management Group) effort — which is closely followed by UNU/IIST.

descriptions. At UNU/IIST we spend typically around a quarter of our manpower resources on Requirements Capture & Definition.

3. Software Design & Verification:

   And only after some requirements capture effort is proper Software Design commenced. Formal approaches are mandatory as they give us highest attainable confidence in the development.

UNU/IIST has hitherto emphasised the first two stages. UNU/IIST's resources does not allow serious software implementation. Some advanced development projects have, however, lead to prototype demonstrator software. All advanced software fellows are, however, trained in actual, final implementation.

Section 4 on page 22 and special appendix section C.4 on page 66 further outlines, respectively additionally describes the development method approaches currently pursued at UNU/IIST [76, 77, 16].

We have numbered the above 'stages'. Real development iterates ("cycles") between these stages. But we start and present proper recording of development in the order listed.

3.2.2 Project Enumeration

Some advanced development projects are:[10]

1. RaCoSy: Railway Computing System: App.: E.1

   Railways, especially freight (goods) transport, will continue to play an important, indispensable rôle in most countries of the world. Proliferation, urgency and safety of, and end-user involvement in, rail services mandate increased automation.

   The target software for a joint project, with the (PRC) Chinese Ministry of Railways' Computer Centres in Beijing and Shanghai, was that of a `Running Map` system, for the monitoring & control of train dispatch (along the busiest 600 kms. stretch of the Chinese railways: between Zhengzhou and Wuhan). Despite this an application domain description (cum theory) was established for a much wider spectrum of railway operations. That theory included careful models of (i) `railway net & services development`, (ii) `timetabling & scheduling`, (iii) `shunting & marshalling`, (iv) `station management` [69], (v) `passenger ticketing, information, & handling`, (vi) `freight reservation, tracing, & handling`, etc.

   Currently UNU/IIST is working with two institutions in Moscow[11] (The Russian Federation) on a system for `commercial marketing units` (i.e. customer-oriented facilities for

[10] The order of listing is rather arbitrary!

[11] These two are: the Russian Academy of Science's Institute of Systems Programming (ISP/RAS) and the Ministry of Railway Transport.

profitable and comprehensive passenger and freight services) [75].

See special appendix section E.1 on page 103 on this topic [75, 69, 20].

2. ATC'2000: Air Traffic Control: App.: E.2

   You have read much about air traffic congestion in the US (and elsewhere), about the breakdown of computerised air traffic control, and about TCAT–II: computerised systems for automated aircraft collision avoidance. Where UNU/IIST is located, within a radius of less than 50 kilometres there are five large (four international) airports: Hong Kong (one of the world's busiest airports), GuangZhou, Shenzhen, Macau and Zhuhai.

   UNU/IIST is therefore scientifically and technologically preparing a project that will study air traffic: scheduling of air-lanes and flights, and monitoring and control of air traffic — involving multiple airport spaces [18].

   See special appendix section E.2 on page 108 on this topic [18].

3. MiTraS: Mega-City (Metropolitan) Transport System App.: E.3

   Taking up one of the leads from our introductory "case story", MiTraS, a project joint with the City of Minsk and the Ministry of Transport in Belarus (through CIT, the Belarussian Center for Information Technology), is building application domain models and suggesting requirements and corresponding software architectures for a Metropolitan Transport System. This system should be able to co-ordinate traffic between metro (underground, subway) trains, city, suburban and regional busses, and taxis. The MiTraS software system should also allow planning: design of routes, time tables and schedules based on statistically observed and customer expressed (i.e. desired) passenger flows. Additionally MiTraS, through the deployment of the previously mentioned $_3$MTM's[12], should enable potential passengers to inquire about suitable routes, traffic congestion (i.e. potential delays), and prices, and to book combination tickets, including reserving taxis [43]!

   See special appendix section E.3 on page 115 on this topic [43].

4. The General Transport Problem: App.: E.4

   The three transport projects: RaCoSy [75, 69, 20], ATC'2000 [18], and MiTraS [43], as well as a currently dormant UNU/IIST project: RoMaCS: (*Road Monitoring & Control System*) [61], has lead to a more foundational research project: A General Theory of Transport Domains. We also refer to [83].

   See special appendix section E.4 on page 122 on this topic [74].

5. MI^{2}CI: Manufacturing Industry Information & Command Infrastructure: App.: E.5

   The project performs a broad study of the issues pertinent to the manufacturing industry. The high-level goal is to investigate how information technology can be best applied to support the development of manufacturing enterprises in developing countries, able to respond quickly and intelligently to changing market demands. The emphasis is on creating mathematical models of enterprises (in all aspects of marketing, administration, finance

[12] $_3$MTM: Multi-Media Metropolitan Transport Machines (!).

and especially production), supply-chains and products, as a prerequisite for the systematic development of software for information and command infrastructure. In particular, the project will study applicability of the emerging ODP (Open Distributed Processing) and OMG (Object Management Group) Reference Models as the underlying computational structure for agile manufacturing, perhaps allowing to share production services and resources across the Internet. MI^{2}CI thus relates to the CaSIno project (see item 3 and special appendix section D.3 (starting page 99) [47]). Current MI^{2}CI work focus on 'The Market': competitive behaviour amongst suppliers, consumers, producers and traders. At the same time current MI^{2}CI R&D relates these competitive facets to the coordination aspects of what goes on inside the individual manufacturing enterprise (CIM, etc.).

See special appendix section E.5 on page 141 on this topic [54].

6. MultiScript: Multi–Lingual Script System: App.: E.6

The MultiScript project is studying the creation and presentation of documents in which more than one language is used, for example a dictionary from one language to another. Particular emphasis is being placed on preserving the natural reading/writing direction of each of the languages in the document, so that, for example, in a document containing both English and Mongolian text the English text would appear exactly like that you are currently reading whereas the Mongolian text would be written vertically in columns, the columns progressing from left to right. Four such directions can be identified (viz.: Hebrew, old Chinese, English, and Mongolian).

The MultiScript project establishes an abstract document syntax capable of representing (recognising, generating) any combination of texts from any language of the four directions. MultiScript then examines abstract and concrete formatting rules for a variety of abstract and concrete document types (newspapers, contracts, manuals, dictionaries, letters, etc.).

See special appendix section E.6 on page 148 on this topic [46].

7. ABC'2000: Airline Business Computing: App.: E.7

The project is studying the day to day operation of an airline, including financing, scheduling and timetabling of flights, ticketing and reservations, maintenance and availability of aircraft, passenger and cargo services, air traffic control, etc.

A view is adopted wherein all of the above facets are seen from postulated *resource management* points of view at three levels. These are: strategic resource acquisition, conversion and "down-sizing", tactical allocation & scheduling, respectively operational deployment. Thus the ABC'2000 project relates to the ReMaCS research; see item 13 on page 20 and special appendix section E.13 on page 190 [17].

See special appendix section E.7 on page 155 on this topic [41].

8. DiMulTS: Digitally Multiplexed radio-Telephone System: App.: E.8

This research, advanced development and training project is initially concerned with basic telecommunications protocol aspects, in particular for a special, new Digital Multiplexed Radio Telephone System currently being researched and developed by The Philippine Government's Advanced Science and Technology Institute (ASTI).

The system being developed in The Philippines is designed for particular circumstances common in Developing Countries: dispersed small population centres with difficult intervening terrain. Hence the project contributes to the areas of both self reliance and infra-structural development.

See special appendix section E.8 on page 160 on this topic [38].

9. MoFIT: Ministry of Finance Information Technology: App.: E.9

   The Vietnamese Ministry of Finance is undertaking, with World Bank support, the development and installation of a Financial Information System. Key functions include synthesis of state budget plans, management of fund allocations, formulation and review of tax policies, exchange of data between various levels and ministries.

   This project will study appropriate infrastructures to support the accurate, timely and secure collection and analysis of data.

   See special appendix section E.9 on page 165 on this topic [40].

10. EDMaCS: External Debt Monitoring & Control System: App.: E.10

    Many countries, whether industrial or developing, must live with large foreign (even internal) debts. The World Bank encourages such countries to apply stringent rules & regulations concerning the management of such debts [7], including using PC-based software to support this management. Rudimentary software does exists ([82]), but need be "streamlined" and augmented with a long list of "features". For reasons of this, of portability, and of increased quality, UNU/IIST is proposing a complete re-development of a comprehensive External Debt Monitoring & Control System that would include "sockets" for the insertion of sophisticated economic *&c.* modelling packages that can optimise debt re-scheduling.

    See special appendix section E.10 on page 170 on this topic [51].

11. RuBanCS: Russian Bank Computing System: App.: E.11

    Banks, everywhere, are increasingly "spreading their net". As a result of recent and ongoing liberalisation of the finance industry, worldwide, financial institutions, whether banks, insurance companies, securities firms (brokers), credit card firms, etc., are taking up and including each others financial services in their repertoire of such [37, 78].

    UNU/IIST is currently collaborating with two institutions in Moscow[13] on preparing a project whose aim it is to study and build a demonstrator system that can be used to illustrate a largest variety of financial services, their inter-linking, and thus serve as a basis for training and future software systems development.

    See special appendix section E.11 on page 174 on this topic [19].

12. DSSfSD: Decision Support Systems for Sustainable Development: App.: E.12

    The *emerging economies:* China, India, Indonesia, Malaysia, The Philippines, Thailand, and Vietnam, as well as many other countries — whether developing or industrial — are witnessing *"steam roller"* land development. The construction of mega cities, of huge

[13] These two are: the Russian Academy of Science's Institute of Systems Programming (ISP/RAS) and The Russian Federation of Banks.

hydro-electrical power plants, of water irrigation controls, etc., pose almost insurmountable challenges to an oftentimes fragile environment.

Therefore DSS for SD seems a good idea — however politically correct (PC) it may appear!

The two composite terms (Decision Support System (DSS), Sustainable Development (SD)) have their meaning "corrupted" or "blurred" by jargon and political correctness (PC). Yet, to the UN and the Bretton Woods System[14], to EBRD (European Bank for Reconstruction & Development), and to many aid agencies in industrial countries, these terms are important. The terms also imply software technology — and this is where UNU/IIST enters the picture. It is important to give precise descriptions of the software "meanings" of, first SD and then of DSS. Software technology for DSS for SD imply reliance also on computerised Geo- & Demographical Information Systems (GIS, respectively DIS).

UNU/IIST, with gracious support from IDRC (the Canadian Government's International Development Research Centre), therefore (co-)organised a two week international workshop with some 40 experts on GIS, DIS, DSS and SD in February and March 1996 [42]. UNU/IIST is currently preparing a multi-partner, international joint R&D project, with participation from several African countries, China, Ukraine and Russia.

See special appendix section E.12 on page 183 on this topic [24].

13. ReMaCS: Resource Monitoring & Control System: App.: E.13

    Strategic, tactical and operational enterprise management is also about resources. Strategic resource management is concerned with the timing and financing of resource acquisition (upgrading) and conversion or disposal ("down-sizing") in expectation of predicted customer orders. Tactical resource management is concerned with the allocation and scheduling of resources. Operational resource management is concerned with the actual deployment of allocated and scheduled resources in response to actual customer orders..

    UNU/IIST, in its manufacturing and in its airline business projects ($\mathsf{MI^2CI}$, [54], respectively ABC'2000, [41]), have encountered a "bewildering array" of inadequate software technology supports — usually in the form of a far from complete collection of free-standing, "isolated" application software packages that cannot be "interfaced", and which cannot therefore exchange data, let alone invoke each others' functions.

    UNU/IIST is therefore planning a project and inviting partners to join in researching and advance developing comprehensive software systems for the problem area. The ReMaCS project is thus related to the ABC'2000 [41] and the $\mathsf{MI^2CI}$ [54] projects.

    See special appendix section E.13 on page 190 on this topic [17].

3.2.3 Application "Type Theories"

To use the term 'type theory' may be a bit preposterous! Perhaps 'signature theory' is more appropriate.

[14] Bretton Woods System: IMF (International Monetary Fund), The World Bank, ADB (Asian Development Bank), IDB (Inter-American Development Bank), etc.

Or perhaps one should just say: 'Precise Specification of Incomplete Systems'?

Some comments may be in order:

- Signatures of Knowledge Based Systems:

  The DSSfSD and the ReMaCS Domain Theories illustrate a new kind of formal description approach. Usually formal descriptions have been oriented towards computable functions. The DSSfSD and ReMaCS Domain Theories, however, describe only the signature of the most crucial functions — and then goes on to comment that these functions are not computable.

  Recently UNU/IIST has also "sketched" a *"Type Theory for IT Policy Studies"* — such IT policy studies being the interest of a colleague UNU RTC: UNU/INTECH.

  What is happening here is that in setting up these descriptions we are identifying and recording what is believed to be the most crucial concepts of the application domain. We are characterising these in terms of abstract (algebraic sorts) or concrete (model-oriented) types, signature of functions, relations, predicates (whatever), and some axioms. Usually axioms characterise "bad behaviour" — and, by *"Hegelian negation"* [74], therefore also admissible ones.

  We used the term 'Knowledge Based Systems' (KBS), and could as well have used the term 'Heuristic Systems'. The point we now wish to make is that one may wish to actually provide some computing support. Perhaps a best approach would then be to apply 'Knowledge Engineering' (KE) techniques. It seems that that is what many researchers in KE are in fact doing: proposing KBS's for such 'incompletely' specified systems. If that is so, that is: if we are correct in this observation, then what we are suggesting is that it might be a good idea to first, or at the same time, also develop a 'Signature Theory': some abstract and concrete types (sorts, respectively model-oriented types), including subtypes, and some axioms. What we are a bit worried about is that traditional KE does not clearly separate the separable concerns: *the domain, the requirements* and *the software design* let alone the *software* itself! There seems to be a tendency to go "in one fell swoop" from conceptualisation to 'running' software!

- Incomplete Systems:

  Along another line we can "defend" our investigation as also covering a concept of "incomplete" specifications.

  There are those software engineering researchers & practitioners who claim that we can never make our requirements definitions "complete",[15] and that software design must start before such "completeness" can be achieved. Similarly for domains.

  What we now ("counter-") claim is that we can at least do much better than what is normally done! Normally little, if anything, is done to secure at least a "signature theory" (types and some axioms).

[15] See item 2 on page 28.

4 The Overall Development Methods

A major aim of UNU/IIST is to propagate sound software engineering principles, both as concerns development and as concerns management. So far UNU/IIST has focused on the development methods — over-viewed in this section and further summarised in special appendix section C.4 (starting page 66) — but, in our many joint projects we have occasion to stress important project (and even, in cases, product) management principles.

- By a 'method' we understand: *a set of* `principles` *for* `selecting` *and* `applying` *a number of* `techniques` *and* `tools` *in order* `efficiently` *to* `construct` *an* `efficient artifact`.

(Here software, with all its attendant documents (descriptions) are the target artifacts.)

4.1 Methodological Concerns

UNU/IIST pursues the research & advanced development projects according to three lines of structure & form — which are therefore "Master Class" practised and taught on a daily basis at UNU/IIST:

4.1.1 Stage & Stepwise Unfolding — The "Trinity" Revisited

One — of the three lines — is concerned with the stepwise unfolding of the staged Descriptions of:

1. Domain Theory: complementary and concordant, informal and formal descriptions of the intrinsics, the support technology, the rules & regulations, the human (user, client) behaviours, the economics, etc., of the application problem domain — free from any reference to contemplated (i.e. subsequently required) computing. Domain Theories are subject to Analysis (using *Verification,* as part of *Validation: Are we building the right product?*).
2. Requirements: subset descriptions of client & user requirements, derived (Requirements Captured), by extension, from the Domain Theory, and described (Requirements Definition) in such a way as to not predicate implementation options (where possible). That is: Requirements Definitions emphasise properties, i.e. how. Validation and verification is applied in order to continue *validation* as well as to show (additional, not explicitly requirements expressed) *properties.*
3. Software Design consisting of steps from:

   (a) Software Architecture design, via

(b) Program Organisation design, to (zero, one or more) steps of

(c) Design Refinements, and finally programming language and platform system

(d) Code.

Verification is applied within and between all steps in order to verify properties and argue correctness: *Are we building the product right?*

Special appendix sections C.4.3 (starting page 68) details the above "trinity" while special appendix section F (starting page 194) presents "an-easy-to-understand" analogy. Maybe you should first consult the latter!

4.1.2 Model Experiments

But it is not as simple as perhaps outlined above! The impression you might get from the above is that development stages and steps involve only the construction of descriptions and their validation and verification as 'diagrammed' to the left in figure 2 on the next page:

But that would be a mistake to believe so. What about all the arrows of the "box-and-arrows" diagram to the left in figure 2 on the following page? What — besides refinement and verification — do they signify? Or, to put it more directly, how is an appropriate refinement chosen? In the Refinement Calculus of Carrol Morgan [73] steps are chosen from amongst possible refinement rules, but then which? The answer shall be sought in considerations of 'realisability', 'efficiency', and other, innumerable concerns.

Some examples may be in order:

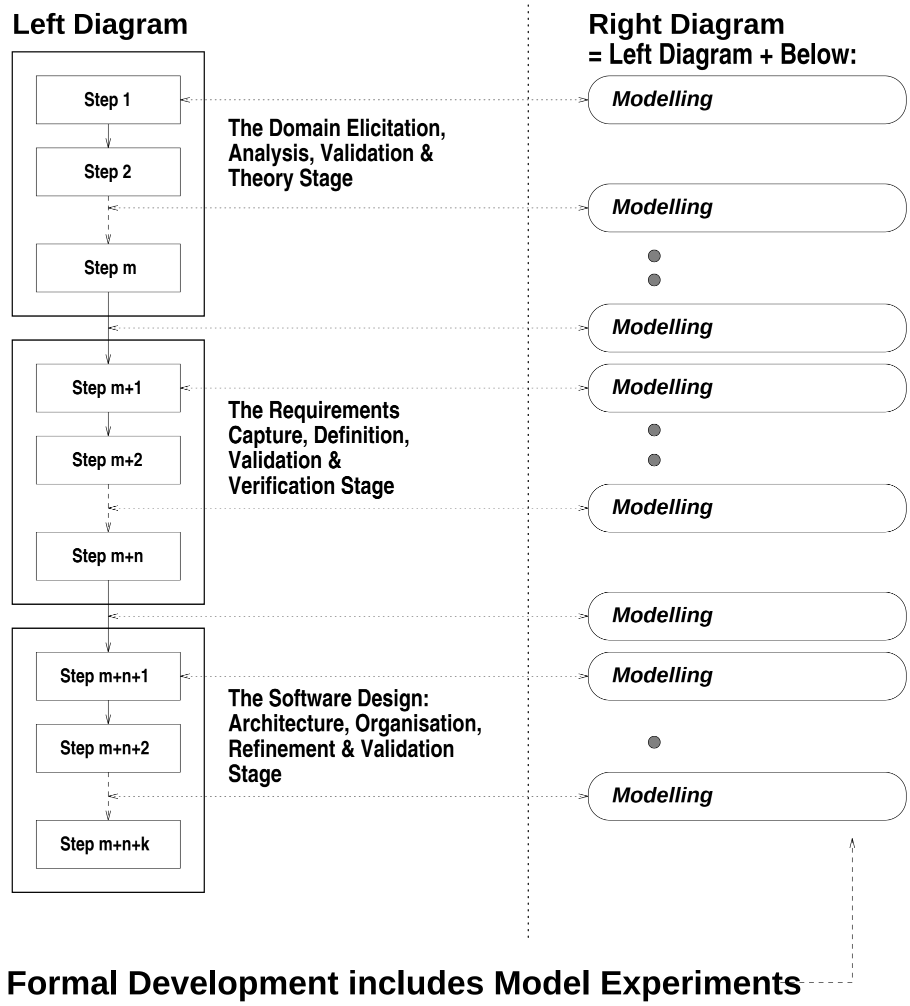
- Choice of Description:

  The point to be made here is that *within a step of development* many choices exist for its description: whether to emphasise one aspect or another. They may all satisfy validation and verification criteria, but differ in style!

  The development of a set of differential equations that purport to describe a set of reasonable movements of for example the inverted pendulum shown in special appendix section F.1 (starting page 194), requires a number of calculations. These are normally not thought of as belonging to the description of the subject (domain, etc.). In special appendix section we show this 'modelling' (which is an "exercise" within control theory) in subsection F.2 on page 197.

  The rather concrete model of metropolitan transport shown in special appendix section E.3 (starting page 115) should really be "validated against" the more abstract models of special appendix sections E.4.1 and E.4.3. In fact we probably should develop the domain description of special appendix section E.3 (starting page 115) in steps of refinement from

Figure 2: Simple & Full "Picture" of Development



that shown in special appendix section E.4.3 (starting page 134) via that shown in special appendix section E.4.1 (starting page 122). Etc.

So: *One Developer's Validation is Another Developers's Verification!"*

- Realisability:

  Choice, amongst a variety of possibilities, of for example a specific requirement may be determined by the possibility of implementing this requirement. In control theory various control principles give rise to different requirements possibilities. Classical Control Theory does not offer solutions to problems that can be solved using Adaptive or Stochastic Control Theory. Adaptive or stochastic control may not offer solutions where Fuzzy Control does. Etc. Similarly when turning to Operations Research. Thus we often have to turn to other, more classical engineering disciplines and their design calculi to justify our description (i.e. development) choices.

- Performance:

  Choice, amongst a variety of possibilities, of for example a specific software design may be determined by the need to achieve a certain performance. (One could, of course, consider this a special case of 'realisability'.) To help guarantee desired performance one develops — usually Queueing or Flow in Networks Theory models from the formal descriptions considered "in-line" (wrt. the Left Diagram of figure 2 on the facing page). Calculations over these model then help determine development, usually requirements or design, step parameters.

Other causes for "off-line" investigations and experiments abound. We cannot list them all as one can simply not enumerate human creativity and ingenuity!

We shall henceforth assume that proper documentation of all such investigations and experiments, also failed ones, is part of the documentation of appropriate steps and stages. Cf. figure 3 on page 67 and section C.4.5 on page 87.

The model experiments are called modelling in the engineering regimes (Control Theory, Queueing Theory, etc.) mentioned above. We thus have to be careful which models and which modelling we refer to (when we use those terms)!

4.1.3 Problem Frame Foca

The last — of the three R&D lines — is concerned with the form & focus taken within the development stages and steps of the Domain, Requirements and Software Design descriptions.

No "One Method": Basically there is no one "gilded and worshipped" method that carries the designer through the construction of descriptions of Domain Theory, Requirements Definition and Software Design specifications.

Depending on the Requirements problem, and, in a few cases even the Software Design problem, the development falls into one or a combination of so-called Problem Frames, a "discovery" and term coined by Michael Jackson [62].

Example Problem Frames and related example application targets are:

1. Translation Frames: — page 71
   Compilers, rewrite systems, JSP.
2. Control Frames: — page 74
   Real-time process control.
3. Information Frames: — page 77
   Management, geographic, demographic, etc., information systems.
4. Workpiece Frames: — page 79
   CAD, CAE, Accounting.
5. Connection Frames: — page 82
   Check & credit card slip clearing, electronic traders, office automation.
6. *&c.*

Referenced items of special appendix section C.4.4 (starting page 70) detail these Problem Frames.

4.2 Interplay between Informality & Formality

The result of every stage and every step of development is a set of Descriptions.

At UNU/IIST we emphasise "pairing or merging" *informal* with *formal* descriptions.

Since Descriptions form an essential "product" (and as they are all part of the Software to be developed and delivered), careful attention is paid to linguistic issues.

The interplay between informality and formality, i.e. *modelling*, also implies issues of *epistemological*, including *ontological* nature — hence, in method research at UNU/IIST we relate to such more Philosophy as well as Theory of Science issues.

Formal may be OK, but informal is certainly acceptable — in human, social life. In software development informal discourse, especially with clients, is a must. But demands of correctness, safety, reliability and other more stringent concerns, mandate the use of mathematics, or at least "computer mathematics", i.e. formal methods — or as we prefer to call them: design calculi.[16] We could also refer to our methods as precise or logical.[17]

Special appendix sections C.4.5 (starting page 87) provide further comments.

[16] Since informal is OK.

[17] Since imprecise or illogical methods sounds rather dubious [cf. Michel Sintzoff].

5 Review & Evaluation

5.1 Science & Technology: Some Comments

Many comments are in order:

1. Whither Formal Methods?

   Yes: We are Professionals! As professionals we know of no better way to secure trust, to ensure highest project & product qualities. Using design calculi, wherever reasonable — and that's quite a few places — is as natural to us as in most other responsible engineering disciplines. And because our artifacts are linguistic documents that can be brought to satisfy formal properties — rather than physical artifacts that must adhere to laws of nature, including "wear & tear" and "bulky or tiny" mass construction constraints, etc. — because of that, we are primarily forced to use mathematics.

   We do not understand all this anxiety about formal methods. There is no choice. It is mandatory.

2. Domain Theories will Never be Complete!

   Yes, sure they won't! But we can at least start. We must at least demonstrate that we have seriously tried to understand the domain. Anything else would, in a Court of Law — when Your software has been shown to be seriously wrong — amount to *Gross, Criminal Neglect!*

   We can expect the next many decades of work in academic computer science departments to be devoted to collaborative efforts in which Domain descriptions are developed and analysed. These will be joint with other university departments (Control Theory & Engineering, Operations Research & Mathematical Modelling, Business Management, etc.), and joint also with infrastructure stake-holders: operators and users. Much needs to be done!

3. 'Infrastructures' as 'Sets of (Professional) Languages':

   One way of looking at Domain Theories is to view them as giving precise abstract syntax and formal semantics to core terms of the meaning of varieties of professional languages spoken by the various groups of staff of these infrastructures and by their clients. The railway system, as an example, "is home" to several sub-languages, namely those spoken by: *(i) strategic management & planners, (ii) timetable and tactical resource planners, (iii) traffic operators (including sub-sub-languages for cabin men, engineers, station managers, etc.), (iv) ticket agency staff, (v) freight handlers, (vi) train maintenance, (vii) shunting & marshaling workers,* etc. I do not claim that we give semantics to their languages — that would be impossible. But I do claim to give alternative view semantics to core verbs and nouns, and perhaps even to some adjectives.

4. Laws of Man-made Infrastructures:

   Another way to look at Domain Theories is to see them as expressing Laws of Man-Made Systems. Just as *Newton's Laws on Gravity and Dynamics* are the basis for any serious work in for example mechanical (automotive, aeronautical, etc.) engineering, so we should expect to find ourselves, as developers of software support for such infrastructure systems, rely on their laws. Examples of laws of for example transport could be phrased in terms of railway, respectively air traffic examples: *(i) "Kirschhoff's" Law: Over a suitably chosen time interval it is the case that the number of trains arriving at a station, minus the number of trains ending their journey at that station, plus the number of trains put into service at that station equals the number of trains leaving the station; (ii) "No Ghost Trains [nor Flights]": If at ascending times t and t'' a train [a flight] is observed as part of the traffic, then at all intermediate times t' the train [resp. the flight] will also be part of the traffic; (iii) "Continuity": If at time t a flight was moving and in the air, then at some infinitesimal time δt, either before, or after, t that flight is also moving (not necessarily in the air);* etc.

   Report [74] investigates basic laws about transport.

5. Duration Calculi vs. Infrastructure & "Trilogy" Software Research:

   When UNU/IIST started, in 1992, research into the Duration Calculi was well underway: the subject was relatively well understood, and a sizable international community was following up on our research. In contrast the infrastructure software was only identified during 1993 and the sharp delineation within the "triptych" of Domain Theory, Requirements Definition and Software Design similarly only emerged during the first years of UNU/IIST. There is still a longer way to go before this "tripartite division of labour and concern" will be sufficiently well understood, let alone more widely recognised.

   UNU/IIST's research projects into the real-time and hybrid design calculi is, in a sense — which is not meant to belittle the tremendous effort that has been carried through — more straightforward: aims & objectives are to further the scientific understanding (not so much the methodological consequences), and scientific papers intended for publication must therefore be produced.

   In contrast, UNU/IIST's research into infrastructure software and the "trinity" of development stages plays "second fiddle" to actual advanced software development: we must secure that our partners have a reasonable basis on which to further develop the actual software once we have jointly contributed to clearer application domain understanding and requirements. So "production" of development documents is the first concern. Only then comes the concern for polishing our insight into scientific, publishable theory and methodology papers. Such is life at UNU/IIST: We must transfer current technology also.

6. "Floating Trinity" Interfaces:

   (a) "One Man's Requirements is Another Man's Domain!"

      A requirement to, for example, a railway system, may be that some of its operations should be supported by computing. Once that computing has been installed and is operating it becomes part of the domain!

(b) "One Man's Requirements is Another Man's Design!"

A requirement to a system may be that it be implemented using specific, usually available, platforms. But a requirement to the "same" system, but maybe for another client, may not need to "fix" such aspects of implementation — which is then left to software design.

We illustrate the issues above, amongst others, in special appendix sections E.2.5 on page 112, E.3.2 on page 117 and E.3.3 on page 120.

7. Formal Methods, cum Design Calculi, will Never Secure Complete Trust!

   That's true! But it is also true that we can do far better with Design Calculi than without. It is often used as an argument, "against" the use for design calculi, that they can not cover everything. Of course that is obvious. But not using mathematics would be irresponsible given that one can achieve so much higher confidence when using it properly than when not using it.

8. We Shall Never Fully Understand the Interplay between Formal & Informal!

   Right! Epistemological (even Ontological) issues "creep" in and make us ponder and reflect. Such issues deal with, amongst others: *"What can be modelled?"*, *"How do we know that what is modelled actually reflects 'the real' world?"*, etc. Software engineers ought be rather well versed in topics of Philosophy, in Philosophy of Science and in Theory of Science, and in the way in which philosophers reason.

9. Development is Not Linear, It is Cyclic — even "Spiralic"!

   Yes. We do not proceed straight from first doing all of the Application Domain Theory part, to then doing all of the Requirements Definition bit, and only then the Software Design.

   Rather we find an oftentimes ad hoc "cyclic iteration" between all stages and, within stages, all steps. And we also often find that we start the requirements capture while we are still working on the domain theory, and that we may even start the software design while we are still defining the requirements. Often actions on "later" descriptions help us "complete" former descriptions.

   Since the Application Domain Theory will never be "complete[d]" we will necessarily have to commence Requirements Capture and its documentation (the Requirements Definition) after we (informally) believe that we have a sufficient Domain Theory basis on which to build the (derived & enriched) Requirements. To a lesser extent this is true for Software Design: the Domain is "infinite", whereas Requirements ought to be "finite".

5.2 International Collaboration: Some Comments

The following remarks may seem, to some, not quite PC.

1. Technology Transfer ≈ Human Transfer:

   In UNU/IIST's work we aim at a kind of "technology transfer". Since we are primarily focusing on transferring the methods (techniques and tools) of development of software technology, our aim is not commercial, or industrial technology, but more training and educational. However, as fellows return to their institutions they bring with them home significant development documents and even some prototype software which together constitutes 'technology'. So technology transfer works through temporary transfer of people "the other way"!

2. "Brain Drain":

   An overall aim of the UN University (UNU) is to help alleviate (even avoid) "brain drain" — as it is expressed in the UNU Charter: by alleviating the *intellectual isolation* of lecturers, researchers and technologists in developing countries and in countries (whose economies are) in transition. Now, to really succeed, in achieving this laudable goal, would far exceed even the abilities of UNU/IIST! That would, as a minimum, require that the UN could set minimum salary and working standards for lecturers, researchers and technologists throughout the developing world. Clearly not a task appropriate for the UN. But, by confronting our fellows with latest science & technology, by training & educating them to highest levels, by conducting *front-of-the-wave* projects — such as outlined in this report — and by helping to ensure that the fellows will continue along such lines when they return to their seconding institutions, UNU/IIST can thereby entice them to "stay relatively put"!

3. The Developing World:

   One speaks of Asia as one entity — as one speaks of Europe and Latin America. For the latter it might be true. Certainly the peoples of Europe have travelled each others' lands for the past almost 2,000 years and all belong to a tightly knit linguistic, religious, philosophical, scientific, and social sphere that can perhaps best be characterised as Judean--Hellenic-Christian. The software methodology & technology propagated by UNU/IIST may be said to have first evolved in that sphere — which includes North America and Australia + New Zealand. Latin America seems also to be such an entity. But Asia and Africa? Can one rightfully refer to these as single entities. I doubt it! Afghanistan and Korea have far less to do with one another than Portugal and Sweden. The basically Islamic country of Indonesia and that of Saudi Arabia? Certainly not. Asia is so huge, and Africa likewise, that one approach to collaborating with individual countries on these continents would obviously fail. Just think of the widely diverse scripts of Asia!

   UNU/IIST is working with many institutions in most of these countries — and it is very rewarding. But to claim that we have succeeded equally well in all is wishful thinking.

   So what are we trying to do?

4. Beliefs:

   We are creating "beliefs". We are showing that there are few, if any differences when it comes to basic techniques in research into and development of software. We are showing that there are beautiful roads leading from science to technology. And that the the best can be offered to all.

There may be philosophical distinctions, such as those between, for example, an Existentialist, or Logical Positivist, or 'Popperian'[18] world and that of, for example, a Confucian world. But does it materially influence the drive of individuals and institutions on the different continents?

5. Insecurity of Collaborative Efforts:

   UNU/IIST, in order not to waste too many resources, needs relatively stable partners, needs partners who understand the spectrum between and including science to technology, and needs partners — from academia and industry — who wishes to collaborate. Economic & social conditions are seemingly currently such which hinder even pre-competitive collaboration.

   In Europe, as another example, academia and industry works now works closely together on for example the information technology R&D efforts. University departments and academy institutes, in some developing countries, are not yet prepared, as in Europe or North America, to collaborate in 'pre-competitive' areas. Instead they are concerned with creating new businesses: new sources of income for their institutions and themselves. One cannot blame them. But the situation obviously impairs UNU/IIST's efficiency if not properly handled.

5.3 UNU/IIST Now & in the Next Future

UNU/IIST has existed in barely five years. Yet we, immodestly, believe it has been a rather surprising success! We strongly believe this, and we are constantly told so by our many visitors — see Acknowledgements below. But the reasons why we so unashamedly "blast" this self-praise out to the world are that (i) UNU/IIST is small, (ii) relatively new, (iii) still not fully on the world firmament of computing science, software engineering & technology, and (iv) that it therefore need be "heavily advertised", however under the banner of "truth in advertising"!

Surely, as a final reason: (v) T*he first & founding Director of* UNU/IIST *is leaving at the end of five years and wishes his legend to be appreciated!*

- *We are currently consolidating our strengths and trying to alleviate our weaknesses. In future I expect further positive progress in both strengths and reduction of weaknesses.*

5.3.1 UNU/IIST Strengths

The current, outgoing Director, has had five years to do what is reported here. He has had and has a staff of very dedicated scientists and engineers. In the first two years there were three such (incl. the Director); since 1995 four. The fall of 1995 saw a staff of seven; and since 1996

[18] After Sir Karl Popper's Theory and Philosophy of Science

there has been eight scientific staff! Occasionally there is one or two three–six months visiting scholars.

In the first 11 months local logistics problems and a low start-up capital (both beyond our control) prevented a full scale fellowship programme and R&D.

With this staff UNU/IIST has been able to bring about a "success story" due to a number of facts:

- General Focus on Design Calculi:

  UNU/IIST claims that its exclusive focus on Design Calculi is unique: few computer science departments or research institutes can or does make sure that all of its staff are *walking in the same direction!*

- Special Focus on The Duration Calculi:

  As has been mentioned before: UNU/IIST must be considered a, if not the, leading research centre today, worldwide, in the field of research into the foundations and experimental applications of the Duration Calculi [34, 33].

- General Focus on Infrastructure Software:

  UNU/IIST further claims that its main, development focus on Infrastructure Software is likewise unique: most computer science departments and research institutes have yet to get around to this crucially important subject area [75, 69, 20, 18, 43, 61, 74, 54, 46, 41, 38, 40, 51, 19, 42, 24, 17]!

- "Directorial Power":

  Not many, if any, university or research centre computer science department chairmen, can be given the freedom in choosing strategic direction and tactical means — such as can be given to a UNU Director!

  Of course the UNU/IIST Director is accountable, in four ways: annually (i) to the Board of UNU/IIST, and (ii) the Council of UNU, and daily (iii) to the scientific staff of UNU/IIST, and (iv) the fellows. So it is simply a matter of making all these four "parties" happy.

  The Director, coming from and returning to academia, cherishes this accountability! It gives additional freedom knowing that all these four parties stand firmly behind UNU/IIST's strategic & tactical policies, as well as behind the handling of daily operations.

- "Narrowness" of Scope:

  UNU/IIST, as a computing science cum software technology research and training centre, has focused rather narrowly — as compared to what most reasonably comparable institutions must cover. UNU/IIST covers only the kind of things shown in this report: Programming Methodology along the lines of Design Calculi, in general, and Design Calculi for Real-time, Safety Critical, Hybrid and, for the time being, also for Telecommunications systems.[19]

- "Fitness" of Group & Master Class Work:

  UNU/IIST can [therefore] hire its professional staff, after conventional, but strict external evaluation, to fit the "narrow" scope — and, through the "directorial power", make this staff work, very closely, daily, with each of the fellows in their charge. The UNU/IIST staff — in a sense — does not have full academic freedom (in choosing what they wish to study), but by matching their previous and ongoing interests with the evolving interests of UNU/IIST, it can achieve high productivity, job satisfaction and career opportunities! By additionally letting some professional interests of the academic staff become part of UNU/IIST's R&D agenda we can further achieve efficiency.

5.3.2 UNU/IIST Weaknesses

What are the present weaknesses of UNU/IIST or the UNU/IIST concept? Perhaps they are:

- No Local R&D Projects:

  Despite valiant efforts, many tries and several carefully detailed proposals, UNU/IIST has not been able to involve Macau's public or private sector in joint R&D efforts. Despite many inquiries, formal and informal, we have, so far, not been able to determine what is locally & feasibly expected from us. This is an unusual situation for any research and training centre — and has been a great disappointment to us. It stands in stark contrast to our previous experience, in the US and Europe, and to our current successes in many developing countries etc.

  Perhaps the reasons shall be sought in Macau's relatively recent entry into higher profile business and in the lack of a local technology industry & tradition.

  Whatever may be the reason, it is a fact that to conduct such local R&D projects, UNU/IIST would dearly wish to collaborate with staff and students at, and with software engineering candidates from the local University of Macau. *Over a short period of five years it has rather successfully established a very popular Software Engineering Programme.* But the university has only a very recent history and thus very little science & technology tradition. It has yet, and understandably so, to generate sufficient numbers of local software engineering M.Sc.'s. This makes it a special challenge for UNU/IIST to work locally at the post-graduate and postdoctoral level at which any UNU RTC[20] is assumed to operate, and at which it operates with public & private institutions in many other countries.

[19] To put this "narrowness" claim in perspective, let us recall that most other computer science departments must also cover theoretical aspects such as Complexity Theory, Cryptography, Computability, Foundations of Algebraic & Denotational Semantics, etc., and such programming aspects as Compiler Design, Operating Systems Design, Database & Information Systems Design, AI & Knowledge-Engineering, etc.

[20] RTC: Research & Training Centre

UNU/IIST, however, continues to feel embarrassed at its lack of a reasonable local "track record". Given that for example Vietnam's Ministry of Finance, The Philippine Government's ASTI (Adv. Sci. & Techn. Inst.), The Chinese Government's State Education Commission, State Sci. & Techn. Commission, Academy of Sciences and National Natural Science Foundation, and many other countries' institutions can work successfully with UNU/IIST, it ought to be able to secure at least one similar local project!

- Instability of Partners:

  Initially UNU/IIST focused, as does any traditional research centre or university department in Europe and North America, mostly on individual, person-to-person contacts: attracting fellows in this way. This proved, in a few instances, to be a mistake. Since 1995 UNU/IIST has therefore "formalised" its contacts by "institutionalising" them. In the past there was no formal, moral obligation on the part of seconding institutions to place returning fellows in positions where they would be able to contribute locally with the knowledge and results, incl. software, gained while at UNU/IIST.

  With a number of institutional 'agreements' (currently in the form of [legally speaking] non-binding Memoranda-of-Understanding) now in place, there are at least formal, moral obligations. The PRaCoSy, respectively the MoFIT projects are cases in point. The former (PRaCoSy) basically had little impact in China (due, it seems, to internal PRC Ministry of Railways management). The latter (MoFIT) is virtually guaranteed a high local (i.e. Vietnam) impact (due to strong management commitments).

  Most recently UNU/IIST has been able to establish formal agreements, at top level, with The State Science & Technology Commission for China, The State Education Commission for China, The Chinese Academy of Sciences, and The National Natural Science Foundation for China. These agreements help provide stability while giving UNU/IIST a very strong basis from which to effectively propagate its Agenda in China. They set an example for other's to follow.

- Inability to Attract UN + Bretton Woods System Funding:

  The UN and the Bretton Woods System has still to grant UNU/IIST consultancy work, and, through UNU/IIST to grant our partners funds for using UNU/IIST as their consultant and trainer.

  When one considers that the UN System: UNDP, UNESCO, UNIDO, etc., is strongly engaged in infrastructure development support, then there must be a good case for UNU/IIST to be involved with that system! Similar comments can be made wrt. the Bretton Woods System: The World Bank, ADB (Asian Development Bank), IDB (Inter-American Development Bank), etc., and the EBRD (European Bank for Reconstruction & Development). UNU/IIST will continue trying to get funding, for our partners, for joint R&D projects — so that UNU/IIST with them can embark on more and larger efforts.

  UNU/IIST would like to further propagate and enlarge the scope of existing projects: RaCoSy, ATC'2000, ABC'2000, MiTraS, DiMulTS, MoFIT, EdMaCS, RuBanCS, DSSfSD, and ReMaCS — mentioned elsewhere in this document — as well as new projects in the areas of (i) River Basing Management, (ii) Disaster Preparedness, Assessment, Relief & Recovery Management, (iii) Health Care Computing Systems etc.

At the moment, perhaps due to bilateral (i.e. donor) constraints, the international aid system (listed above), uses national (i.e. donor country) consultants and trainers where UNU/IIST could be at least as good a choice! Similarly that system seems to encourage acquisition of software from industrial world industries. UNU/IIST is not interested in "growing big & rich" on consultancy, training and development contracts. UNU/IIST is, instead, interested in transferring highest development technologies to developing countries so that they themselves can carry on and create local software industries etc. Contrast that with the commercial aims & objectives of consultant (etc.) firms of industrialised donor countries!

It also seems that too many UN and World Bank agencies, departments and divisions are "scrambling over one another" busily engaging themselves in significantly overlapping programmes and projects in the broad area of information technology (IT). We all wish to become successful. And IT promises success. From the point of view of UNU/IIST we are all too often saddened by the lack of will to really co-operate within the UN and Bretton Woods System. Where UNU/IIST spends relatively few monies, but seemingly very efficiently, on highly intensive advanced training (cum education) courses, we often see UN agencies (etc.) arrange seemingly expensive one week awareness & "training" events with, we believe, too many topics covered in too short time by too many different people. We wonder whether anything really useful is "technology transferred" at such events.

UNU/IIST is very effective in its work with institutions in developing countries and challenges the above-mentioned agencies to show equal efficiency!

UNU/IIST will continue to inform the UN System of the above views and their many possibilities.

- Inability to Attract IT Industry Funding:

  The establishment of UNU/IIST was preceded by two studies: an informal, impromptu Exploratory Study (Spring 1987), and a more formal Feasibility Study (Spring 1989). The studies, although not binding, raised the expectation or concluded that UNU/IIST:

  1. would be able to attract funding from the computer industry, and
  2. should be able to generate income by developing software at cost plus!

  Both these assumptions turned out to be rather "optimistic":

  1. The computer industry has not been willing to fund UNU/IIST, and perhaps for at least three reasons: α: their businesses went through some very bad years from which they have hardly recovered; β: they tend to donate equipment to universities of national countries (and UN is not a country); and γ: the industry is mostly US: the UN concept does not "swing" well with public opinion in the US.

     But UNU/IIST hopes to "turn the tide".[21]

[21] Recently Sun Microsystems of California has pledged an approximately US $ 25,000 system for the MultiScript project (see special appendix section E.6 (starting page 148)) in Mongolia.

2. A UN institution can and should not engage in commercial development or stand in the way of such. UNU/IIST wants its partners in developing countries (and in countries whose economies are in transition) to thrive: to obtain their rightful share of worldwide software development and commercialisation.

   The UN should, and cannot stand in the way of the institutions it is mandated to serve.

   A UN institution cannot be legally responsible for any commercial product.

- Stretched Present Resources:

  To even keep the present activities going, and to plan for next ones, require more resources that UNU/IIST either has, or can expect to continue drawing heavily upon. That is: present resources have been over-utilised in the past. All staff has worked far more than the 35 hours a week which is length of the standard UN System work week. It is a safe bet to guess, that taken as an average the present staff has typically been working more than 50 hours a week the first three and a half years, with only a slight decrease, if any, in the last 36 months! The Director must now secure that the scientific staff gets time to complete and write up more individual research — including developing the Theory aspects of Infrastructure Domains.

  Such research is, after all, the basis for UNU/IIST's next programmatic activities.

6 Acknowledgements

I could not have directed the work and written this report without the compassionate, always curious, continuously prodding and inspiring collaboration with my staff, with now innumerable UNU/IIST fellows from five continents, and with visiting researchers and colleagues from six continents.

6.1 Scientific Staff

Daily life at UNU/IIST, as it has been, would not have been possible without the constant, full and extremely loyal support of past and present staff colleagues:

Chris George[22], Dang Van Hung, Tomasz Janowski, Kees Middelburg, Richard Moore, Søren Prehn, Zhou Chao Chen, and Xu Qi Wen;

as well as long term visiting scholars:

Pandya K. Paritosh, Bo Stig Hansen, Jan Goossenaerts, and Nikolaj S. Nikitchenko.

6.2 Fellows

Then I wish to thank all the fellows who have spent time at UNU/IIST since its inception:

Cleta Melagros Acebedo, Rumel V. Atienza, Aristides Dasso, Isabel L. Cafezeiro, Philip Chan, Dang Van Hung, Dao Nam Anh, Do Tien Dung, Doan Anh Tuan, Dong Yu Lin, Roderick Durmiendo, Myatav Erdenechimeg, Laure Pauline Fotso, Fu Hong Guang, Gao Jian Ping, He Wei Dong, Hoang Thi Tung Lam, Hong Mei, Huang Xuan Huan, Mia Indrika, Kåre Sloth Jensen, Jin Dan Hua, Ko Kwang-Il, Souleymane Koussoube, Lai Jian Huang, Hans Laustrup, Le Thi Linh Chi, Li Xiao Shan, Li Xuan Dong, Li Yang Min, Liu Lian Suo, Liu Xin, Ma Chao, Mao Xiao Guang, Swarup Mohalik, Wojciech I. Mostowski, Namsrai Yumbayar, Ngo Quoc Tao, Nguyen Le Thu, Nguyen Quoc Toan, Roger Noussi, Ou Song, Erwin Paguio, Pham Hong Giang, Pham Hong Thai, Phung Phuong Nam, Qiu ZongYan, Mohammad Rais, P. Venkat Reddy, Suman Roy, Marisa Sanchez, Gueorgui Satchok, Manoanjan Satpathy, Radu Soricut, Arun Kumar Pujari, Sun Guo Qing, Tan Xin Ming, Tian Si Yuan, Tran Thi Mai Lien, Yaroslav Ussenko, Wang HanPin, Wang Ji, Bogdan Warinschi, Belawati Widjaja, Yang Dong, Yang Zheng Yu, Yu Hui Qun, Yu Xin Yao, Namsrai Yumbayar, Vladimir Zadorozny, Zheng Tao, and Zheng Yu Hua.

[22]— all listings are alphabetically by family name

6.3 Visitors

I also wish to thank all our short and long term visitors in the first five years:

Egidio Astesiano, Bach Hung Kang, Ralph-Johan Back, Jaco W. de Bakker, José Barata, Klaus Berkling, Egon Börger, Luis M. Camarinha-Matos, Paul Curzon, Chen Huo Wang, Chen Tien Chi, Chen Zong Ji, Oliver Corff, Pierre-Louis Curien, Norihisa Doi, Henrik Esbensen, Glenn Evans, Anthony Finkelstein, Michael Fourman, Kokichi Futatsugi, Norman Gibbs, David Gries, Armando Haeberer, Michael R. Hansen, He Jifeng, Ric Hehner, John E. Hopcroft, Michael Jackson, Kanchana Kanchanasut, David Kahaner, Kyu Chul Kang, Kouichi Kishida, Leslie Lamport, Hans Langmaack, Peter Gorm Larsen, Rogerio de Lemos, Sereeterin Lodoysamba, Liu Xi Ming, Lu Jian, Zohar Manna, Hermann Maurer, Vladimiro Miranda, Jayadev Misra, J Strother Moore, Carrol Morgan, Peter D. Mosses, Mukund Madhavan, Jürg Nievergelt, Ernst-Rüdiger Olderog, Park Chan Mo, Yakup Paker, Prakash Panangaden, Manfred Paul, Amir Pnueli, Willem Paul de Roever, Marek Rusinkiewicz, Achim Schneider, Jens Ulrik Skakkebæk, Srinivasan Parthasarathy, Erling V. Sørensen, Tang Chi Song, Andrzej Tarlecki, T. H. Tse, Władysław Turski, Jeanette M. Wing, Omar Wing, Yuan Chong Yi, Yang Lu, Yong Sun, and Zhang Jing Zhong — including also scientists & technologists who have elsewhere contributed to UNU/IIST's programmatic activities.

6.4 The UNU/IIST Board

We take great pleasure in also acknowledging the inspired review from members of the UNU/IIST Board:

Gilles Kahn, Ivan Havel, Ji Fu Sheng, Kesav V. Nori, Ana Regina Cavalcanti da Rocha, José Soromenho-Ramos, Maurice Tchuente, and Zhou Li Gao.

6.5 Macau and Initial Donors

We are happy to express our gratitude to colleagues at the local University of Macau (UoM):

- Profs. Ottokar Babka, Han Yan, Li YiPing, Rui Martins, and Wang Ju An.[23]

We also wish to formally acknowledge the gracious support of the Initial Donors:

- The Governments of the Republic of Portugal,

[23] UNU/IIST offers regular B.Sc. and M.Sc. courses at UoM. Several UoM staff do joint research with UNU/IIST. And UNU/IIST co-supervises UoM M.Sc. Thesis work.

- The People's Republic of China,

  and, last — but certainly not least:

- The Governor & Government of Macau

  represented by Their Excellences:

  - General Vasco Rocha Vieira, The Governor, and
  - Dr. Jorge Alberto Hagedorn Rangel, Under–Secretary for Administration, Education and Youth Affairs.

Finally we wish to acknowledge the always cheerful, charming and interesting daily support given to UNU/IIST in the last five years by:

- Dr. António Rodrigues Juniór, President, The Macau Foundation

and his staff.

A Bibliographical Notes

Below we list documents directly referenced in this report. Some of the below entries further references other UNU/IIST documents.

Some such references are to UNU/IIST Reports. Their listing starts page 199. We **bold-face** these references below. Others are for UNU/IIST Notes. Their listing starts page 208. We list the latter references in *slanted font.*

Proper UNU/IIST Reports (and, in cases, Notes) may be obtained by E-Mailing Miss Alice Pun, UNU/IIST librarian, `pci@iist.unu.edu` or `library@iist.unu.edu`.

You may wish to inspect the UNU/IIST home page: `http://www.unuiist` for latest update on UNU/IIST Reports and Notes, including how to obtain these over the net.

References

[1] D. Andrews and D. Ince. *Practical Formal Methods with VDM.* McGraw-Hill Book Company, 1991.

[2] J.C.M. Baeten and J.A. Bergstra. Global Renaming Operators in Concrete Process Algebra. *Information and Control*, 78:205–245, 1988.

[3] J.C.M. Baeten and J.A. Bergstra. Process Algebra with Signals and Conditions. In M. Broy, editor, *Programming and Mathematical Methods*, pages 273–323. NATO ASI Series F88, Springer-Verlag, 1992.

[4] J.C.M. Baeten and J.A. Bergstra. Discrete Time Process Algebra with Abstraction. In H. Reichel, editor, *Fundamentals of Computation Theory*, pages 1–15. LNCS 965, Springer-Verlag, 1995.

[5] J.C.M. Baeten and J.A. Bergstra. Discrete Time Process Algebra. *Formal Aspects of Computing*, 8:188–208, 1996.

[6] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[7] World Bank. Miscellaneous Debt Management Reports. 1: Thomas M. Klein: External Debt Management: An Introduction; 1994, Technical Paper, No.245, Washington DC, USA; 2: Debt Management Systems, 1990; 3: Debt Systems Conference Proceedings; World Bank, Washington, D.C.

[8] F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification.* Prentice-Hall, 1991.

[9] J.A. Bergstra, W.J. Fokkink, and C.A. Middelburg. Algebra of Timed Frames. *International Journal of Computer Mathematics*, 61:227–255, 1996.

[10] J.A. Bergstra, W.J. Fokkink, and C.A. Middelburg. A Logic for Signal Inserted Timed Frames. Logic Group Preprint Series 155, Utrecht University, Department of Philosophy, January 1996.

[11] J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60:109–137, 1984.

[12] J.A. Bergstra and C.A. Middelburg. Process Algebraic Underpinning of Communication and Timing in SDL. Research Report 87, United Nations University, International Institute for Software Technology, December 1996.

[13] D. Bjørner. *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, 1980.

[14] D. Bjørner and C.B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, 1978.

[15] D. Bjørner and C.B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall International, 1982.

[16] Dines Bjørner. LSD: Logical Systems Development. UNU/IIST Report: [**95**].

[17] Dines Bjørner. Models of Enterprise Management: Strategy, Tactics & Operations — A Case Study Applied to Arlines and to Manufacturing. UNU/IIST Report: [**60**], 1996.

[18] Dines Bjørner. Selected Papers & Reports on The ATC'2000 Project. UNU/IIST Reports: [**48, 52**].

[19] Dines Bjørner. Towards a Domain Theory for Financial Services & Institutions. UNU/IIST Report: [**96**], 1997.

[20] Dines Bjørner, Chris George, Hans Laustrup Bo Stig Hansen, and Søren Prehn. A Railway System. UNU/IIST Report: [**93**].

[21] Dines Bjørner, Tony Hoare, and Hans Langmaack. Provably Correct Systems. Technical report, (i) Department of Computer Science, Technical University of Denmark; (ii) Programming Research Group, Oxford University, England; (iii) Institute for Informatics, University of Kiel, Germany, 1993.

[22] Dines Bjørner, Souleymane Koussoube, Roger Noussi, and Gueorgui Satchok. Jackson's Problem Frames: Domain, Requirements and Design. UNU/IIST Research Report No. 102; Draft. Expanded version of [23], UNU/IIST, May 1997.

[23] Dines Bjørner, Souleymane Koussoube, Roger Noussi, and Gueorgui Satchok. Jackson's Problem Frames: Domain, Requirements and Design. In Shaoying Liu, editor, *International Conference on Formal Engineering Methods: ICFEM'97*, Washington D.C., USA, 12–14

November 1997. IEEE Computer Science Press; IEEE sponsored conference, Hiroshima, Japan. Brief version of [22].

[24] Dines Bjørner and Mohd Rais. Federated GIS+DIS–based Decision Support Systems for Sustainable Development. UNU/IIST Report: [**61**], 1996.

[25] R.M. Burstall and J.A. Goguen. Putting Theories together to Make Specifications. In *Proc. of (IJCAI) Int'l. Joint Conf. on AI.* Boston, Aug. 1977.

[26] R.M. Burstall and J.A. Goguen. The Semantics of CLEAR: A Specification Language. *[13]*, pages 292–332, 1980.

[27] D.L. Chalmers, B. Dandanell, J. Gørtz, J. Storbank Pedersen, and E. Zierau. Using RAISE — First Impressions From a LaCoS User Trial. In *Proceedings of VDM '91*, volume 551 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[28] Zhou Chaochen and Michael R. Hansen. Chopping a Point. In *7th refinement workshop (to appear)*, 1996.

[29] Zhou Chaochen and Michael R. Hansen. A First Order Interval Logic. Technical report, Department of Information Technology, DTU, 1996. Draft.

[30] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.

[31] Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A Duration Calculus with Infinite Intervals. In *Fundamentals of Computation Theory, Horst Reichel (Ed.)*, pages 16–41. LNCS 965, Springer-Verlag, 1995.

[32] Zhou Chaochen and Li Xiaoshan. A Mean Value Calculus of Durations. In *A Classical Mind: Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed)*, pages 431–451. Prentice Hall International, 1994.

[33] Zhou Chao Chen, Dang Van Hung, and Xu Qi Wen et al. Papers & Reports on The DeTFoRS Project. UNU/IIST Reports: [**16, 19, 20, 24, 26, 27, 28, 29, 35, 36, 38, 39, 41, 44, 49, 54, 56, 57, 73**], etc.

[34] Zhou Chao Chen, Dang Van Hung, and Xu Qi Wen et al. Reports & Notes on The Duration Calculi (DC). UNU/IIST Reports: [**5, 9, 10, 11, 13, 15, 21, 25, 40, 50, 70, 77**].

[35] Oliver Corff. Mongolian Language Support — a Mongolian Language Environment for IBM-Compatible PCs. UNU/IIST Report [**8**], (joint with Mongolian Academy of Science's Institute of Language and Linguistics), Ulaan Baator, Mongolia.

[36] B. Dandanell, J. Gørtz, J. Storbank Pedersen, and E. Zierau. Experiences from Applications of RAISE. In *FME'93: Industrial Strength Formal Methods*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[37] J. Kimball Dietrich. *Financial Services and Institutions — Value Creation in Theory and Practice.* Prentice Hall (Publ.), 1996.

[38] Roderick A. Durmiendo and Chris George. The DiMulTS Project. UNU/IIST Report: [**67**].

[39] B. Dutertre. On First Order Interval Temporal Logic. Technical report, Report no. CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Eghan, Surrey TW20 0EX, England, 1995.

[40] Chris George et al. Selected Papers & Reports on The MoFIT Project. UNU/IIST Notes: *[35, 36, 37, 38, 39, 40, 41]*.

[41] Dao Nam Anh et al. Selected Documents on The ABC'2000 Project. UNU/IIST Report: [**74**] and Notes: *[8, 9, 10]*.

[42] Expert Group (Eds. Pat A. V. Hall et al.). Decision Support Systems for Sustainable Development: Experience and Potential — A Position Paper. UNU/IIST Report: [**80**] (submitted), August 1996.

[43] Gueorgui Satchok et al. Selected Papers & Reports on The MiTraS Project. UNU/IIST Notes: *[63, 52, 79, 74]*.

[44] Kees Middelburg et al. Formal Aspects of SDL (Systems Description Language. UNU/IIST Report [**68**].

[45] Kees Middelburg et al. Papers & Reports on The DesCaRTeS Project. 1996–1997.

[46] Myatav Erdenechimeg et al. Selected Reports on The MultiScript Project. UNU/IIST Report: [**75**], and Note: *[7]*.

[47] Tomasz Janowski et al. Selected Reports & Notes on CaSIno. UNU/IIST documents, 1996–1997.

[48] Widjaja et al. Paper using the Inverted Pendulum as Example. UNU/IIST Report: [**36**].

[49] Dines Bjørner et al. (Eds.). Design Calculi in Action — Five Years of R&D at UNU/IIST. UNU/IIST Report: [**94**], Second draft version, January 1997.

[50] K. Futatsugi, J.A. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ-2. In *12th Ann. Symp. on Principles of Programming*, pages 52–66. ACM, 1985.

[51] Chris George. A Note on a Formal Model of External Debt. UNU/IIST Note: *[24]*.

[52] Chris W. George. Graph Models of Transport. Technical report, UNU/IIST, 1997.

[53] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics (Extended Abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version: Report CS-9120, CWI.

[54] Jan Goossenaerts and Tomasz Janowski et al. Selected Papers & Reports on The MI^{2}CI Project. UNU/IIST Reports: [**30, 31, 32, 62, 63, 64, 69, 92**], and Notes: *[11, 12, 13]*.

[55] John V. Guttag and James J. Horning, editors. *Larch: Languages and Tools for Formal Specification*. Texts and Monographs in Computer Science. Springer-Verlag, 1993. With Stephen J. Garland, Kevin D. Jones, Andrés Modet, and Jeannette M. Wing.

[56] John V. Guttag, James J. Horning, and Jeannette M. Wing. Larch in Five Easy Pieces. Report 5, DEC Systems Research Center, Palo Alto, CA, July 1985. Superseded by [55].

[57] M.R. Hansen, P.K. Pandya, and Zhou Chaochen. Finite Divergence. *Theoretical Computer Science*, 138:113–139, 1995.

[58] R. Harper, D. MacQueen, and R. Milner. Standard ML. Technical Report ECS-LFCS-86-2, Lab. f. Found. of Comp. Sci., Dept. of Comp. Sci., Univ. of Edinburgh, Scotland, 1986.

[59] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[60] Dang Van Hung and Zhou Chaochen. Probabilistic Duration Calculus for Continuous Time. Technical report, UNU/IIST Report No. 25, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1994.

[61] Mia Indrika. Formal Description of a Toll-way Control System. UNU/IIST Report: [**45**].

[62] Michael Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices.* ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995. ISBN 0-201-87712-0; xiv + 228 pages.

[63] Tomasz Janowski. Stepwise Unfolding of Transport Domain & Requirements. Technical report, UNU/IIST, 1997.

[64] C.B. Jones. *Software Development: A Rigorous Approach.* Prentice-Hall International, 1980.

[65] C.B. Jones. *Systematic Software Development — Using VDM.* Prentice-Hall International, 1986.

[66] C.B. Jones. *Systematic Software Development — Using VDM, 2nd Edition.* Prentice-Hall International, 1989.

[67] Cliff B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International, Englewood Cliffs, New Jersey, second edition, 1990. ISBN 0-13-880733-7.

[68] P. G. Larsen, B. S. Hansen, H. Brunn N. Plat, H. Toetenel, D. J. Andrews, J. Dawes, G. Parkin, et al. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language, December 1996.

[69] Dong Yu Li. Railway Station Management. Master's thesis, University of Maca, 1995.

[70] S. Mauw. Example Specifications in φSDL. Computing Science Report 96-04, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1996.

[71] R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

[72] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, Cambridge, Mass. and London, England, 1990.

[73] Carroll Morgan. *Programming from Specifications.* Prentice Hall International, 1994.

[74] Nikolai S. Nikitchenko. Towards Foundations of the General Theory of Transport Domains. UNU/IIST Report: [**88**].

[75] Søren Prehn, Dines Bjørner, and Chris George et al. Selected Papers & Reports on The RaCoSy Project. UNU/IIST Reports: [**12, 23, 24, 37, 42, 51**]; and Notes: *[16, 17, 18, 19]*; 1993–1997.

[76] The RAISE Language Group. *The RAISE Specification Language.* BCS Practitioner Series. Prentice Hall International, 1992.

[77] The RAISE Method Group. *The RAISE Development Method.* BCS Practitioner Series. Prentice Hall International, 1995.

[78] Peter Rose. *Commercial Bank Management.* Irwin Publ., Times Mirror Higher Education Group, Inc., 1995.

[79] Gueorgui Satchok. MiTraS: a Metropolitan (Mega-City) Transport System,. Technical report, UNU/IIST, 1997.

[80] S. Sokołowski. *Applicative Higher-Order Programming: the Standard ML Perspective.* Chapman and Hall, 1991.

[81] J.M. Spivey. *The Z Notation: A Reference Manual.* International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 2nd edition, 1992.

[82] UNCTAD and UNDP. 1: UNCTAD: *Effective Debt Management;* 2: UNDP: *Debt Management and the Developing Countries: A Report to the United Nations Development Programme;* 3: UNCTAD: DMFAS: *(i) The DMFAS Programme, A Brief Description;* April, 1994; *(ii) An Introduction to DMFAS 5.0 for* DMFAS 4.1 Plus *Users;* October 1994; *(iii) DMFAS 5.0 Hardware, Software and Training Requirements;* 25.07.1996; *(iv) DMFAS 5.0 Technical Information Note;* 23. November 1993; *(v) UNCTAD's DMFAS 5.0 in Brief;* 22. November 1993; *(vi) DMFAS Progress Report 1994;* 9. August 1995; *(vii) Effective Debt Management;* No date; *DMFAS Newsletters: No.8 & 9, August 1995, Spring 1996.* UNCTAD: The UN's *Conference on Tariffs and Development,* UN, Geneva, Switzerland.

[83] Tan Xinming. Enquiring about Bus Transport — Formal Development using RAISE. UNU/IIST Report: [**83**].

[84] Specification and Description Language (SDL). ITU-T Recommendation Z.100, Revision 1, 1994.

B About UNU/IIST

B.1 The UNU System

UNU/IIST is one of the may diverse RTC's (Research & Training Centres) of the UN University (UNU). UNU's Headquarter and Academic Division is located in Tokyo, Japan, and their founding and operations are significantly funded, through endowments and grants, by the Japanese Government. Thanks to these substantial Japanese contributions, the UNU has been able to manage the creation of a number of RTC's: for Development Economics Research (UNU/WIDER in Helsinki, Finland), for Socio Economic Consequences of New Technologies (UNU/INTECH in Maastricht The Netherlands), for Natural (sustainable) Resources in Africa UNU/INRA in Accra, Ghana, for Water, Environment & Health UNU/INWEH in Ontario, Canada, etc. These RTC's are all separately, usually host government funded.

B.2 Funds & Budget, Staff & Fellows

UNU/IIST is graciously endowed by the Governments of the Republic of Portugal, the People's Republic of China, and the Governor of Macau — with respectively US $5, 5 and 20 million. In addition the Governor of Macau graciously provides free premises, furniture and fixtures, as well as support wrt. fellow and visiting scholar lodgings, etc. Currently the annual UNU/IIST budget is somewhat over US $2 million. Its scientific staff numbers around 10 scientists and research engineers, and its general services staff likewise. Their salaries and all operational costs (travels, computing equipment, utilities, operations) are covered by the budget. Perhaps the most externally visible and important daily facet of UNU/IIST's life is its currently approximately 24 fellows. They come from all over the developing world and from countries whose economies are in transition. Their travel and usually 6–10 month stipends are also covered by our budget.

B.3 The UNU/IIST Charter

The UNU/IIST Charter is to help secure self-reliance wrt. software technology: its procurement, possibly own development and advanced use by institutions and individuals in developing countries and countries whose economies are in transition. This self-reliance can be decomposed into three concerns:

- Software Industry:

  the ability to develop own software to highest relevant quality;

- Universities:

  the ability to offer post-graduate computing science & software technology education to highest relevant quality; and

- Research Centres:

  the internationalization of their computing science and software technology research.

B.4 UNU/IIST's Programmatic Activities

UNU/IIST currently implements this Charter along the following lines:

1. Advanced Courses:

   UNU/IIST gives advanced computing science research and software development methodology courses worldwide. At these courses UNU/IIST staff additionally identify potential candidates for fellowships and potential joint R&D projects.

   Courses have so far been given (up to several times) in Argentina, Belarus, Brazil, China, Gabon, India, Indonesia, Japan, the North & South Koreas, Malaysia, Mongolia, the Philippines, Rumania, Russia, Turkey, Ukraine and Vietnam.

   UNU/IIST is currently planning further courses in other African (Burkina Faso, Cameroun, Ghana), Arabic World (Tunis), Asian (Pakistan), East European (Bulgaria) and Latin American countries (Costa Rica).

   Scientists trained at UNU/IIST will now give UNU/IIST courses in their home countries and on their home continents: Africa (several countries), Argentina, China, India, Vietnam, etc.

2. Fellowships:

   UNU/IIST fellows have their stipends and travel funded either by UNU/IIST, or by international aid agencies or local governments. Stipends are sufficient to cover local lodging, meals, transport, etc. as well as reasonable leisure expenses. Fellows are each provided with latest Sun Microsystems work stations, full complement of advanced software tools, full INTERNET access, etc.

   Academic staff and fellows work in a daily "Master Class" atmosphere: instruction, discussion, colloquia and seminars.

   Fellows and staff use our well-equipped library. That library currently features some 130 of the leading computer & computing science and software engineering periodicals, research and technical reports from some 220 (mostly industrial world) software research centres, and a reasonable reference handbook library. The library is open to all information technology professionals and students in Macau.

3. Research & Advanced Development Projects:

   As this report amply illustrates, UNU/IIST is engaged in a long range of very tightly coordinated: interfaced and otherwise related research and advanced development projects.

4. Events:

   Occasionally UNU/IIST [co–]sponsors events such as one to two week workshops. Examples are: (i) November 1993: University Curricula in Software Technology (one week), (ii) January 1994: IFIP WG2.3 Seminar (two weeks); (iii) February–March 1996: Software for Sustainable Development, see special appendix section E.12 (starting page 183) [42, 24]; and (iv) September 1996: IFIP WG2.2 Meeting + Seminar (one + one weeks). These and other events feature leading, international scientists.

   Future events include regular symposia held with Chinese institutions under the State Science & Technology Commission (responsible for the Chinese industry), the State Education Commission (responsible for presently 1034 Chinese universities), the Chinese Academy of Sciences (responsible for some 123 research institutes), and as co-sponsored by the National Natural Science Foundation of China. Future such symposia are expected to also include similarly originating and sponsored participation from other Asian countries.

5. Dissemination:

   Through its advanced courses, fellowships, R&D projects, and events UNU/IIST distributes knowledge worldwide. Additionally UNU/IIST has recently begun distribution of a compendium, [49], of the most important results of the last five years of fellowship projects. This five volume, almost 2000 page document, can be seen as a non-personal "expanded" version of the current report.

   UNU/IIST also helps establish libraries at now ten of its affiliate centres (China (3), India (2), Mongolia, Russia (2), Ukraine, and Vietnam). UNU/IIST has been able to secure a US $ 25,000 full complement Sun Microsystems installation at Mongolian National University (to share with the Academy of Sciences and the Technical University of Mongolia) — as part of the joint MultiScript project. We are most grateful to Sun Microsystems.

B.5 The UNU/IIST Network

UNU/IIST currently works with institutions in Argentina (Univs. of San Luis, Bahia Blanca and Buenos Aires), Brazil (PUC-Parana, PUC-Rio/LMF, CITS), Burkina Faso (Govt. Informatics Comm.), China (three Research Institutes of Academia Sinica [Beijing and Chengdu], and some 20 universities), Gabon (IAI), Hong Kong (HKU, CUHK, BUHK, HKPU), India (two IIT's, IIS, TIFR, SPIC SM, some Universities, TRDDC, etc.), Indonesia (UI, Jakarta), Iran (Tabat Modarres Univ., ITRC), Japan (JAIST, Kyushu/ISIS, etc.), North Korea [DPRK] (Science & Technology Commission), South Korea [RoK] (Postech/CAST), Macau (Univ. of Macau), Malaysia (Penang, Kushing), Mongolia (MNU), the Philippines (DLSU, UoP, AIST), Portugal (UNL, UM), Rumania (Univ. of Bucharest), Russia (ISP/RAS, IPI/RAS, Ershov Inst.), Ukraine (Kiev Univ.), and Vietnam (IoIT, Hanoi Univ., Min. of Finance, Vietnam Airlines). This network is expanding.

C The Design Calculi & Formal Methods

We cover four aspects:

1. DC: The Duration Calculi — Appendix C.1
2. SDL: Specification and Description Language — Appendix C.2
3. RAISE: A Formal Software Engineering Method — Appendix C.3
4. LSD: Logical Systems Development — Appendix C.4

C.1 DC: The Duration Calculi

We[24] refer to item 1 on page 12. In this special appendix section we will review a few of the Duration Calculi.

DC represents a logical approach to formal design of real-time systems. DC is based on interval logic, and uses the real numbers to model time, and Boolean-valued (i.e. $\{0, 1\}$-valued) functions over time to model states and events of real-time systems. The duration of a state in a time interval is the accumulated presence time (i.e. the integral) of the state in the interval. DC extends interval logic with a calculus to specify and reason about properties of state durations.

The research of DC was introduced in the ProCoS project (ESPRIT BRA 3104 and 7071) [21], when the project was investigating formal techniques for designing safety critical real-time systems. In a project case study, it was realised that state duration had not been well studied yet as an essential measurement of real-time behaviour of computing systems. A research of a logic for state durations was therefore initiated by the project in 1990, with the first paper on DC by Zhou Chaochen, C.A.R. Hoare and A.P. Ravn [30] published in 1991.

So far, a whole family of real time models have been formalised in Duration Calculi.

- The Duration Calculus:

  The basic calculus of DC [30] axiomatises for the Boolean state model state durations, i.e. integrals of Boolean-valued functions, under an assumption of *finite variability* (also called *non-Zeno phenomenon*) of states. The finite variability stipulates that any state can only change its presence and absence finitely many times in any bounded time period. This calculus can be used to specify and verify state based safety properties of real-time systems. Formalisations of other models are *conservative* extensions of this calculus.

[24] This section is written by Zhou Chao Chen, Dang Van Hung and Xu Qi Wen

- The Mean Value Duration Calculus:

  This studies Boolean state and event model, where an event is a Boolean-valued δ-function, i.e. a Boolean-valued function with value of 1 at points. In order to describe point properties in DC, The Mean Value Duration Calculus [32] replaces integrals of Boolean-valued functions by their mean values. By relating events to state transitions, this calculus can be used to refine from state based requirements via mixed state and event specifications to event based specifications or programs.

- The Extended Duration Calculus:

  The Extended Duration Calculus investigates real state model of real-time systems. A real state model consists of a set of real-valued functions, which describe behaviour of physical components of a software embedded system. Therefore specifications and reasonings at the level of real state may have to employ real analysis. The Extended Duration Calculus suggests an approach to combine DC with real analysis (including differential equations), so that real state models can be specified in the framework of DC.

- Dependability:

  The dependability of a requirement for an implementation can be quantitatively measured by the satisfaction probability of the requirement for this implementation. With respect to the Boolean state model, the Probabilistic Duration Calculus provides designers with a set of rules to reason about and calculate whether a given requirement written in DC will hold with a sufficient high probability for given failure probabilities of the components used in an implementation, where implementation with imperfect components is taken to be a finite automaton with history-independent transition probabilities in discrete time domain. [60] generalises this work for continuous time domain.

- Finite Divergence Model:

  The finite variability of states and events stipulates that within a finite time period state transitions and events can happen only finitely many times. The finite variability assumption is always adopted by software systems where time is progressing discretely. However the assumption of finite variability may be violated in a software embedded system where time is continuous. The notion opposite to the finite variability is called *finite divergence* (also called *Zeno phenomenon*). [57] formalises the finite divergence model by introducing into DC the rules to calculate a state duration in a finite divergence model as a limit of its approximations in finite variability model.

- Super Dense Computation:

  A *super dense computation* is a sequence of operations which is assumed to be timeless. It is an abstraction of a real-time computation within a context with a grand time granularity. This assumption has been adopted by digital control system, where the cycle time of an embedded computer may be nanoseconds, while the sampling period of a controller may take seconds. Therefore computation time of embedded software of digital control system is negligible, and computational operations can be abstracted as timeless actions. [28] adapts the chop modality (renamed as *super dense chop*), so that it can map an operation in a grand time space into a time space with finer time granularity. Thus, properties of super dense computation can also be specified and verified in DC.

- Expanding Modalities: With contracting modalities one can only specify *safety* properties of real-time systems. In order to specify *unbounded liveness* and *fairness* properties of real-time systems in DC, a number of researchers have proposed to introduce into DC expanding modalities. [29] introduces the left and right neighbourhood modalities and shows that they are adequate in the sense that the other contracting and expanding modalities can be derived from them in a first order logic with the interval length ℓ. [29] establishes a complete first order calculus for the left and right neighbourhood modalities and demonstrates how expanding modalities are also convenient for formulating notions of real analysis.

- Infinite Intervals:

  Time is modelled by the infinite set of the real numbers, and a behaviour of a real-time system is modelled by a set of Boolean-valued or real-valued functions over time. Thus, behaviour of real-time systems is assumed *infinite*. However the interval logic based on which DC is established is a logic of finite intervals. An infinite behaviour is therefore specified in DC as a set of all finite prefixes of the behaviour. To specify liveness and fairness properties of a system behaviour in terms of its finite prefixes, people introduces expanding modalities. However an alternative to expanding modalities is to introduce into DC infinite intervals. An extension of DC which allows infinite intervals is described in [31]. In this extension, a state duration over an infinite interval is determined by a property which specifies the limit of the state duration over finite intervals. Since the extension includes both finite and infinite intervals, *terminating* and *infinite* system behaviour can be straightforwardly expressed.

We will briefly explain three Duration Calculi. We first review the Interval Logic on which the three Duration Calculi are based.

We adopt the following proof system of Interval Logic (IL) [39]. To formulate the axioms and inference rules, we need the standard notion of free (global) variables, whose meaning is given by a *value assignment*, which is a function associating a real number with each global variable:

$$\mathcal{V} \in GVar \to \mathcal{R}$$

Temporal variables and propositional letters are "interval dependent symbols", and their semantics is defined by an interpretation:

$$\mathcal{J} \in \begin{pmatrix} TVar \\ \cup \\ PLetters \end{pmatrix} \to \begin{pmatrix} \mathcal{I}\mathrm{ntv} \to \mathcal{R} \\ \cup \\ \mathcal{I}\mathrm{ntv} \to \{\mathrm{tt},\mathrm{ff}\} \end{pmatrix}$$
$$\text{where } \mathcal{I}\mathrm{ntv} \mathrel{\widehat{=}} \{[b,e] \mid b,e \in \mathcal{R} \text{ and } b \le e\}$$

associating a real-valued interval function with each temporal variable and a Boolean-valued interval function with each temporal propositional letter.

The semantics of the basic modality, $\frown$ (chop) is defined as follows

$$\begin{array}{l} \mathcal{J}, \mathcal{V}, [b,e] \models \phi \frown \psi \\ \text{iff } \mathcal{J}, \mathcal{V}, [b,m] \models \phi \text{ and } \mathcal{J}, \mathcal{V}, [m,e] \models \psi, \text{ for some } m \in [b,e] \end{array}$$

The following abbreviations will be used:

$$\begin{array}{lcll} \Diamond\phi & \widehat{=} & \mathbf{True}\frown(\phi\frown\mathbf{True}) & \text{reads: "for some sub-interval: } \phi\text{"} \\ \Box\phi & \widehat{=} & \neg\Diamond(\neg\phi) & \text{reads: "for all sub-intervals: } \phi\text{"} \end{array}$$

A term (formula) is called *flexible* if a temporal variable, the symbol ℓ or a propositional letter occurs in the term (formula). A term or formula which is not flexible is also called *rigid*.

The axioms of IL are:

$$\begin{array}{ll}
\mathrm{A0}: & \ell \geq 0 \\
\mathrm{A1}: & \begin{array}{l} ((\phi\frown\psi) \wedge \neg(\phi\frown\varphi)) \;\Rightarrow\; (\phi\frown(\psi \wedge \neg\varphi)) \\ ((\phi\frown\psi) \wedge \neg(\varphi\frown\psi)) \;\Rightarrow\; ((\phi \wedge \neg\varphi)\frown\psi)) \end{array} \\
\mathrm{A2}: & ((\phi\frown\psi)\frown\varphi) \iff (\phi\frown(\psi\frown\varphi)) \\
\mathrm{R}: & \begin{array}{l} (\phi\frown\psi) \Rightarrow \phi \quad \text{if } \phi \text{ is a rigid formula} \\ (\phi\frown\psi) \Rightarrow \psi \quad \text{if } \psi \text{ is a rigid formula} \end{array} \\
\mathrm{B}: & \begin{array}{l} ((\exists x)\phi\frown\psi) \Rightarrow (\exists x)(\phi\frown\psi) \quad \text{if } x \text{ is not free in } \psi \\ (\phi\frown(\exists x)\psi) \Rightarrow (\exists x)(\phi\frown\psi) \quad \text{if } x \text{ is not free in } \phi \end{array} \\
\mathrm{L1}: & \begin{array}{l} ((\ell = x)\frown\phi) \Rightarrow \neg((\ell = x)\frown\neg\phi) \\ (\phi\frown(\ell = x)) \Rightarrow \neg(\neg\phi\frown(\ell = x)) \end{array} \\
\mathrm{L2}: & (x \geq 0 \wedge y \geq 0) \Rightarrow ((\ell = x + y) \iff ((\ell = x)\frown(\ell = y))) \\
\mathrm{L3}: & \phi \Rightarrow (\phi\frown(\ell = 0)) \\
\mathrm{L4}: & \phi \Rightarrow ((\ell = 0)\frown\phi)
\end{array}$$

The inference rules of IL are:

$$\begin{array}{ll}
\mathrm{MP}: & \text{if } \phi \text{ and } \phi \Rightarrow \psi \text{ then } \psi \text{ (modus ponens)} \\
\mathrm{G}: & \text{if } \phi \text{ then } (\forall x)\phi \text{ (generalisation)} \\
\mathrm{N1}: & \text{if } \phi \text{ then } \neg(\neg\phi\frown\psi) \\
\mathrm{N2}: & \text{if } \phi \text{ then } \neg(\psi\frown\neg\phi) \\
\mathrm{M1}: & \text{if } \phi \Rightarrow \psi \text{ then } (\phi\frown\varphi) \Rightarrow (\psi\frown\varphi) \\
\mathrm{M2}: & \text{if } \phi \Rightarrow \psi \text{ then } (\varphi\frown\phi) \Rightarrow (\varphi\frown\psi)
\end{array}$$

The inference rules N1 and N2 are called rules of necessitation, and the inference rules M1 and M2 are the monotony rules for chop. The inference rule G is the standard generalisation rule from first order logic.

The proof system also contains axioms of first order predicate logic with equality.

1. The Duration Calculus:

   We establish DC as an extension of IL in the sense that temporal variables $v \in TVar$ have a structure
$$\int S$$
   where S is called a *state expression* and is generated from a set $SVar$ of *state variables* $P, Q, R, \ldots$, according to the following abstract syntax:
$$S \quad ::= \quad 0 \mid 1 \mid P \mid \neg S_1 \mid S_1 \vee S_2$$
   We will use the same abbreviations for propositional connectives in state expressions as introduced for IL formulas.

   When we generate temporal variables from state variables, the semantics of temporal variables must be derived from the semantics of the state variables. To this end we introduce an *interpretation for state variables* (and propositional letters) as a function:
$$\mathcal{I} \in \begin{pmatrix} SVar \\ \cup \\ PLetters \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{Time} \rightarrow \{0,1\} \\ \cup \\ \mathcal{I}\mathrm{ntv} \rightarrow \{\mathrm{tt},\mathrm{ff}\} \end{pmatrix}$$
   where $\mathcal{I}(P) \in \mathbf{Time} \rightarrow \{0,1\}$, $\mathcal{I}(X) \in \mathcal{I}\mathrm{ntv} \rightarrow \{\mathrm{tt},\mathrm{ff}\}$, and each function $\mathcal{I}(P)$ has at most a finite number of discontinuity points in every interval $[b, e]$, an analytic definition of the finite variability of states, hence $\mathcal{I}(P)$ is integrable in every bounded interval.

   The semantics of a state expression S, given an interpretation $\mathcal{I}$, is a function
$$\mathcal{I}[\![S]\!] \in \mathbf{Time} \rightarrow \{0,1\}$$
   defined inductively on the structure of state expressions by:
$$\begin{array}{lcl} \mathcal{I}[\![0]\!](t) & = & 0 \\ \mathcal{I}[\![1]\!](t) & = & 1 \\ \mathcal{I}[\![P]\!](t) & = & \mathcal{I}(P)(t) \\ \mathcal{I}[\![(\neg S)]\!](t) & = & 1 - \mathcal{I}[\![S]\!](t) \\ \mathcal{I}[\![(S_1 \vee S_2)]\!](t) & = & \begin{cases} 0 & \text{if } \mathcal{I}[\![S_1]\!](t) = 0 \text{ and } \mathcal{I}[\![S_2]\!](t) = 0 \\ 1 & \text{otherwise} \end{cases} \end{array}$$
   We shall use the abbreviation $S_{\mathcal{I}} \mathrel{\hat=} \mathcal{I}[\![S]\!]$. We see by this semantics that each function $S_{\mathcal{I}}$ has at most a finite number of discontinuity points in any interval $[b, e]$ and is thus integrable in every bounded interval.

   The semantics of temporal variables, which now have the form $\int S$, is given by a function $\mathcal{I}[\![\int S]\!] \in \mathcal{I}\mathrm{ntv} \rightarrow \mathcal{R}$ defined by:
$$\mathcal{I}[\![\int S]\!][b, e] \quad = \quad \int_b^e S_{\mathcal{I}}(t)dt$$

This function can be used to induce an interpretation $\mathcal{J}_{\mathcal{I}}$ for temporal variables v of the form $\int S$ and propositional letters from $\mathcal{I}$:

$$\begin{array}{lcll} \mathcal{J}_{\mathcal{I}}(X) & = & \mathcal{I}(X) & \text{for every propositional letter } X \\ \mathcal{J}_{\mathcal{I}}(v) & = & \mathcal{I}[\![\int S]\!] & \text{when } v \text{ is } \int S \end{array}$$

The semantics of a duration calculus formula ϕ, given an interpretation $\mathcal{I}$ to state variables, is a function:

$$\mathcal{I}[\![\phi]\!] \in \mathit{Val} \times \mathcal{I}\mathrm{ntv} \to \{\mathrm{tt},\mathrm{ff}\}$$

for which we use the abbreviations:

$$\begin{array}{lcl} \mathcal{I}, \mathcal{V}, [b,e] \models_{dc} \phi & \widehat{=} & \mathcal{I}[\![\phi]\!]\ (\mathcal{V}, [b,e]) = \mathrm{tt} \\ \mathcal{I}, \mathcal{V}, [b,e] \not\models_{dc} \phi & \widehat{=} & \mathcal{I}[\![\phi]\!]\ (\mathcal{V}, [b,e]) = \mathrm{ff} \end{array}$$

The function can be defined as follows:

$$\mathcal{I}, \mathcal{V}, [b,e] \models_{dc} \phi \quad \text{iff} \quad \mathcal{J}_{\mathcal{I}}, \mathcal{V}, [b,e] \models \phi$$

The notions of satisfiability and validity of DC formulas are defined as for IL formulas.

Since DC is an extension of IL we adopt all axioms and inference rules of IL from the previous section as axioms and inference rules for DC. Furthermore, we add axioms reflecting the structure which DC adds to temporal variables:

(DC-A1) $\int 0 = 0$

(DC-A2) $\int 1 = \ell$

(DC-A3) $\int S \geq 0$

(DC-A4) $\int S_1 + \int S_2 = \int (S_1 \vee S_2) + \int (S_1 \wedge S_2)$

(DC-A5) $((\int S = x)^\frown(\int S = y)) \Rightarrow (\int S = x + y)$

(DC-A6) $\int S_1 = \int S_2$, provided $S_1 \iff S_2$ holds in propositional logic

Furthermore, we add two induction rules to formalise the finite variability of states:

IR1: Let $H(X)$ be a formula possibly containing the propositional letter X, and let S be any state expression.

$$\begin{array}{ll} \text{If} & H(\lceil\!\lceil\ \rceil\!\rceil) \text{ and } H(X) \Rightarrow H(X \vee (X^\frown\lceil\!\lceil S \rceil\!\rceil) \vee (X^\frown\lceil\!\lceil \neg S \rceil\!\rceil)) \\ \text{then} & H(\mathbf{True}) \end{array}$$

where $H(\phi)$ denotes the formula obtained from $H(X)$ by replacing every occurrence of X in H with ϕ.

IR2: Let $H(X)$ be a formula possibly containing the propositional letter X, and let S be any state expression.

$$\begin{array}{ll} \text{If} & H(\lceil\!\lceil\ \rceil\!\rceil) \text{ and } H(X) \Rightarrow H(X \vee (\lceil\!\lceil S \rceil\!\rceil^\frown X) \vee (\lceil\!\lceil \neg S \rceil\!\rceil^\frown X)) \\ \text{then} & H(\mathbf{True}) \end{array}$$

In these rules $H(\lceil\!\lceil\ \rceil\!\rceil)$ is called the *base case*, and $H(X)$ is called the *induction hypothesis*.

2. The Extended Duration Calculus:

   To describe the physical component of a software embedded system, the model should also contain real-valued functions, such as pressure p and the rate that the pressure changes $\dot{p}$. Formally, we extend the interpretation, so that it now also maps function variables to real numbers

$$\mathcal{I} \in \begin{pmatrix} SVar \\ \cup \\ PLetters \\ \cup \\ FVar \end{pmatrix} \to \begin{pmatrix} \mathbf{Time} \to \{0,1\} \\ \cup \\ \mathcal{I}\mathrm{ntv} \to \{\mathrm{tt},\mathrm{ff}\} \\ \cup \\ \mathbf{Time} \to \mathcal{R} \end{pmatrix}$$

   Real-valued functions can be compared by relational operators of real arithmetics to form the *relations*, e.g. $f < g$. A relation A denotes a boolean valued function of time: A : $\mathbf{Time} \to \{0,1\}$. This function is also defined point-wisely, e.g. $(f < g)(t) = 1$ iff $f(t) < g(t)$, describing a property of the system over the time. The length that such a property holds (over an interval) is denoted by the duration $\int(f < g)$. As in Duration Calculus, an atomic duration formula can now be formed by comparing reals and durations, and duration formulas are formed from the atomic ones using the connectives of predicate logic.

   A particularly useful formula characterises that a certain property H holds all the time in a non-empty interval, and this is denoted by $\llbracket H \rrbracket$. For example, $\llbracket Continuous(f) \rrbracket$ says that f is continuous at every point of the interval. In addition, two interval functions (actually functionals) $\mathbf{b}.(\cdot)$ and $\mathbf{e}.(\cdot)$ are introduced. Applied to a (piecewise) continuous function f on a non-empty interval $\mathbf{b}.f$ ($\mathbf{e}.f$) gives the value of f at the beginning (end) of the interval (Strictly speaking they give the limit from the right and left of the respective end points. Their values on empty intervals are assigned to a constant).

   EDC is an extension of DC. It also contains the following axioms which provide an interface to control theory

$$\llbracket\ \rrbracket \Rightarrow \mathbf{b}.f = \mathbf{e}.f$$

$$\mathbf{b}.f = r_1 \wedge ((\mathbf{b}.f = r_2 \wedge \ell > 0)^\frown true) \Rightarrow r_1 = r_2$$

$$\mathbf{e}.f = r_1 \wedge (true^\frown(\ell > 0 \wedge \mathbf{e}.f = r_2)) \Rightarrow r_1 = r_2$$

$$\llbracket Continuous(f) \rrbracket \wedge (\ell > 0 \wedge \mathbf{e}.f = r_1)^\frown(\mathbf{b}.f = r_2 \wedge \ell > 0) \Rightarrow r_1 = r_2$$

$$\llbracket Continuous(f) \rrbracket \Rightarrow (\llbracket H(f) \rrbracket \iff \neg((\ell > 0 \wedge \neg H(\mathbf{e}.f))^\frown \ell > 0))$$

   Mathematical theorems, denoted by $\vdash_{MT}$, can be introduced by the following axiom

Let $R(\llbracket H(f) \rrbracket,\ \mathbf{b}.f,\ \mathbf{e}.f,\ \ell)$ be a formula without "chop" ($\frown$).
If $\vdash_{MT} \forall c < d \bullet R(\forall t \in (c,d) \bullet H(f(t)),\ f(c^+),\ f(d^-),\ d-c)$ then

$$\llbracket\ \rrbracket \vee R(\llbracket H(f) \rrbracket, \mathbf{b}.f, \mathbf{e}.f, \ell)$$

3. The Mean Value Duration Calculus:

In order to be able to refine a system requirement in terms of states into a set of system components which coordinate their behaviour by communications, the Duration Calculus is extended with point values of Boolean functions such that Boolean δ-functions to represent instant actions. This extended calculus is called Mean Value Calculus. The Mean Value calculus uses mean value of Boolean function F (denoted $\overline{F}$) over intervals to replace the integral of F, $\int F$, in the original calculus. The integral over a point interval always degenerates to zero value, while the mean value of a function over a point interval delivers the value of the function at that point

$$\mathcal{I}\llbracket\overline{F}\rrbracket[b,e] = \begin{cases} \int_b^e F_{\mathcal{I}}(t)dt/(e-b) & \text{if } e > b \\ F_{\mathcal{I}}(b) & \text{if } e = b \end{cases}$$

$\int F$ can be defined by $\overline{F}$

$$\int F \mathrel{\widehat{=}} \overline{F} * l$$

A relatively *complete* inference system for the Mean Value Calculus is established by replacing $\int F$ with $\overline{F} * l$ in the axioms of the Duration Calculus.

A Boolean δ-function E is a function which satisfies

$$\neg\Diamond\llbracket E \rrbracket$$

That is, any Boolean δ-function will not last its presence for a non-zero period. Instant actions can then be represented by Boolean δ-functions in the Mean Value Calculus.

Events cause state transitions of a system. State transition from S_1 to S_2 determines the system states in a neighbourhood of the transition. That is, right before the transition the system stays at S_1 and right after the transition it stays at S_2. The transition can be expressed by the conjunction of δ-functions

$$\downarrow S_1 \wedge \uparrow S_2$$

where $\downarrow S_1$ represents *falling edge* of S_1 and $\uparrow S_2$ represents the *rising edge* of S_2.

In topology, neighbourhood properties of functions are characterised by so-called *function germs*. A function germ at a point is an equivalence class of functions which have the same value in an arbitrarily small neighbourhood of the point. States are Boolean functions without *isolated* points, so there are only four interesting state germs at any point. For a point t, they are

$\mathcal{G}_1(t)$: $\{S | \exists\delta\forall x : (0,\delta).S(t-x) = 0 \wedge S(t+x) = 1\}$,
$\mathcal{G}_2(t)$: $\{S | \exists\delta\forall x : (0,\delta).S(t-x) = 1 \wedge S(t+x) = 0\}$,

$\mathcal{G}_3(t)$: $\{S | \exists \delta \forall x : (0,\delta).S(t-x) = 0 \wedge S(t+x) = 0\}$, and
$\mathcal{G}_4(t)$: $\{S | \exists \delta \forall x : (0,\delta).S(t-x) = 1 \wedge S(t+x) = 1\}$.

$\uparrow S$ becomes a state 'germship', which discriminates the membership of S for germ $\mathcal{G}_1$: $\uparrow S(t) = 1$ iff $S \in \mathcal{G}_1(t)$. Similarly $\downarrow S(t) = 1$ means $S \in \mathcal{G}_2(t)$. The other two germships corresponding to $\mathcal{G}_3$ and $\mathcal{G}_4$ are denoted $\top S$ (*keep high*) and $\bot S$ (*keep low*). The state transition $\downarrow S_1 \wedge \uparrow S_2$ becomes a *composite* state germship.

The behaviour of an automaton is determined by the correspondence between its events and state transitions. It can be specified in the Mean Value Calculus with formulas over events and state germships. For example, let E_1 and E_2 be δ-functions to stand for two events. Then

$$(\downarrow S_1 \wedge \uparrow S_2) \Rightarrow (E_1 \vee E_2)$$

specifies that E_1 and E_2 are the only events which may drive an automaton from S_1 to S_2.

The state germships can be easily defined in the Mean Value Calculus. The axioms to define $\uparrow S$ and $\downarrow S$ are

$$(l > 0)^\frown \uparrow S^\frown (l > 0) \Leftrightarrow \mathit{true}^\frown \lceil\!\lceil \neg S \rceil\!\rceil^\frown \lceil\!\lceil S \rceil\!\rceil^\frown \mathit{true}$$
$$(l > 0)^\frown \downarrow S^\frown (l > 0) \Leftrightarrow \mathit{true}^\frown \lceil\!\lceil S \rceil\!\rceil^\frown \lceil\!\lceil \neg S \rceil\!\rceil^\frown \mathit{true}$$

The axioms for $\top S$ and $\bot S$ are similar.

A calculus for the germships follows. For any point intervals

(a) Completeness & Exclusiveness
$\bigvee_\alpha \alpha P$ $\qquad \alpha \in \{\uparrow, \downarrow, \top, \bot\}$
$\bigwedge_{\alpha \neq \alpha'} (\alpha P \Rightarrow \neg \alpha' P)$ $\qquad \alpha, \alpha' \in \{\uparrow, \downarrow, \top, \bot\}$

(b) Constant Zero
$\neg \uparrow 0$
$\neg \downarrow 0$
$\neg \top 0$
$\bot 0$

(c) Negation
$\uparrow \neg P \Leftrightarrow \downarrow P$
$\downarrow \neg P \Leftrightarrow \uparrow P$
$\top \neg P \Leftrightarrow \bot P$
$\bot \neg P \Leftrightarrow \top P$

(d) Conjunction
$\uparrow (P \wedge Q) \Leftrightarrow (\uparrow P \wedge \top Q) \vee (\uparrow Q \wedge \top P) \vee (\uparrow P \wedge \uparrow Q)$
$\downarrow (P \wedge Q) \Leftrightarrow (\downarrow P \wedge \top Q) \vee (\downarrow Q \wedge \top P) \vee (\downarrow P \wedge \downarrow Q)$
$\top (P \wedge Q) \Leftrightarrow (\top P \wedge \top Q)$
$\bot (P \wedge Q) \Leftrightarrow \bot P \vee \bot Q \vee (\uparrow P \wedge \downarrow Q) \vee (\downarrow P \wedge \uparrow Q)$

C.2 SDL: Specification and Description Language

SDL[25] is an acronym for Specification and Description Language.

SDL [8, 84] is widely used for specification and design of telecommunications systems. This section gives an overview of SDL. We focus on the part of SDL that deals with the behavioural aspects of systems. This means that structural concepts such as blocks are not covered. Besides, some details are left out to improve the comprehensibility of the overview. First of all, we explain this part of SDL informally. After that, we give some formal details: we outline the semantics of SDL, based on discrete time process algebra [4, 5], presented in [12].

Informal description

In the SDL view, a system basically consists of *processes* which communicate with each other and the environment by sending and receiving *signals* via *signal routes*. A process proceeds in parallel with the other processes in the system and communicates with these processes in an asynchronous manner. This means that a process sending a signal does not wait until the receiving process consumes it, but it proceeds immediately. A process may also use local *variables* for storage of values. A variable is associated with a value that may change by assigning a new value to it. A variable can only be assigned new values by the process to which it is local, but it may be viewed by other processes.

A signal can be sent from the environment to a process, from a process to the environment or from a process to a process. A signal may carry values to be passed from the sender to the receiver; on consumption of the signal, these values are assigned to local variables of the receiver. A signal route is a unidirectional connection between two processes, or between one process and the environment, for conveying signals. If a signal is sent to a process via a signal route, it is instantaneously delivered to that process.

A system description $\mathsf{system}S;D_1 \ldots D_n\mathsf{endsystem};$ contains definitions $(D_1, \ldots, D_n)$ of the processes present in the system, the local variables used by the processes for storage of values, the types of signals used by the processes for communication and the signal routes via which the signals are conveyed.
A process definition $\mathsf{process}X;\mathsf{start};tr\ d_1 \ldots d_n\mathsf{endprocess};$ defines a process X which will be created during the start-up of the system. The process body $\mathsf{start};tr\ d_1 \ldots d_n$ describes the behaviour of the process X in terms of states and transitions (see next paragraph). It will start by making the transition tr, called its start transition, to enter one of its states. The state definitions $d_1, \ldots, d_n$ define all the states in which the process may come while it proceeds.

In the SDL view, a process is either in a *state* or making a *transition* to another state. Besides, when a signal arrives at a process, it is put into the unique *input queue* associated with the

[25] This section is written by Kees Middelburg

process until it is consumed by the process. The states of a process are the points in its behaviour where a signal may be consumed. However, a state may have signals that have to be saved, i.e. withhold from being consumed in that state. The signal consumed in a state of a process is the first one in its input queue that has not to be saved for that state. If there is no signal to consume, the process waits until there is a signal to consume. So if a process is in a state, it is either waiting to consume a signal or consuming a signal.

A transition from a state of a process is initiated by the consumption of a signal, unless it is a spontaneous transition. The start transition is not initiated by the consumption of a signal. A transition is made by performing certain actions: signals may be sent, variables may be assigned new values and *timers* may be set and reset. A transition may at some stage also take one of a number of branches, but it will eventually come to an end and bring the process to a next state or to its termination.

A timer can be set which sends at its expiration time a signal to the process setting it. A timer is identified with the type and carried values of the signal it sends on expiration. Thus an active timer can be set to a new time or reset; if this is done between the sending of the signal noticing expiration and its consumption, the signal is removed from the input queue concerned. A timer is de-activated when it is reset or the signal it sends on expiration is consumed.

The value of expressions in SDL may vary according to the last values assigned to variables, including local variables of other processes. It may also depend on the system state, e.g. on timers being active or the system time.

A state definition $\mathsf{state}\,st;\mathsf{save}\,s_1,\ldots,s_m;alt_1\,\ldots\,alt_n$ defines a state st. The signals of the types $s_1,\ldots,s_m$ are saved for the state. Each transition alternative $alt_1,\ldots,alt_n$ consists of an input guard and a transition. The input guard gives a type of signals that may be consumed in the state; the corresponding transition is the one that is initiated on consumption of a signal of that type. An alternative with inputnone; instead of an input guard is a transition that may be made spontaneously from the state.
An input guard $\mathsf{input}\,s(v_1,\ldots,v_n)$; may consume a signal of type s and, on consumption, it assigns the carried values to the local variables $v_1,\ldots,v_n$. If the signals of type s carry no value, $(v_1,\ldots,v_n)$ is left out.
A transition $a_1\,\ldots\,a_n\,\mathsf{nextstate}\,st$; performs the actions $a_1,\ldots,a_n$ in sequential order and ends with entering the state st. Replacing $\mathsf{nextstate}\,st$ by stop yields a transition ending with process termination. It can also be replaced by a *decision* for transfer of control to one of two or more transition branches.

Formal details

Process algebra can be used as the basis for the semantics of SDL. We will make use of ACP, introduced in [11], extended with the silent step τ and the abstraction operator τ_I for abstraction. Semantically, we adopt the approach to abstraction, originally proposed for ACP in [53]. In ACP with abstraction, processes can be composed by sequential composition, written $P \cdot Q$, alternative composition, written $P + Q$, parallel composition, written $P \parallel Q$, encapsulation, written $\partial_H(P)$, and abstraction, written $\tau_I(P)$. Additionally, we will use the state operator λ_G, added to ACP in [2], and the conditional operator $:\rightarrow$, added to ACP in [3]. $\lambda_G(P)$ stands for the process P executed in a state G. $b :\rightarrow P$ is the process P if the condition b equals true and δ (the process that can neither perform an action nor terminate successfully) otherwise. We also need a discrete time extension of ACP with relative timing. We will make use of the extension introduced in [5], called $\mathrm{ACP_{drt}}$, with abstraction as added to it in [4]. We will use the constants $\underline{\underline{a}}$ (one for each action a), as well as the delay operator σ_{rel}. The process $\underline{\underline{a}}$ is a performed in the current time slice. The process $\sigma_{\mathsf{rel}}(P)$ is P delayed one time slice. For a systematic introduction to process algebra in the form of ACP, the reader is referred to [6].

Using process algebra as the basis for the semantics of SDL, the meaning of a system description can be represented by a triple $(P, E, \mathcal{G})$ where:

1. P is a process expression describing the behaviour of the system;
2. E is a set of equations recursively defining the processes referred to in P;
3. $\mathcal{G}$ is the state space to be associated with a state operator (λ_G) used in P to formalise execution of a process in a state G.

The meaning of each process definition is a set of equations recursively defining the process concerned. The meaning of a process definition is expressed in terms of the meaning of its start transition and its state definitions.

$$
\begin{aligned}
&[\![\mathsf{system}\, S; D_1 \ldots D_n \,\mathsf{endsystem};]\!] \;:=\; (\tau_{I \cup \{\boldsymbol{t}\}}(\lambda_{G_0}(P \parallel \mathit{Env})), [\![D_1]\!]^{\kappa} \cup \ldots \cup [\![D_n]\!]^{\kappa}, \mathcal{G}_{\kappa}) \\
&\quad \text{where } \begin{array}{lll} P & = & \|_{X \in \mathit{procs}(\kappa)}\, X, \\ G_0 & = & (0, \{(X, (\{\,\}, \langle\rangle, \{\,\})) \mid X \in \mathit{procs}(\kappa)\}), \\ \kappa & = & \{\![\mathsf{system}\, S; D_1 \ldots D_n \,\mathsf{endsystem};]\!\} \end{array} \\
&[\![\mathsf{process}\, X; \mathsf{start}; \mathit{tr}\; d_1 \ldots d_n \,\mathsf{endprocess};]\!]^{\kappa} \;:=\; \{X \;=\; [\![\mathit{tr}]\!]^{\kappa'}, [\![d_1]\!]^{\kappa'}, \ldots, [\![d_n]\!]^{\kappa'}\} \\
&\quad \text{where } \kappa' \;=\; \mathit{updscopeunit}(\kappa, X) \quad .
\end{aligned}
$$

The process expression Env and the set of actions I are to be regarded as parameters of the semantics.

The process expression $\tau_{I \cup \{\boldsymbol{t}\}}(\lambda_{G_0}(P \parallel \mathit{Env}))$ expresses that the processes defined are executed in parallel, starting in the state G_0, while they are receiving signals via signal routes from the environment Env. Additionally, the internal action $\boldsymbol{t}$ as well as the actions in I are hidden. G_0 is the state in which the system time is zero and each of the processes defined has an initial local state where no variable has a value associated with it, the input queue is empty and there are no active timers. The equation $X = [\![\mathit{tr}]\!]^{\kappa'}$ describes that the process X behaves initially as the start transition tr, while the equations $[\![d_1]\!]^{\kappa'}, \ldots, [\![d_n]\!]^{\kappa'}$ describe how the process X behaves from each of the n states in which it may come while it proceeds.

The meaning of a state definition, occurring in the scope of a process definition, is an equation defining, for the process concerned, its behaviour from the state being defined. It is expressed in terms of the meaning of its transition alternatives. The meaning of each transition alternative is in turn expressed in terms of the meaning of its input guard, if the alternative is not a spontaneous transition, and its transition.

$$
\begin{array}{l}
[\![\mathsf{state}\, st;\mathsf{save}\, s_1,\ldots,s_m;alt_1\ \ldots\ alt_n]\!]^{\kappa} := \\
\quad X_{st} = [\![alt_1]\!]^{\kappa'} + \ldots + [\![alt_n]\!]^{\kappa'} + waiting(s_1,\ldots,s_m,X) :\!\rightarrow \sigma_{\mathsf{rel}}(X_{st}) \\
\quad \text{where } X = scopeunit(\kappa),\ \kappa' = updsaveset(\kappa,\{s_1,\ldots,s_m\}) \\
[\![\mathsf{input}\, s(v_1,\ldots,v_n);tr]\!]^{\kappa} := \underline{\underline{input}}((s,\langle v_1,\ldots,v_n\rangle),ss,X)\cdot[\![tr]\!]^{\kappa} \\
\quad \text{where } X = scopeunit(\kappa),\ ss = saveset(\kappa) \\
[\![\mathsf{input}\,\mathsf{none};tr]\!]^{\kappa} := \underline{\underline{tt}}\cdot[\![tr]\!]^{\kappa}\ .
\end{array}
$$

The equation $X_{st} = [\![alt_1]\!]^{\kappa'} + \ldots + [\![alt_n]\!]^{\kappa'} + waiting(s_1,\ldots,s_m,X) :\!\rightarrow \sigma_{\mathsf{rel}}(X_{st})$ describes that the process X behaves from the state st as one of the given transition alternatives, and that this behaviour is possibly delayed till the first future time slice in which there is a signal to consume if there are no more signals to consume in the current time slice. In equations, we use names of processes with state name subscripts, such as X_{st} above, as variables; in process expressions elsewhere, we use them to refer to the processes defined thus. In the case of a (guarded) transition alternative, the process expression $\underline{\underline{input}}((s,\langle v_1,\ldots,v_n\rangle),ss,X)\cdot[\![tr]\!]^{\kappa}$ expresses that the transition tr is initiated on consumption of a signal of type s.

The meaning of a transition, occurring in the scope of a process definition, is a process expression describing the behaviour of the transition. It is expressed in terms of the meaning of its actions and its transition terminator.

$$
\begin{array}{ll}
[\![a_1\ldots a_n\,\mathsf{nextstate}\, st;]\!]^{\kappa} := [\![a_1]\!]^{\kappa}\cdot\ldots\cdot[\![a_n]\!]^{\kappa}\cdot X_{st} & \text{where } X = scopeunit(\kappa) \\
[\![a_1\ldots a_n\mathsf{stop};]\!]^{\kappa} := [\![a_1]\!]^{\kappa}\cdot\ldots\cdot[\![a_n]\!]^{\kappa}\cdot \underline{\underline{stop}}(X) & \text{where } X = scopeunit(\kappa) \\
[\![a_1\ldots a_n\ dec;]\!]^{\kappa} := [\![a_1]\!]^{\kappa}\cdot\ldots\cdot[\![a_n]\!]^{\kappa}\cdot[\![dec]\!]^{\kappa}\ . &
\end{array}
$$

In the case of a transition terminated by nextstate st, the process expression expresses that the transition performs the actions $a_1,\ldots,a_n$ in sequential order and ends with entering state st – i.e. goes on behaving as defined for state st of the process concerned. In case of termination by stop, it ends with stopping the process concerned; and in case of termination by a decision dec, it goes on behaving as described by dec.

C.3 RAISE: A Formal Software Engineering Method

RAISE[26] is an acronym for Rigorous Approach to Industrial Software Engineering. It was originally coined as the name for a European collaborative R&D project carried out as part of the ESPRIT program between 1985 and 1990. The project was conducted by two companies an Denmark and two in the UK, and developed the RAISE Method, supported by the RAISE Specification Language (RSL) and a collection of prototype tools. RSL was described in a book [76].

The RAISE project was followed by another ESPRIT project LaCoS (Large scale Correct Systems using formal methods) 1990–95, involving nine industrial partners in six European countries. The aim of LaCoS was to try out the RAISE technology in a number of industrial applications and, using this experience, to further develop the tools, method, and, if necessary, language. The language was hardly changed; the method was described in a second book [77] and the tools were developed into a robust and commercially available product. The generally favourable experiences of the partner companies using RAISE were described in "experience reports" [27, 36].

RAISE aimed at providing a unifying improvement over formal methods such as VDM [67], Z [81], CSP [59], CCS [71], Larch [56] and OBJ [25, 26, 50]. In particular it succeeded in unifying the "model-oriented" approach of VDM and Z, the "algebraic" approach of Larch and OBJ and the "process algebra" approach to concurrency of CSP and CCS. The result is a "wide spectrum", general purpose specification language that can be used through software development from initial very abstract descriptions to detailed specifications that are very close to code (and even amenable to automatic translation to programming languages[27]).

There are a number of features of RAISE that make it particularly effective for the kind of projects on which UNU/IIST uses it:

Abstraction : Types can be introduced abstractly, as "sorts", together with "observers" that describe the information that can be extracted from them. So, for example, one might initially describe an abstract railway network as containing "stations" and "lines" between them such that routes between stations can be computed. Details of how such information will be stored, how it might be distributed, what additional information about capacities, signalling etc. might be needed, how routes will be computed, and so on, are left unspecified, to be added later during development.

We can take this abstraction process further. We can describe any network as composed of "nodes" and "connections" on which paths between nodes can be computed. This common description (in formal terms a "theory") can then be instantiated for a railway network, or a road network, or even an electronic communication network.

Rigour : RAISE, like any other formal system, allows one to do proofs. Proof can be an important part of formal development. One can identify important properties of a description,

[26] This section is written by Chris George

[27] The RAISE tools include translators to C++ and Ada.

such as the property that a timetable does not violate safety conditions about how far apart trains must be kept. Such properties can be proved.

Formal development from abstract descriptions to more concrete, detailed ones can also be checked for correctness. This checking can also be done by proof.

However, with current technology, proof is difficult and expensive: it requires highly trained people and a lot of time. In practice proofs are often limited to particular mission or safety critical features. In other situations we can resort to informal arguments — with the important proviso that such arguments can be turned into proofs if they are questioned. Such an informal but formalisable argument is termed "rigorous". RAISE supports rigorous arguments in its justification editor, part of its tool set.

Modularity : Specifications are in general much smaller than the software that is developed from them. But it is still important that specifications, like programs, can be written in pieces that are small enough to be comprehended, and these pieces, "modules" put together conveniently. There are other desirable properties of modules:

- Properties of modules must hold in the modules that use them.
- It should be possible to develop modules separately and safely without the perhaps separate teams working on them needing to know the details of what other teams are doing.
- It should be possible to describe abstract modules, such as networks of nodes and connections, and reuse them for stations and railway lines in one application and telephone switches and trunk lines in another.
- It should be easy to propagate changes to modules to others that use them.

All of these are supported by RAISE.

Method : Ideally there would be a recipe for developing each kind of system: Take a network description; instantiate nodes as ..., connections as ...; etc. This is impractical: we need always to invent, and a method needs to encompass a number of things:

General principles : Ideas such as moving from abstract to concrete descriptions in a (small) number of steps, of "stepwise development", is one such principle. The ideas of "object orientation", of considering the system initially in terms of objects with attributes and relations between them, and reflecting this in the design of the system as a collection of objects with associated "methods" invoked by passing "messages" between them is another.[28] The idea of distinguishing between domain analysis and requirements capture is another. Particular to RAISE is the principle of starting with an applicative (declarative) specification and, after developing this to a concrete specification, transforming it into an imperative sequential or concurrent one more suitable for the final implementation in languages like C++ or Ada.

Standard components : These include both standard specification components, like abstract networks, and also outline architectures, like Michael Jackson's problem

[28] RAISE has some, but not all, of the features of "object oriented" languages. But it does fully support the principles of object oriented development.

frames. The former can be recorded as RSL specifications; the latter are perhaps not formalisable.

Techniques : There is a wide spectrum here, from detailed transformations (such as replacing recursion by iteration), to techniques for translating specifications to programs, and techniques for doing proofs. Techniques for doing quality control of development are also important. Tools for executing or supporting such techniques are an important area of research.

Formal basis : Techniques in particular need to be formally based. We need to know a priori that a program transformation, a translation tool, a proof rule, or the set of proof obligations generated from checking a development step, are correct (which means we need to formalise what "correctness" means in each case).

RAISE supports all of these, in its specification language with its formal semantics and proof rules, in the book on its method, in the examples published in the literature, and in its tools.

Tool support : The RAISE tools consist of

editors : for specifications, theorems, relations between specifications, and justifications. All of these are based on a common "structured editor" that enforces syntactic correctness and checks type correctness interactively. All except the **justification editor** are also usable in batch mode, which is how more experienced users employ them;

proof assistant/justification editor : allows development and check of proofs;

translators : from a subset of RSL to Ada and C++;

pretty printers : that generate LaTeX from the output of the editors for inclusion in documents;

utilities : for managing collections of specifications, translations and justifications. In particular for storing and retrieving them and for bringing them consistently up to date after changes.

C.4 LSD: Logical Systems Development

C.4.1 Overview

We[29] outline the basic ideas behind LSD:

1. VDM + RAISE: Section C.4.2 on the next page

   How it builds on such 'methods' as VDM and RAISE.

2. Development Stages & Steps: Section C.4.3 on page 68

   And how it evolves around the "tripartite" stages of software development:

   (a) Domain Theory & Analysis:

   (b) Requirements Capture & Definition:

   (c) Software Design:
   — with the major steps of:

      i. *Software Architecture:*

      ii. *Program Organization:*

   Stages and steps may involve:

   - Validation wrt. clients & users
   - Verification of
     - *Properties* — eg. as part of validation
     - *Correctness* — between stages & steps
   - Model Experiments — investigations and calculations within other engineering regimes, typically operations research, econometrics, control theory, etc. Cf. section 4.1.2 on page 23.

   while promoting stepwise development of any of the above stages (and major steps).

3. Problem Frames: Section C.4.4 on page 70

   How it incorporates Michael Jackson's concept of Problem Frames [62].

4. Informal & Formal: Section C.4.5 on page 87

   An how it combines colloquial language:

   - *Synopsis*
   - *Narrative* and

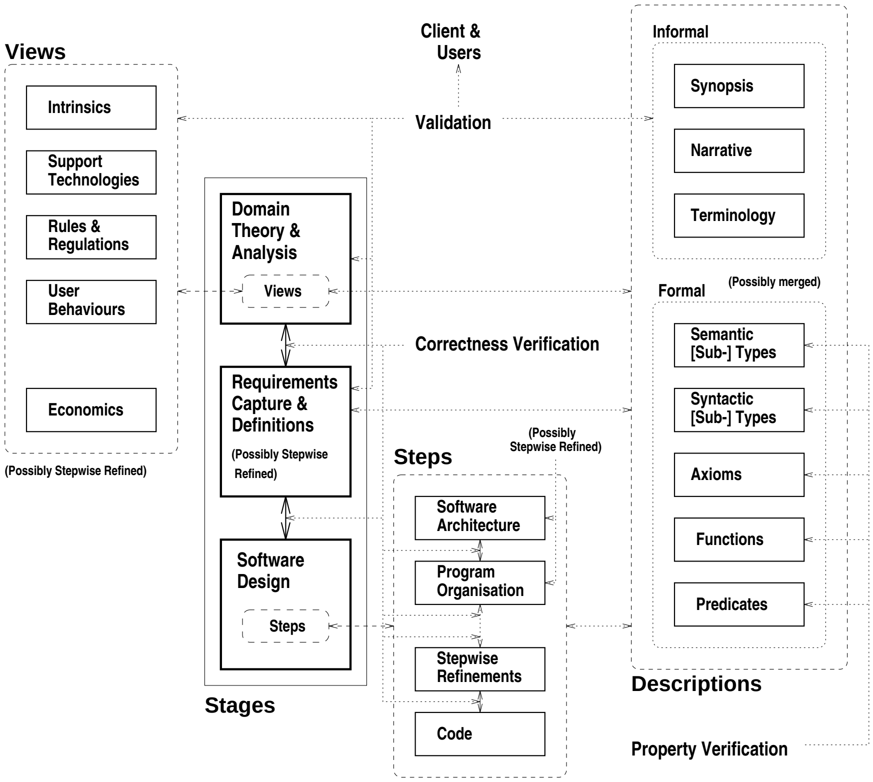
[29] This section is written by Dines Bjørner

- *Terminology*

(i.e. informal) descriptions with formal descriptions.

We refer to figure 3 for a summary of the above.

Figure 3: Development Artifacts & Their Relations



The "LSD" Stage & Step Descriptions, Validations & Verifications

C.4.2 VDM & RAISE

Currently UNU/IIST is applying (i.e. using) RAISE [76, 77] throughout its many advanced development projects — as is witnessed by the examples of special appendix sections E.1 (starting page 103) — E.13 (starting page 190). RSL, the RAISE Specification Language, is based first and foremost on VDM [14, 64, 15, 65, 66, 1, 68], but incorporates essential concepts of ccs [71], CSP [59], OBJ, [25, 26, 50] and Standard ML [58, 72, 80]. The RAISE Method is supported, at UNU/IIST by the RAISE Tool Set, a rather comprehensive, industrial strength system. In our judgment RAISE is the best matured system for professional software development. UNU/IIST secures academic RAISE Tool Set licenses to most of its fellows upon their return to their seconding institutions.

C.4.3 Stage & Stepwise Development

In special appendix section F (starting page 194) we illustrate the below outlined "trinity" of stage-wise development of software through an classical example. It is primarily drawn from control theory & engineering. We believe that the example "speaks" rather directly to a broader audience of readers.

- Domain Theory & Analysis:

  Before we can develop software we must first develop the basic (or full) requirements for that software. And before we can do that we must first understand the application domain. That makes good sense — and is the way things are done in for example control engineering and in operations research engineering.

  So, before we go into deeper reflections on software requirements, we spend some considerable effort on understanding the application domain.

  Understanding the application domain must be done, we argue, **without any reference to computing!** The application domain was there before computing and ought be describable as such: with no computer, let alone software technology jargon!

  The examples, shown — rather cursorily — in special appendix sections E.1 (starting page 103) — E.13 (starting page 190), all focus on the Domain Theory aspects.

  Domain Theories may not necessarily be developed by software engineers, but rather by, or with the help of domain "experts". However, as we shall see in special appendix sections E.1 (starting page 103) — E.13 (starting page 190), these people must first have learned appropriate Domain Theory modelling (specification) methods — for example such as hinted at in the referenced special appendix sections E.1 (starting page 103) — E.13 (starting page 190). These domain modelling methods may have originated in computing science, but should probably be propagated to the practitioners of a long list of other disciplines.

- Requirements Capture & Definition:

  Having understood the application domain the developer can now devote attention to understanding the client's requirements.

  The requirements definition ideally should express externally "visible" (i.e. usable) properties (i.e. "features") of the required software. As such Requirements emphasise **what**. Requirements should not state properties of the domain (such should be part of the domain theory). And Requirements should preferably, but see next, not express properties of the software implementation. That is: Requirements need in many cases not constrain the software designer as **how** the software should offer its functionalities.

  As is proposed, implicitly, in this report, the client's expressions of Requirements are "translated" into informal and formal descriptions, consistent with the Domain Theory descriptions. We often find that client expressions of Requirements mix up to three facets: Domain knowledge, Requirements and Software Design. The first, if not already embodied in the (somehow) existing Domain Theory, must lead to its revision or extension. The latter must either (i) be discarded, or (ii) be pushed to the stage of Software Design, or (iii) be accepted as Software Design constraining requirements.

- Software Design:

  Once the Requirements Definition has been basically completed, Software Design can commence in earnest. What was not mentioned in the two previous points (Domain Theory & Analysis and Requirements Capture & Definition), but which is also true for those stages, is that Software Design proceeds in steps.

  1. Software Architecture:

     A first step is that of determining the Software Architecture: the features of the software implementation which are externally "visible", that is: which can be conceived ("felt") by the end user.

  2. Program Organisation:[30]

     A second step is that of determining the Program Organisation: the decomposition into modules (including processes) and their internal interfaces (including [message] communication).

  3. Further Refinement Steps:

     Usually there then follows several further stages of refinement: "turning" abstract data types and pre-condition/post-condition specified functions into efficient data structures and algorithms.

  4. Code:

     And finally executable code can be derived — perhaps automatically.

Some comments on the "trinity" of stages and their multiplicity of steps seem in order:

- *Validation:*

  It goes without saying that validation (and verification, see next) form essential parts of the development principles and techniques propagated by UNU/IIST.

  Validation is about: *Are we building the right product?* — and is supported, as a development activity, primarily in the stages of Domain and Requirements modelling. In these stages interaction with the client is of utmost importance. In some cases validation requires formal verification of description (document) properties — which are then "claimed" to reflect "real world" properties.

[30] Unfortunately we find that our colleagues at CMU: Profs. Mary Shaw, David Garland et al., are using the term Software Architecture where we would use the term Program Organisation. We insist on our usage — since it is commensurate with that for hardware: A computer systems' architecture is that of its data structures and instruction format (and repertoire), whereas several machine organisations may implement the same computer architecture but be based on anything from 8 bit to 132 bit data flow, strict sequential instruction processing vs. pipelining, etc. These hardware terms were well established already in the early to mid 1960's. It would not be prudent to change "meaning" just for the sake of "hitting a hot item", jargon-wise!

- *Verification:*

  Verification is about: *Are we building the product right?* — and is supported, as a development activity, by verifying properties of Domain, Requirements and Software descriptions as well as verifying relations (including correctness) between stage or step descriptions.

- *Software vs. Computing System:*

  At UNU/IIST we focus primarily on the software side, but we show our partners, that determination and hence procurement of hardware (equipment) can only commence once proper requirements are available.

  We see — all too often — cases, where the international aid system are lending or granting monies to information technology, where acquisition of hardware precedes determination and completion of either software acquisition or [sometimes own] development of software — and by years! Meanwhile the hardware equipments becomes outdated; yes, in cases, "rots away"!

  Methodologically, therefore, the requirements usually embody more or less implicit references to both hardware and software platforms to be used, or these platforms are determined in the first steps of software design.

- *Possibly Cyclic Iterations:*

  Although "billed" as a linear progression from Domain via Requirements to Software, development usually "cycles" (i.e. iterates) between these stages and their steps. Management therefore ensures that proper documentation records this "history" of development changes.

- *High–level Platforms:*

  UNU/IIST propagates implementation of software both in terms of traditional programming languages and in terms of high level software platforms. Among the latter we mention concrete as well as abstract examples: X-Windows & Athena Widgets, CORBA, ODP, etc. — i.e. we implicitly "sponsor" the OMG effort (see also special appendix section D.3 (starting page 99)).

C.4.4 Michael Jackson's Problem Frames

Michael Jackson, in his delightful book [62] (given to all fellows at UNU/IIST), identifies a number of rather distinct development frames — each requiring rather distinct development principles, techniques and tools.

This section represents work in progress joint with Drs. Koussoube, Noussi and Satchok [23, 22].

Translation Frame

Examples of problems that belong to the translation frame are: compilers, JSP, and rewrite rule systems.

- **Translation Domain**

`A Basis for a Translation Domain:` *We have the basis for a translation problem frame if the domain can be expressed in terms of two formal languages. That is: in terms of their formal syntax and semantics.* Let the formal languages be designated by L_1 and L_2. Let the corresponding semantic domains[31] be designated by D_1 and D_2 — where D stands for Denotations. Then we can express the basis for a translation frame by:

type p1:L_1, p2:L_2; d1:D_1, d2:D_2
value M_1: L_1 $\stackrel{\sim}{\rightarrow}$ D_1; M_2: L_2 $\stackrel{\sim}{\rightarrow}$ D_2

Usually, for formal languages, the Meaning functions M_1, M_2 can be expressed as the composition, each of two functions:

value
  Mss_i: L_i → **Bool**; Mds_i: L_i → D_i
  M_i(pi) ≡ **if** Mss_i(pi)
    then Mds_i(pi) **else** ⊥ **end**

`Transation Domain:` *We have a translation frame if furthermore we have a set of abstraction or retrieve functions between semantics domains of the two languages.* To express the retrieve relation we postulate a notion of 'state', subsets S_1 of D_1, respectively S_2 of D_2.[32] Then: Retrieve: R: S_2 → S_1; Abstraction: A: D_2 × D_1 → **Bool**.

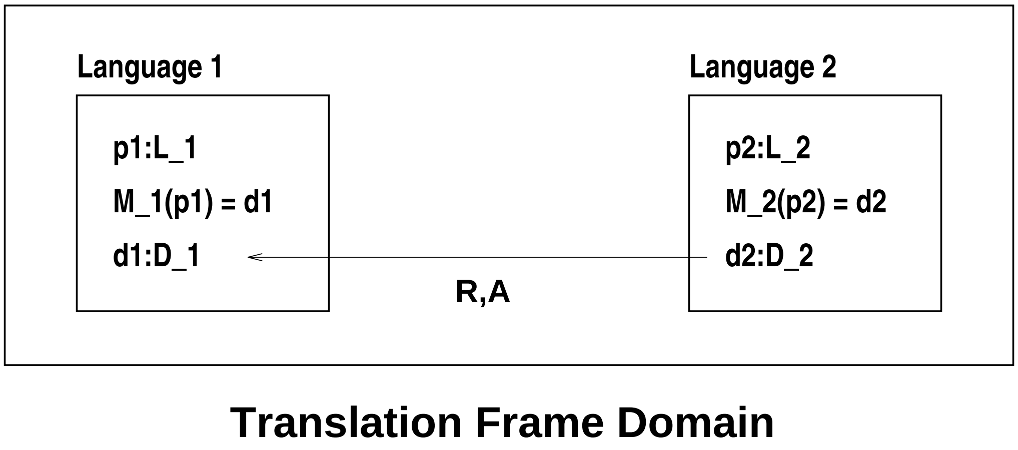
- **Translation Requirements**

The translation requirements define what is, at a minimum, required from a translator T: `Translation Requirements:` *Translator* T *requirements are expressed in terms of the commutation of a diagram.* Commutation: $R(s_1, s_2) \supset A(M\_1(p1)s_1, (M\_2(T(p1)s_2)))$ Interactive diagnostics and editing requirements belong to a workpiece-like frame.

[31] Pls. be aware that the term domain is used in two senses: (i) the application domain, and (ii) the syntactic and semantics domains (as designated here by L_1, L_2, D_1, and D_2 respectively).

[32] At present it is not important to know other than that about states!

Figure 4: Translation Domain



- **Translation Design**

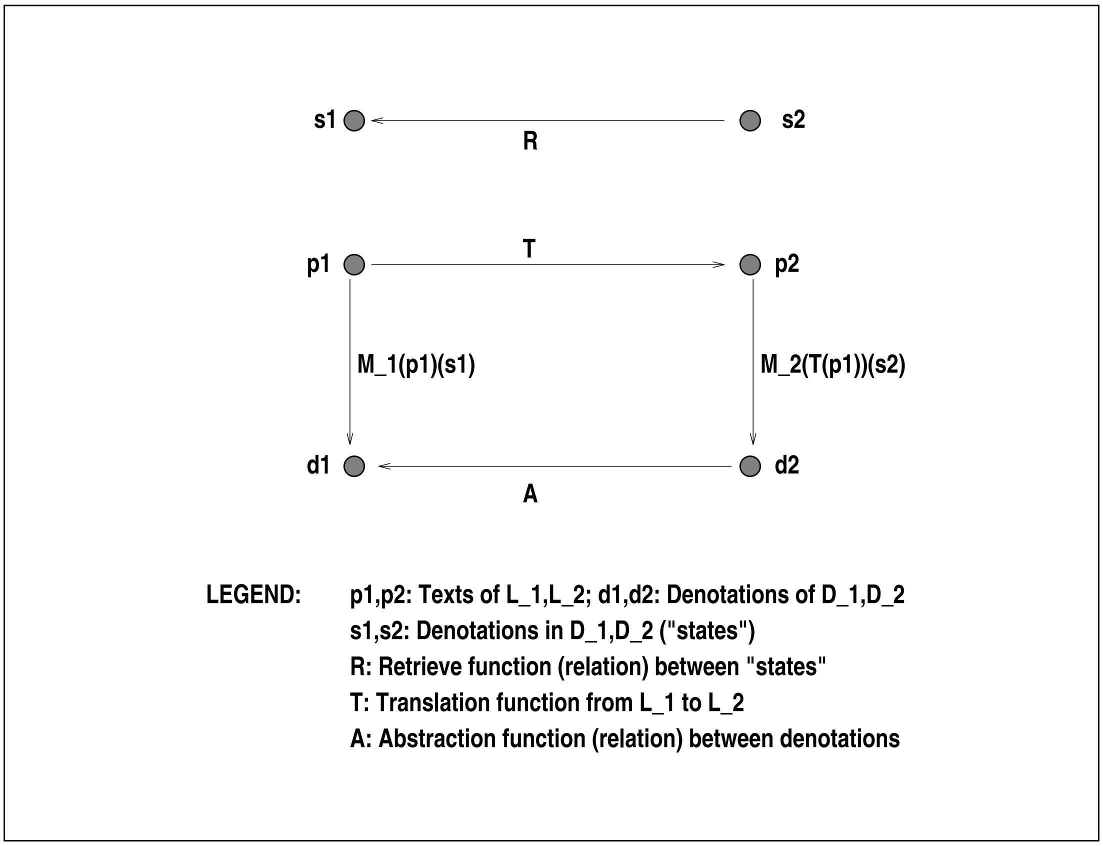
We "narrow" our Translation Frame example to that of Compilation between inert program texts, typically from a Programming Language L_1 to a Machine Language L_2. In such a case issues like optimisation may enter our concern — and would then either be mandated in the Requirements, or would be a Design option. The translator can now be implemented, observing the requirements. A number of theories and techniques & tools can be applied in the design process. One design paradigm is expressed and pictured below.

`Translator (Compiler) Design:` *Following one compiler design paradigm we may structure the program organisation as follows:*

`Lexical Scanner:` *In a Lexical Scanner input language (L_1) texts have their individual characters assembled, i.e. "translated" into tokens.* `Syntax Parser:` *In a Syntax Parser linear sequences of tokens are parsed, that is: "translated" into a parse, i.e. a tree-structured text in a language PS_1.* Static Semantics Analyser: *The Static Semantics Analyser implements Mss_1.* ``` ``Optimiser'': ``` *An "Optimiser" may reorganise the abstract PS_1 text in a way that may make the subsequent Code Generator able to generate so-called "efficient" texts.* `Code (Text) Generator:` *The Code Generator "implements" parts of Mds_1 such that the diagram commutes.*

type *TK_1, PS_1*
value
  LS: $L\_1 \stackrel{\sim}{\to} TK\_1$
  PS: $TK\_1 \stackrel{\sim}{\to} PS\_1$
  SA: $PS\_1 \to$ **Bool**
  OPT: $PS\_1 \to PS\_1$

Figure 5: Translation Design



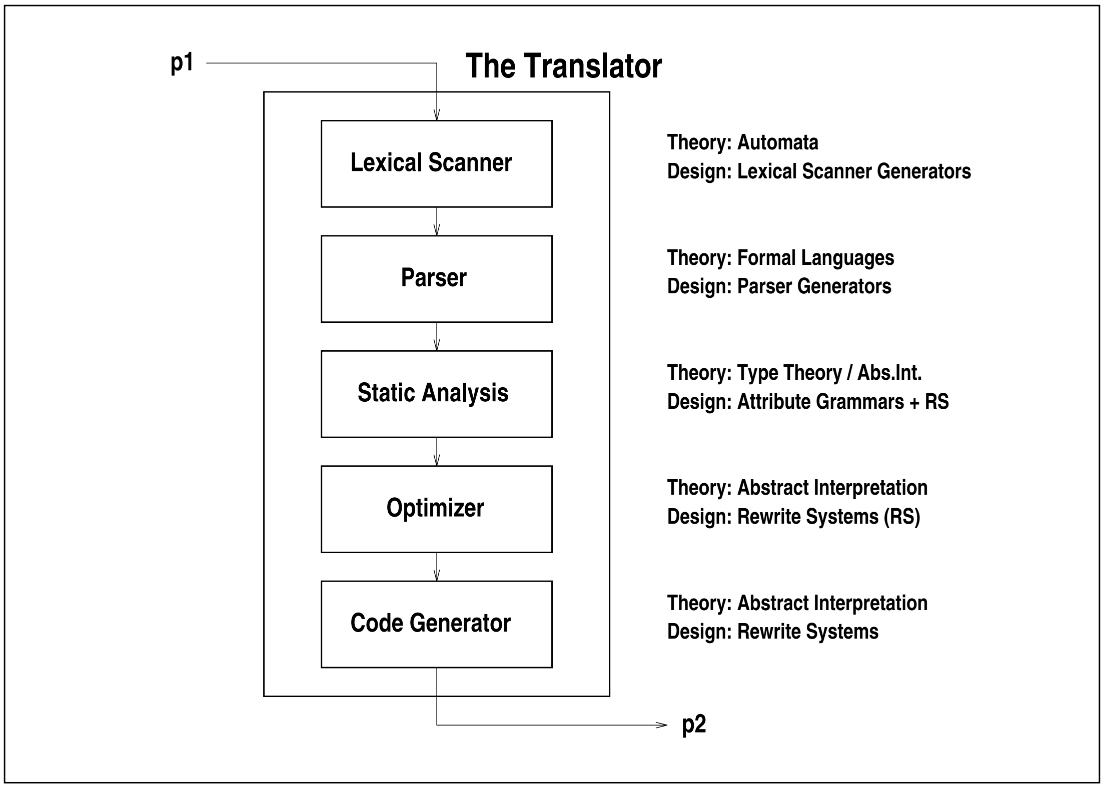
Translation Frame Requirements

$\quad CG\!:\ PS\_1 \rightarrow L\_2$
axiom
$\quad \forall\ p1{:}L\_1,\ \exists\ s1{:}S1,\ s2{:}s2 \bullet$
$\qquad R(s1,s2) \Rightarrow$
$\qquad$ **let** $ps = PS(LS(p1))$ **in**
$\qquad SA(ps) \Rightarrow A(M\_2(CG(ps)(s2)),M\_1(p1)(s1))$
$\qquad$ **end**

The sum total is a multi-pass compiler (translator):

What has not been shown in the above is the possibility that the compiler (i.e. the translator) is further program structured around the concept of multi-pass compilation.

Figure 6: Translation Design



Translation Frame Design

Control Frame

Examples of problems which belong to the control frame are: automated process control, humanly "controlled" air traffic, and train dispatch.

• Control Domain

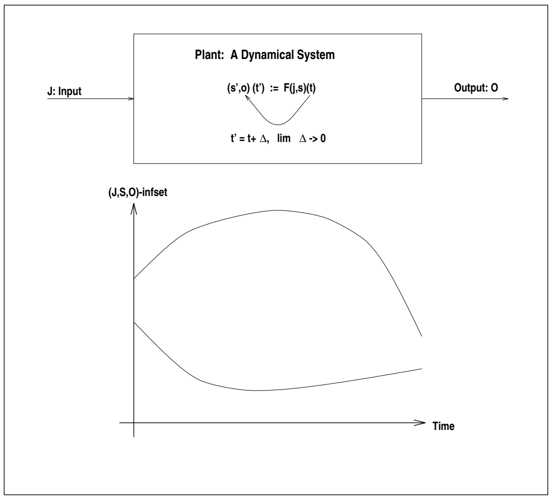
`A Basis for a Control Domain:` *We have the basis for a Control Frame if the Domain can be expressed in terms of time, T, a set of states, S, a set of external inputs, J, a set O of external outputs, the dynamics of the input and state as relations Σ and Γ over time, and a time dependent next-state and output function, F — which usually is (implicitly described, i.e.) approximated by a set of differential equations.*

type $\mathrm{T}, \mathrm{S}, \mathrm{J}, \mathrm{O}, \Sigma = \mathrm{T} \overset{\sim}{\to} \mathrm{S}, \Gamma = \mathrm{T} \overset{\sim}{\to} \mathrm{J}$

value F: (T $\tilde{\to}$ (J×S)) $\tilde{\to}$ (T $\tilde{\to}$ (S×O)-**infset**)

The Basis function F determines a space, an operating regime, within which — over time and otherwise determined by the input and the (previous) state — the (next) state and (next) values of the observable variables will lie. Σ and F express that the Domain is dynamic and reactive.

Figure 7: Control Domain



Reactive Systems Frame Domain

In figure 7 the uncontrolled operating regime — i.e. the set of all possible, including erroneous, (i.e. undesirable) behaviours[33], is "symbolised" by the space between the two curves. The "X" axis depicts time, the "Y" axis a mapping onto one dimension of the three-dimensional point set space $(J \times S \times O)$-**set**.

`Further towards a Control Domain:` *Usually one can describe a number of observer predicates. These "speak" of coherency, stability, robustness, and controllability properties of the Domain:*

type
  Sys_base = T → J×S×O
  Sys = {| sb:Sys_base • coherent(sb) |}
value
  coherent: Sys_base → **Bool**
  stable: Sys → **Bool**

[33] We call $T \to J \times\ S \times\ O$ a space of behaviours. The sequence:

$$< (t,j,s,o),(t',j',s',o'),...,(t'',j'',s'',o''),... >$$

such that the sequence of time stamps $t, t', ..., t''$ is dense and ascending resembles a behaviour.

robust: Sys → **Bool**
observable: Sys → **Bool**
controllable: Sys → **Bool**

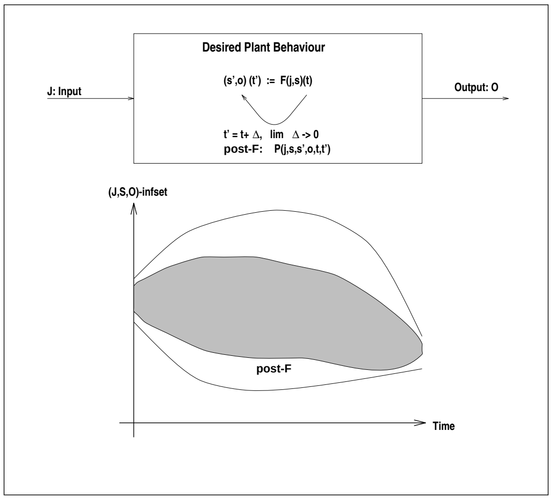
• Control Requirements

`Control Requirements:` *We finally have a Control Frame if in addition to its Basis, see section C.4.4 on page 74 we have value constraints for the observable variables — i.e. if only some behaviours among all possible are permitted.*

value P: (T → (J×S×O)-**infset**) → **Bool**

The idea is that P is expressed in terms of, amongst others, the coherency, stability, robustness, and controllability properties of the Domain. Additionally optimality criteria form part of P. In figure 8 the constraints are symbolically shown as the "narrower" space of grey shaded behaviours.

Figure 8: Control Requirements



Reactive Systems Frame Requirements

Major requirements definition concerns are: determination of the image of the state to be projected (into the computer), regularity of update, and initial input.

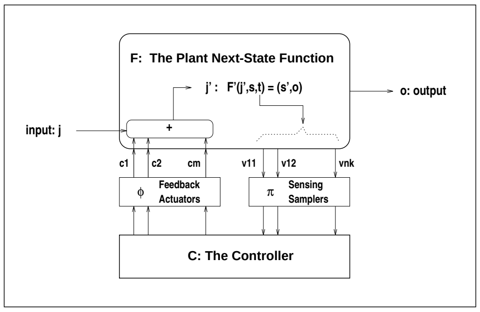
• Control Design

Control engineering is about controlling some of the observable variables based on past values of all (required) observable values. The iteration over time, referred to in section C.4.4 on page 74, together with the non-deterministic nature of the input j:J, implies a feedback loop. Some observable variables are sampled (sensed at regular (incl. computer) time periods).

`Control Design:` *We have solved the Control Requirements if we have: a sampling (sensor) function π which observes the state, a Control Function, C, and a feedback activator function ϕ, such that s' in: $C(\pi(s)_{(t)} = (c1, \ldots, cm)_{(t+\delta)}$, and $F(j \oplus \phi(c1, \ldots, cm)_{(t+\delta)}, s")(t + \delta') = s'_{(t+\Delta)}$, lies within the control regime (s" is the state at time $t + \delta'$).*

type c1,...,cm:V
value
  π: S $\rightarrow$ T $\rightarrow$ (V $\times$ T)
  C: (V $\times$ T) $\rightarrow$ (V $\times$ T)
  ϕ: (V $\times$ T) $\rightarrow$ J $\times$ T
  $\oplus$: J $\times$ J $\rightarrow$ J
  F: J $\rightarrow$ S $\tilde{\rightarrow}$ T $\tilde{\rightarrow}$ (S $\times$ O)

Figure 9: Control Design



Reactive Systems Frame Design

Information System Frame

• Information System Domain

`A Basis for an Information Domain:` *We have the Basis for an Information System Frame if there is given a model of the "real world", W, in terms of the types, $W\_1, W\_2, \ldots, W\_n$, of some of its atomic or composite individuals, the signature of its observer functions (relations)* *and predicates constraining some of its (atomic or composite) individuals.* The below illustrates such a model.

type for i=1,2,...,n:
  W, W_i
  $\Sigma = \mathrm{T} \tilde{\rightarrow} \mathrm{W}$
  $\mathrm{Q\_i} = \mathrm{T} \tilde{\rightarrow} \Sigma \tilde{\rightarrow} \mathrm{W\_i}$
value for i,j,k=1,2,...,n,m,n:
  P_j: W_i1×W_i2...W_ik → **Bool**
  obs_W_i: $\mathrm{Q\_i} \times \mathrm{T} \tilde{\rightarrow} \Sigma \tilde{\rightarrow} \mathrm{W\_i}$
  obs_W_i(p,t)(σ) ≡ {| w_i:W_i • w_i = p(t)(σ) |}

The *W_i* (observer) functions so-to-speak act as "projections" from the "real world" into the model referred to above. Typical observer functions, in a non-computerised "world", amount to ordinary human data collection — say the gathering of population census in paper files, or the gathering of experimental data from a series of scientific experiments likewise in a paper file. The predicates correspond to the data vetting normally done in the above informal example situations: i.e. the human checking and cross-checking that collected data satisfy various constraints.

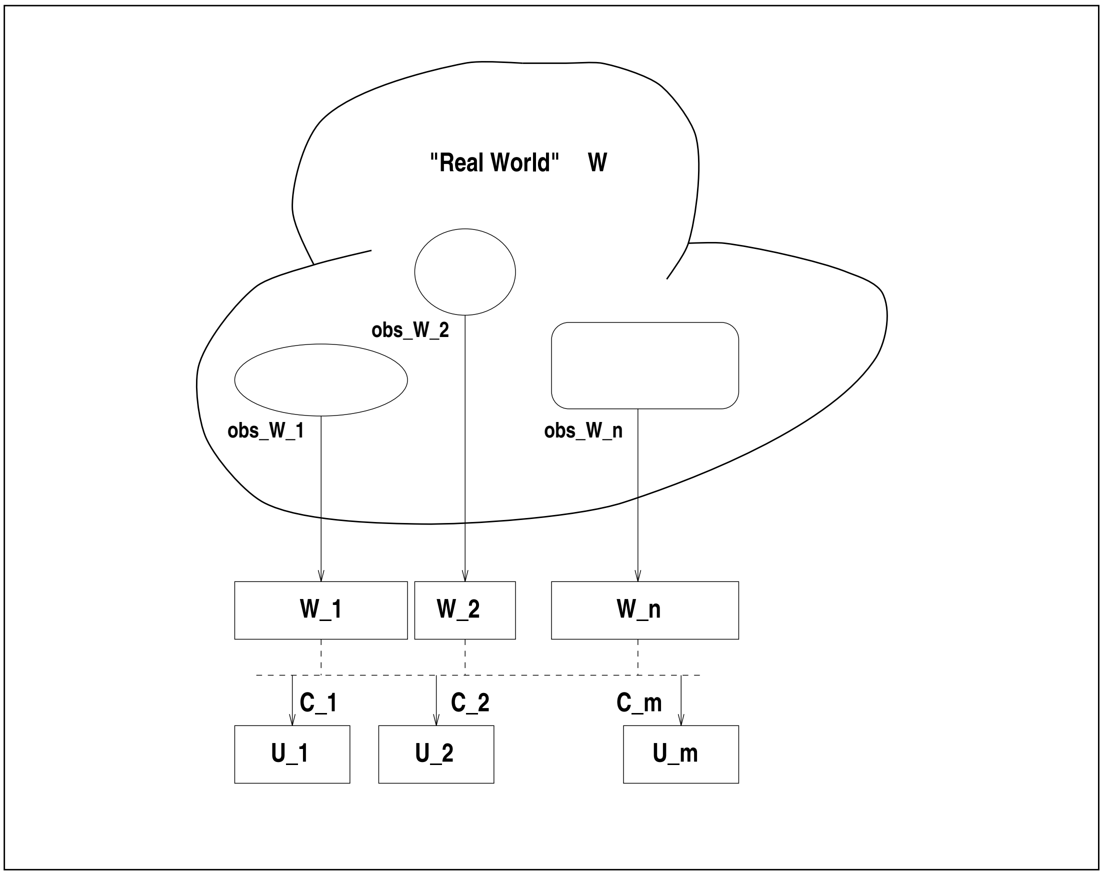
`Information Domain:` *We have an Information System Frame if — in addition to the above Basis we further have a number of functions which perform a number of calculations over the projected data.*

type
  U_1, U2, ..., U_m
value for i=1,2,...,m:
  C_i: W_i1 × W_i2 × ... × W_ij $\tilde{\rightarrow}$ U_i

• **Information System Requirements**

`Information System Requirements:` *Information System Requirements is expressed basically as follows.* Data Collection: *Implement observer functions obs_W_i together with data vetting as expressed by the predicates Q_i and the axiom(s).* Data Storage: *"Implement" (including migration [i.e. conversion] of) paper data files.* Data Queries: *Implement subset of calculations C_j with provisions for temporary storage and display.* Data "Creation": *Implement remaining calculations C_j with provisions for longer-range storage and display.* "Completions": *Whereas the Domain functions obs_W_i, C_j, predicates P_j and the constraint axiom(s) may have been incomplete, their Requirements definition must now be "complete(d)".*

Figure 10: Information System Domain



Information Frame Domain

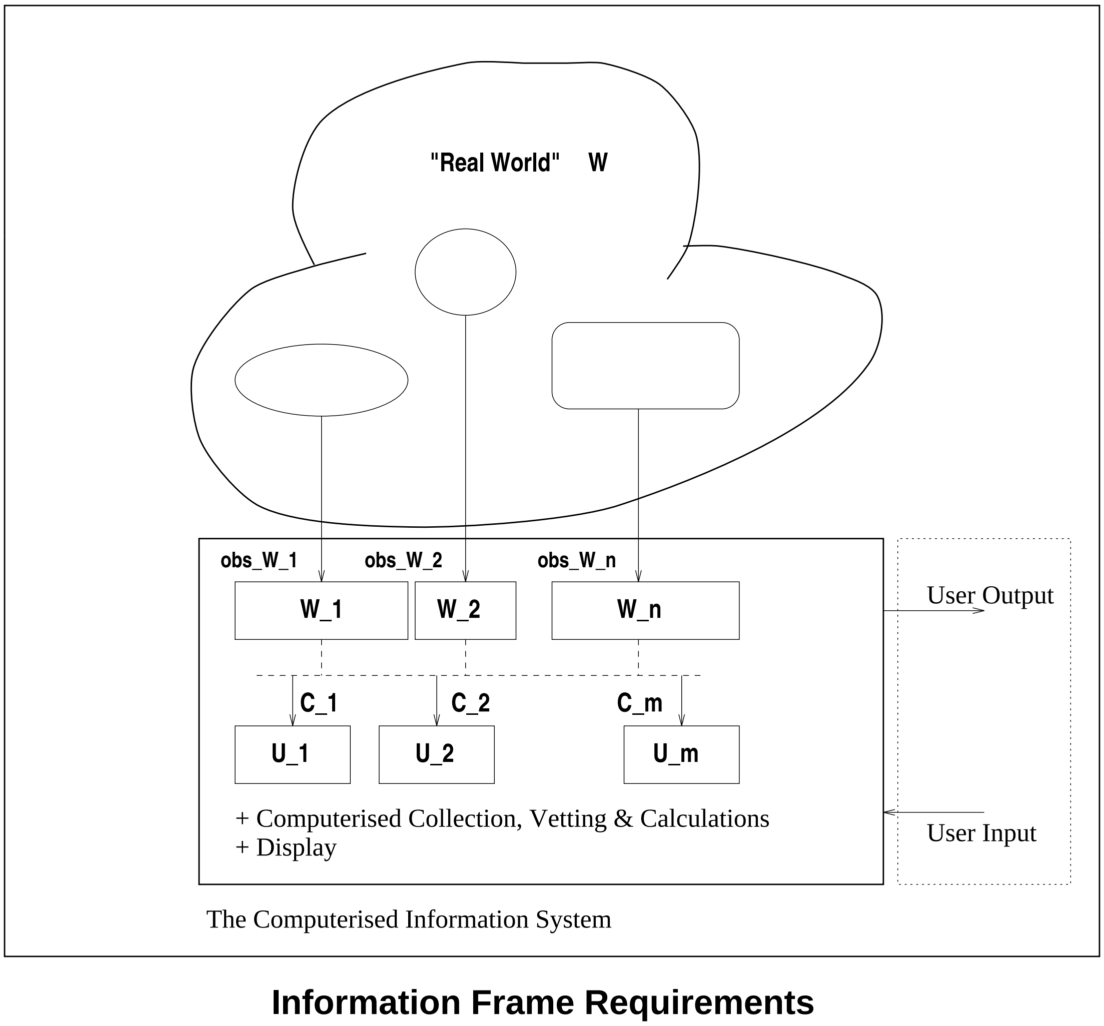
Workpiece Frame

Diverse examples of problems which belong to the workpiece frame are: office procedure forms automation, CAE/CAD, and word processors.

• Workpiece Domain

`Workpiece Domain:` *We have a Workpiece Frame when the universe of discourse can be characterised in terms of:* Users: *Users manipulate tangible, manifest documents such as templates, designs and aggregates.* Templates: *Templates are one kind of documents — think of them as forms with preformatted rules & regulations and ready to be "filled-in".* Designs: *Designs are partially or fully "completed" templates. Examples of designs will be given later.* Aggregates: *Aggregates are based on templates and are composed from zero, one or more designs.* We refer

Figure 11: Information System Requirements



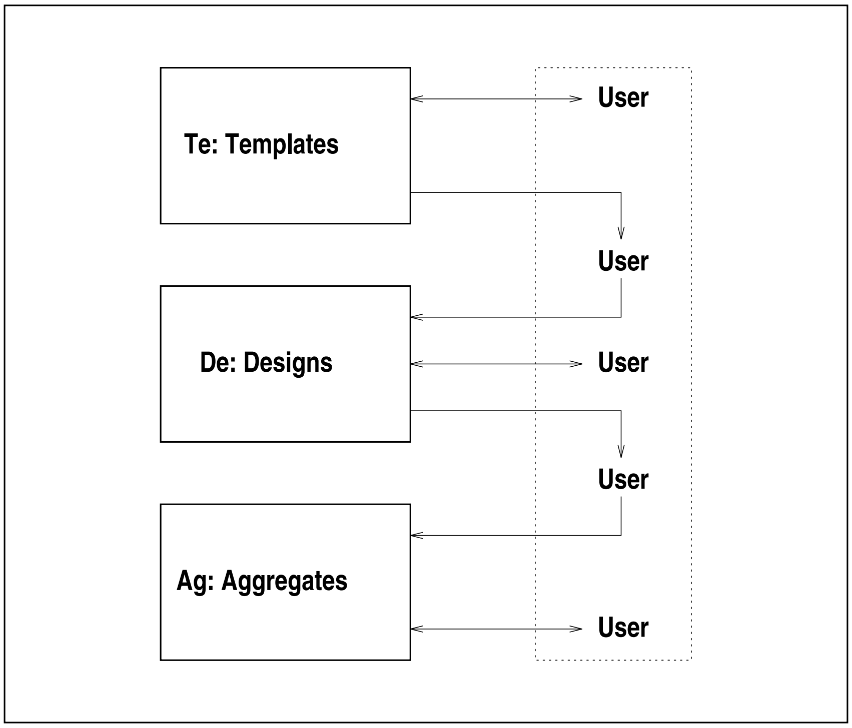
to templates, designs and aggregates as units. Our descriptions above are deliberately sketchy. This allows us to consider a large class of applications as belonging to the Workpiece Frame. Formally we can summarise:

type
  Te, De, Ag, Inf
value
  gTe: Inf $\stackrel{\sim}{\to}$ Te
  bTe: Inf $\stackrel{\sim}{\to}$ Te-**set**
  uTe: Te $\times$ Inf $\stackrel{\sim}{\to}$ Te
  gDe: Te $\times$ Inf $\stackrel{\sim}{\to}$ De
  bDe: Te $\times$ Inf $\stackrel{\sim}{\to}$ De-**set**

uDe: De $\times$ Inf $\xrightarrow{\sim}$ De
gAg: Te $\times$ Inf $\times$ De* $\xrightarrow{\sim}$ Ag
bAg: Te $\times$ Inf $\xrightarrow{\sim}$ Ag-**set**
uAg: Ag $\times$ Inf $\times$ De* $\xrightarrow{\sim}$ Ag

Legend: Te: Template, De: Design, Ag: Aggregate. Inf: Information. g: generate, b: browse, u: update. **Explanation:** gTe: Generate Template based on (some) Information. bTe: Browse for (suitable) Template with (some) Information. uTe: Update (given) Template with (some) Information. gDe: Generate Design given Template and (some) Information. uDe: Update given Design with (some) Information. gAg: Generate Aggregate given (its) Template and list of (input) Designs. uAg: Update (given) Aggregate with Inf and list of (input) Designs. *&c.*

Figure 12: Workpiece Domain



Workpiece Frame Domain

- **Workpiece Requirements**

`Workpiece Requirements:` *Workpiece Requirements typically states: Implement the Workpiece "System" on the computer. That is: Instead of manifest (tangible) units and the physical* *communication of these among users, support templates, designs and aggregations: their "storage" and manipulation (creation, update, etc.) by computing.*

Workpiece Requirements usually are more detailed in their description of templates, designs and aggregations as well as the operations on these. Typically the units all have own, unique identifications and separate or shared storage. Sometimes time-stamped versions of units are kept for reference, etc.

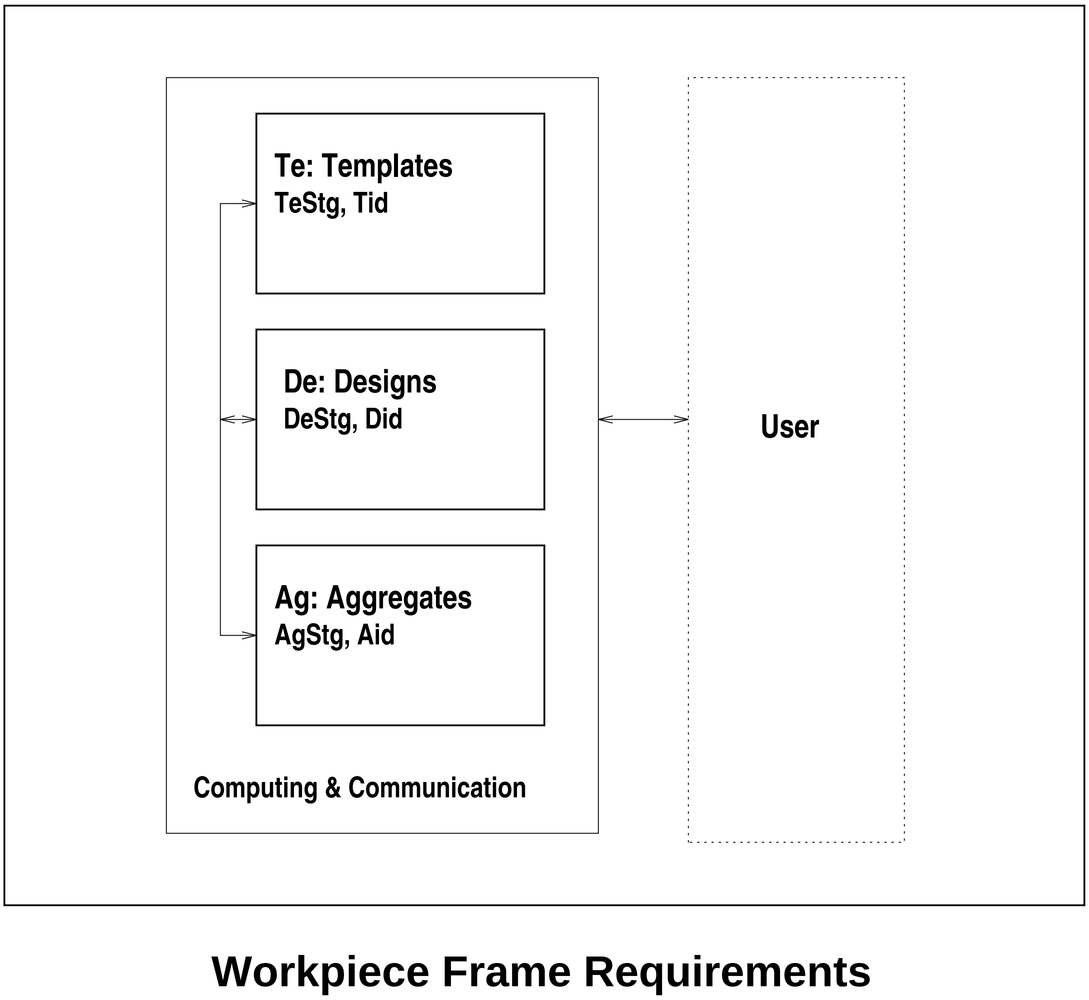
type
$\mathrm{T, Ti, Di, Ai, Inf}$
$\mathrm{Temp} = \mathrm{Ti} \times \mathrm{Te}$
$\mathrm{Dsgn} = \mathrm{Di} \times \mathrm{Ti} \times \mathrm{De}$
$\mathrm{Aggr} = \mathrm{Ai} \times \mathrm{Ti} \times \mathrm{Ag}$
$\Sigma = \mathrm{TeStg} \times \mathrm{DeStg} \times \mathrm{AgStg}$
$\mathrm{TeStg} = \mathrm{Ti} \underset{m}{\to} (\mathrm{T} \underset{m}{\to} \mathrm{Te})$
$\mathrm{DeStg} = \mathrm{Ti} \underset{m}{\to} \mathrm{Di} \underset{m}{\to} (\mathrm{T} \underset{m}{\to} \mathrm{De})$
$\mathrm{AgStg} = \mathrm{Ti} \underset{m}{\to} \mathrm{Ai} \underset{m}{\to} (\mathrm{T} \underset{m}{\to} \mathrm{Ag})$
value
$\mathrm{GTe}: \mathrm{Inf} \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma \times \mathrm{Ti}$
$\mathrm{BTe}: \mathrm{Inf} \to \Sigma \to \mathrm{Ti}\textbf{-set}$
$\mathrm{UTe}: \mathrm{Ti} \times \mathrm{Inf} \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma$
$\mathrm{GDe}: \mathrm{Ti} \times \mathrm{Inf} \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma \times \mathrm{Di}$
$\mathrm{BDe}: \mathrm{Ti} \times \mathrm{Inf} \to \Sigma \stackrel{\sim}{\to} \mathrm{Di}\textbf{-set}$
$\mathrm{UDe}: \mathrm{Di} \times \mathrm{Inf} \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma$
$\mathrm{GAg}: \mathrm{Ti} \times \mathrm{Inf} \times \mathrm{Di}^* \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma \times \mathrm{Ai}$
$\mathrm{BAg}: \mathrm{Ti} \times \mathrm{Inf} \to \Sigma \stackrel{\sim}{\to} \mathrm{Ai}\textbf{-set}$
$\mathrm{UAg}: \mathrm{Ai} \times \mathrm{Di}^* \times \mathrm{Inf} \to \mathrm{T} \stackrel{\sim}{\to} \Sigma \stackrel{\sim}{\to} \Sigma$

Legend: T stands for (the domain of) Time. Ti, Di, and Ai are Template, Design and Aggregate Identifiers. Temp, Dsgn and Aggr are "more" concrete forms of Te, De and Ag. Σ is the "grand" state. It consists of the Template, the Design and the Aggregate Storages. G, B, U stands for Generate, Browse and Update. **Explanation:** Designs and Aggregates "carry" their Template identifications. Storage of Templates, Designs and Aggregates are time-stamped — thus "old" versions are (never) discarded. Generation updates Storage and yields an appropriate (Template, Design or Aggregate) identifier. Browsing may yield zero, one or more Template, Design or Aggregate identifiers but does not change Storage. Updates apply to latest time stamped and identified Template, Design or Aggregate and change Storage.

Connection Frame

Examples of problems which belong to the connection frame are: check & credit card slip transaction clearing, and programmable (cabled) sensor and actuator connections between humans or

Figure 13: Workpiece Requirements



processes and computers. Data communication with its protocols belong to the communications frame.

- **Connection Domain**

`Connection Domain:` *When one amongst one or more (so-called) senders wishes to send a message (a signal, a package, whatever) to one amongst one or more (so-called) receivers then we have the Basis for a Connection Frame.* We assume all senders and all receivers to be uniquely identified. Senders may "store" messages sent and mark these acknowledged when such has been obtained. Receivers may "store" messages received and mark these acknowledged when such has been done.[34]

```
type
   T, Si, Ri, M
   Sn = Ri →m (T×M)-set ...
   Rc = Si →m (T×M)-set ...
   Σ = (Si →m Sn) × (Ri →m Rc)
value
   SndAck: Si × M × Ri ~→ T ~→ Σ ~→ Σ × T
   snd: M ~→ Sn ~→ T ~→ (Sn × T)
   rcv: M ~→ Rc ~→ T ~→ (Rc × T)
   ack: M ~→ Sn ~→ T ~→ (Sn × T)
   max: T* → T
   SndAck(s,m,r)(t)(ss,rs) ≡
         let (sn,rc) = (ss(s),rs(r)) in
      1. let (sn′,t′) = snd(m)(sn)(t) in
         let t″:T • t″ ≥ max(<t,t′>) in
      2. let (rc′,t‴) = rcv(m)(rc)(t″) in
      3. let (sn″,t⁗) = ack(m)(sn′)(t‴) in
         let t′′′′′:T • t′′′′′ ≥ max(<t,t′,t″,t‴,t⁗>) in
      4. (ss † [s↦sn″],rs † [r↦rc′],t′′′′′)
      end end end end end end
```

Legend: T stands for time. Si, Ri: Sender and Receiver Identifiers. M: Messages and Acknowledgements. The state Σ "summarises" all senders and receivers. **Explanation:** At any time any sender has sent messages to and possibly received acknowledgements from identified receivers. Similarly for receivers. (1.) Sender s processes the message to be sent at time t thereby changing its (own) state at that time. (2,) Receiver r receives the message at time t''' thereby changing its state at that time. (3.) Sender s receives an acknowledgement from receiver r at time t'''' thereby changing the state at that time. (4.) The sum–total result is a new system state at time t'''''. Connection via the Medium takes time as illustrated above.

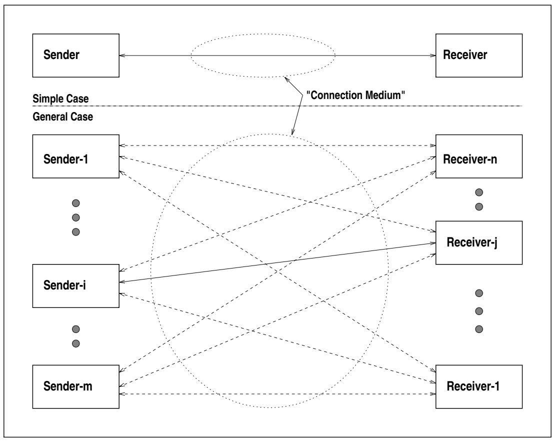
• Connection Requirements

`Connection Requirements:` *When one amongst one or more (so-called) senders wishes to send a message (a signal, a package, whatever) to one amongst one or more (so-called) receivers* via a shared connector *and within a certain time period then we have a Connection Frame.*

There may be one or more potentially sharable connectors. For each type of message and pair of senders and receivers there may be exactly one connector.

[34] In a "real world" one and the same physical entity may take on both sender and receiver rôles, and there is nothing in the above description that prevents this.

Figure 14: Connection Domain



Connection Frame Domain

type

$\quad \mathrm{MTyp}, \mathrm{Ci}, \Xi, \mathrm{R}$

$\quad \Psi = \mathrm{ClrIns} \times \mathrm{Conns} \times \Sigma$

$\quad \mathrm{ClrIns} = (\mathrm{Si} \times \mathrm{MTyp} \times \mathrm{Ri}) \underset{m}{\rightarrow} \mathrm{Ci}$

$\quad \mathrm{Conns} = \mathrm{Ci} \underset{m}{\rightarrow} \Xi$

value

$\quad \mathrm{SndAck}: \mathrm{Si} \times (\mathrm{MTyp} \times \mathrm{M}) \times \mathrm{Ri} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \mathbf{\Psi} \stackrel{\sim}{\rightarrow} \mathbf{\Psi}$

$\quad \mathrm{sc}: \mathrm{Si} \times \mathrm{M} \times \mathrm{Ri} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \mathrm{Sn} \stackrel{\sim}{\rightarrow} (\mathrm{R} \times \mathrm{T} \times \mathrm{Sn})$

$\quad \mathrm{cr}: \mathrm{R} \times \mathrm{Si} \times \mathrm{M} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \Xi \stackrel{\sim}{\rightarrow} \mathrm{R} \times \mathrm{T} \times \Xi$

$\quad \mathrm{ak}: \mathrm{R} \times \mathrm{M} \times \mathrm{Si} \stackrel{\sim}{\rightarrow} \mathrm{Rc} \stackrel{\sim}{\rightarrow} \mathrm{T} \stackrel{\sim}{\rightarrow} \mathrm{R} \times \mathrm{T}$

$\quad \mathrm{rc}: \mathrm{R} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \mathrm{Rc} \stackrel{\sim}{\rightarrow} \mathrm{R} \times \mathrm{T} \times \mathrm{Rc}$

$\quad \mathrm{cs}: \mathrm{R} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \Xi \stackrel{\sim}{\rightarrow} \mathrm{R} \times \mathrm{T} \times \Xi$

$\quad \mathrm{cl}: \mathrm{R} \times \mathrm{T} \stackrel{\sim}{\rightarrow} \mathrm{Sn} \stackrel{\sim}{\rightarrow} \mathrm{Sn}$

$\quad \mathrm{TimeLimit}: \mathrm{T}{\times}\mathrm{T}{\times}\mathrm{T}{\times}\mathrm{T}{\times}\mathrm{T}{\times}\mathrm{T} \rightarrow \mathbf{Bool}$

$\quad \mathrm{SndAck(s,(mt,m),r,t)}(\psi) \equiv$

$\qquad \mathbf{let}\ (\mathrm{cins,cnns},\sigma) = \psi\ \mathbf{in}$

$\qquad \mathbf{let}\ \mathrm{c{=}cins(si,mt,ri)},\ (\mathrm{ss,rs}){=}\sigma\ \mathbf{in}$

$\qquad \mathbf{let}\ (\mathrm{p,t'},\sigma_s){=}\mathrm{sc(s,m,r,t)(ss(s))}\ \mathbf{in}$

$\qquad \mathbf{let}\ (\mathrm{p',t''},\xi){=}\mathrm{cr(p,s,m,t')(cnns(c))}\ \mathbf{in}$

$\qquad \mathbf{let}\ (\mathrm{p'',t'''},\sigma_r){=}\mathrm{rc(ak(p',m,s),t'')(rs(r))}\ \mathbf{in}$

$\qquad \mathbf{let}\ (\mathrm{p''',t''''},\xi') = \mathrm{cs(p'',t''',s)}(\xi)\ \mathbf{in}$

$\qquad \mathbf{let}\ \sigma'_s = \mathrm{cls(p''',t'''')}(\sigma_s)\ \mathbf{in}$

$\qquad (\mathrm{cins,cnns} \dagger [\,\mathrm{c} \mapsto \xi'\,],$

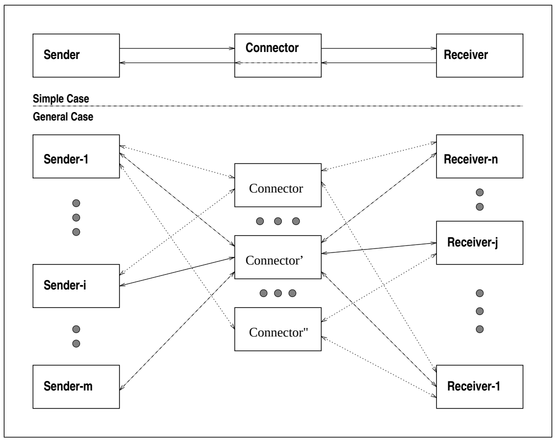
$\qquad\ (\mathrm{ss} \dagger [\,\mathrm{s} \mapsto \sigma'_s\,],\mathrm{rs} \dagger [\,\mathrm{r} \mapsto \sigma_r\,]))$

$\qquad \mathbf{end\ end\ end\ end\ end\ end\ end}$

$\quad \mathbf{post}\ \mathrm{TimeLimit(t,t',t'',t''',t'''',t''''')}$

Legend: MTyp designate message types. Ci connector (clearer) identifiers. Ξ designates connector states. R designates forwarding and reply reports. **Explanation:** The requirements are thought of as stipulating strict TimeLimits on the "processing" of message forwarding and acknowledgement returns.

Figure 15: Connection Requirements



Connection Frame Requirements

Issues like message transmission failures additionally renders the problem a data communications [hence multi–] frame problem.

— **Conclusion**

We have briefly reviewed Jackson's ([62]) Translation (JSP), Control, Information System, Workpiece and Connection Problem Frames from the "triptych" points of view of Application Domains, Requirements, and in two cases also Design. Our characterisations thus deviate in three ways from [62]: the "triptych" decomposition, more specifics as to what each frame "means", and their ("pseudo") formalisations.

Much work needs to be done before our approach can be claimed adequate.

I acknowledge initial prodding from Ms Veronica Argaaraz and Ms Natalia Romero both of National Univ. of San Luis, Argentina, for first suggesting formal characterisations of Jackson's Problem Frames.

C.4.5 Informality & Formality

At UNU/IIST we emphasize pairing *informal* with *formal* descriptions, and we further subdivide these into:

1. Informal Descriptions:

   — are expressed in a national, i.e. in a natural, colloquial language.

   (a) *Synopses:* Brief, introductory presentation of the very essence of the designated description.

   (b) *Narratives:* Detailed, complete & consistent explication of all relevant facets of the description target. Given the target, the description is indicative (if domain), optative (if requirements), or imperative (if software); etc.

   (c) *Terminologies:* For every professional term of the description target a careful definition must be given. In the domain theory stage the terms are the professional terms of the application problem domain. For the requirements and software stages and steps they are additionally those of information technology.[35] Throughout development the terminologies must be established, maintained (updated, extended) and used.

   Since national (natural, colloquial) language use, however professional, however precise, is fraught with problems of ambiguities, since it is difficult to maintain consistency and completeness, and since it is more convincing to carry out reasoning in a formal system, formal systems (i.e. languages + logic) are applied.

2. Formal Descriptions: So far using RAISE[36] + DC[37]

   (a) *Semantic Types:* Semantic individuals and composites are usually *what the whole thing is all about!* — i.e. denotations of the terms of our narrative. Descriptions here are usually in the form of abstract types (á la algebraic sorts) or concrete types (á la denotational domain equations). Either way: axioms or sub-type definitions with predicates are often used to suitably constrain the types.

   (b) *Syntactic Types:* Syntactic individuals and composites are abstractions of how we *speak about the essence!* — i.e. the designators of semantics, or: terms in our narrative and terminologies. Description means are as for semantic types.

   (c) *Semantics:* Semantics is about the relations between syntactic and semantic types. Descriptions are usually in the form of axioms and of pre/post or explicitly defined functions (or relations).

   (d) *Auxiliaries:* Oftentimes, to explicate a complex notion, auxiliary functions have to be provided. Descriptions are as for semantics.

[35] Information technology, incl. software terms may have to be revised wrt. existing dictionaries.

[36] RAISE: Rigorous Approach to Industrial Software Engineering, see special appendix section C.3 (starting page 63).

[37] DC: the Duration Calculi, see special appendix section C.1 (starting page 50).

Usually the various parts above are carefully integrated into a flowing textual presentation — easy for lay readers to follow!

We refer back to the closing comments of section 4.2 on page 26.

C.5 Infrastructures — A Research Topic

Most of UNU/IIST's advanced development projects are characterised by addressing infrastructure issues. We refer to section 1.2.2 on page 5.

In this section we shall illustrate two kinds of research issues. The aim is to understand more precisely, more formally: *What do we mean by an Infrastructure?*

C.5.1 Transport Infrastructures

We refer to special appendix sections E.1 (starting page 103) — E.3 (starting page 115) where we present three transport system domain models. Special appendix sections E.4 (starting page 122) represents two levels of abstractions of the models given in special appendix sections E.1 (starting page 103) — E.3 (starting page 115). That section (special appendix section E.4 (starting page 122)) illustrates that domains can be modelled in a variety of styles.

- Transport Graphs: See special appendix section E.4.1 (starting page 122)

  One set of models, also used by operations researchers (but with a different style of modelling), express transport systems in terms of graphs. Here the emphasis is on the networks first, then the flow of vehicles is superimposed.

- Transport Movement: See special appendix section E.4.3 (starting page 134)

  Another set of models, more typical perhaps of physics, model movement (first) and (then) the resources involved in movement. Movement is expressed in terms of state change and evolution (of traffic) as a sequence of movements. The traffic resources are of three kinds: the routes, the vehicles and the things transported.

These two views must then be co-ordinated: They are, so-to-speak, to be made concordant.

A number of (given and derived) properties of traffic can be expressed in terms of laws: axioms and theorems, respectively.

From the above points of view a transport infrastructure is now anything that satisfies the laws (of the two kinds of models and their concordance).

But this is basically only "speculation"! Much research need be done.

C.5.2 Public Administration Infrastructures

Usually we think in terms of a bureaucracy as a set of related hierarchical structures — that is: When seen organisationally. Seen operationally a bureaucracy can also be viewed as a "flat" set of actors, their actions and interactions.

In special appendix section E.9 (starting page 165) we present cursory models of the Vietnam Ministry of Finance's Taxation Department, and Budget Department and Treasury.[38] Some models reflect the hierarchical structure, others the ("flat") distributed system.

It seems, therefore, that a Public Administration Infrastructure is something that both portrays hierarchical, political structures and "flat", operational structures as well as relations between these.

But this is basically only "speculation"! Much research need be done.

C.5.3 An Infrastructure Research Agenda

UNU/IIST has identified the above possible ways of understanding infrastructures. But UNU/IIST can, at present, not prioritise its R&D so as to favour an in-depth study of this, or other emerging Theories of Infrastructures. Our partners in developing countries are naturally impatient and wishes to see "results now". Therefore our advanced development domain theory approach is as illustrated in special appendix sections E.1 (starting page 103) — E.13 (starting page 190).

We hope, however, to interest our more research-"positive" collaborators anywhere to pursue with us the more foundational study of infrastructure theories.

[38] For your information: The Taxation Department assesses the tax base of a country: conveys orders down the hierarchy to evaluate next years income, property, sales (or value added) tax possibilities as well as potential import & export duties, fees etc. Based on preliminary estimates the Budget Department in collaboration with other ministries can then establish frame budgets. Parliament can subsequently approve state budgets and the Taxation Department can now start collecting taxes. Collected taxes flow into the Treasury which is then able to disburse funds to all ministries. All this happens at various levels: state, provincial, district and commune — and in interleaved fashion: Last year's remaining actions, this year's collection, accounting & disbursement, and next year's predictions.

D UNU/IIST's Research Projects

In this appendix we cover the following research projects:

1. DeTFoRS: Real-time & Hybrid Systems App.: D.1
2. DesCaRTeS: Telecommunication Systems App.: D.2
3. CaSIno: System Implementation Platforms App.: D.3

D.1 DeTFoRS: Real-time & Hybrid Systems

DeTFoRS[39] is an acronym for Design Techniques for Real-time Systems. We refer to item 1 page 14 (section D.1).

We will illustrate the application of various Duration Calculi to interesting problems. See section C.1 for a brief introduction to Duration Calculi.

1. The Duration Calculus

   The Elevator Problem:

   Suppose that there are $0, \ldots, n$ floors. At each floor, there is a button by pressing which the lift is called, and a door which can be opened when the lift is at this floor. We choose the following state variables

$$\begin{array}{ll} floor_i: & \mathbf{Time} \to \{0,1\} \\ buttom_i: & \mathbf{Time} \to \{0,1\} \\ door_i: & \mathbf{Time} \to \{0,1\} \end{array}$$

   The lift is at floor i, and the button or the door at the i-th floor is pressed or open iff the value of $floor_i$, $buttom_i$ or $door_i$ is 1. A safety requirement is that the i-th door can only be opened if the lift is indeed at this floor:

$$SafetyReq = \Box \llbracket door_i \Rightarrow floor_i \rrbracket$$

   We can also specify several requirements concerning the service of the lift. The lift should arrive within t_s units of time if the button is pressed

$$F1 = \Box(\llbracket buttom_i \rrbracket \Rightarrow l \leq t_s \vee \Diamond \llbracket floor_i \rrbracket)$$

   The system must guarantee that when a floor is served, the door must be open for at least t_o units of time:

$$F2 = \Box(\llbracket \neg door_i \rrbracket ^\frown \llbracket door_i \rrbracket ^\frown \llbracket \neg door_i \rrbracket \Rightarrow l \geq t_0)$$

[39]This section is written by Zhou Chao Chen and Xu Qi Wen

The Gas Burner Problem:

A gas burner is either heating when the flame is burning or idling when the flame is not burning, and it alternates indefinitely between heating and idling. Usually, no gas is flowing while it is idling. However, when changing from idling to heating, gas must be flowing for a short time before it can be ignited, and when a flame failure appears, gas must be flowing before the failure is detected and the gas valve is closed. Hence, there may exist a time where gas is flowing and a flame is not burning, i.e. gas is *leaking*. A design of a safe gas burner must ensure that the time intervals where gas is leaking do not get too long.

Let us assume that the ventilation required for normal combustion would prevent dangerous accumulation of gas, provided that the proportion of leak time is not more than one twentieth of the elapsed time for any time interval being at least one minute long. This can be formalised as

$$Req = l \geq 60 \Rightarrow \ 20\textstyle\int Leak \leq l$$

Turning next to the task of design, certain decisions must be made as to how the real-time requirement is to be met. For example, for any period where the requirement is guaranteed, any leak (in this period) should be detectable and stoppable within one second; and to prevent frequent leaks it is acceptable that after any leak during this period the gas burner rejects switching on gas for thirty seconds. The design can be expressed as

$$\begin{array}{rl} Des1 = & (\lceil\!\lceil Leak \rceil\!\rceil \Rightarrow l \leq 1) \\ Des2 = & (\lceil\!\lceil Leak \rceil\!\rceil ; \lceil\!\lceil \neg Leak \rceil\!\rceil ; \lceil\!\lceil Leak \rceil\!\rceil \ \Rightarrow \ l \geq 30) \end{array}$$

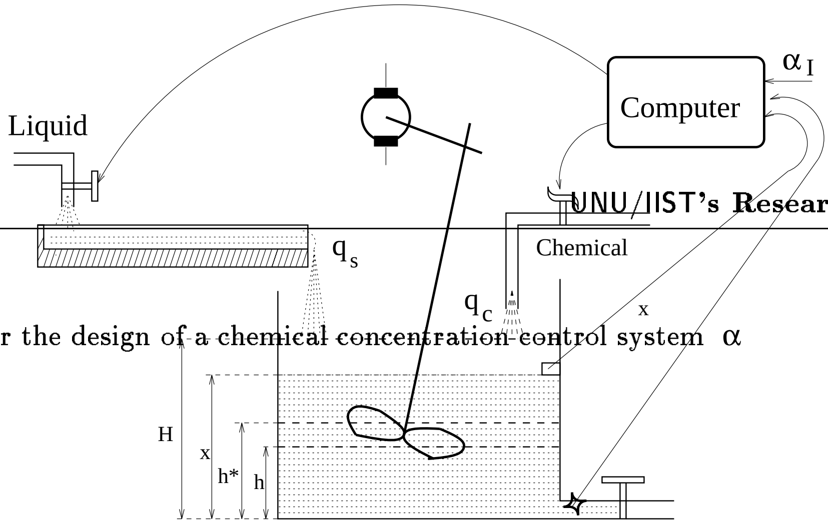
The correctness of the design is captured by the following fact

$$\Box Des1 \wedge \Box Des2 \Rightarrow Req$$

which can be formally deduced.

2. The Extended Duration Calculus & The Chemical Concentration Control Problem:

We consider the design of a chemical concentration control system α



Chemical Concentration Control

The main components of the system are a tank in which a chemical is dissolved in a liquid and a computer which controls the process. The principal requirement is that the output solution of the tank should be within a limited bound of the requested level and the solution should be sufficiently uniform. newpage

The concerned physical state variables are as follows:

- m_c: mass of the chemical in the tank
- m_s: mass of the liquid in the tank
- m_{c1}: mass of dissolved chemical in the tank
- α: concentration level,
- α_*: concentration level if all chemical was dissolved,
- q_c: rate of supply of the chemical
- q_s: rate of supply of the liquid
- Q: rate of outflow
- x: solution depth

They are all continuous functions from **T**ime to reals. The dynamics of the system are described by the following equations:

$$
\begin{array}{rcl}
\alpha & = & \frac{m_{c1}}{m_s+m_{c1}} \\
\alpha_* & = & \frac{m_c}{m_s+m_c} \\
\dot{\alpha} & = & 5(\alpha_* - \alpha) - \frac{\alpha(1-\alpha)}{m_s} q_s \\
\dot{m}_c & = & q_c - \alpha Q \\
\dot{m}_s & = & q_s - (1-\alpha)Q \\
x & = & k_1(m_c + 3m_s)
\end{array}
$$

The third equation just above characterises the change of α: $5(\alpha^* - \alpha)$ represents the effect from the dissolving process while $\frac{\alpha(1-\alpha)}{m_s} q_s$ represents the effect from adding the liquid. The last equation relates the depth of the solution to the quantities of the chemical and the liquid in the tank.

We next examine the requirements. The main objective is that the concentration level of the solution flowing out of the tank should be close enough to the requested one. In particular, it is required that the difference between α and α_I should not be more than $0.1\%\alpha_I$. Formally, this is expressed as:

$$Req_1 = \Box [\![Q > 0 \Rightarrow |\alpha - \alpha_I| \leq 0.1\%\alpha_I]\!]$$

To ensure that the solution is sufficiently uniform, it is stipulated that $\alpha_* - \alpha$ should not be more than 2%. Let

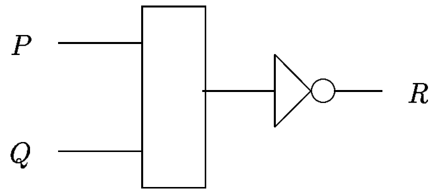
$$Req_2 = \Box [\![\alpha_* - \alpha \leq 2\%]\!]$$

3. The Mean Value Duration Calculus & The Combinational Circuit Problem:

A combinational circuit is a circuit made from logic gates, and its output voltage/signals are determined by its *current* inputs (i.e. no storages in the circuit).

In terms of circuits, let state P stand for a wire named P. $P(t) = 1$ means that wire P connects to power at time t, and $P(t) = 0$ means wire P connecting to ground at t. Germ-ships $\uparrow P$ and $\downarrow P$ represent changes of voltage of wire P, and the other two germ-ships $\top P$ and $\perp P$ represent stable voltage of the wire. Thus they describe the dynamic behaviour of a wire.

The following figure shows a NOR circuit made from an OR gate and an inverter NEG.



NOR Circuit $R = \neg(P \vee Q)$

Using the *Germ-ship Calculus* we can calculate the behaviour of the output wire R in terms of the behaviour of the inputs P and Q. For instance:

$$\begin{array}{lll} \uparrow R & \Longleftrightarrow & \uparrow\neg(P \vee Q) \\ & \Longleftrightarrow & \uparrow(\neg P \wedge \neg Q) \\ & \Longleftrightarrow & (\uparrow\neg P \wedge \uparrow\neg Q) \vee (\uparrow\neg P \wedge \top\neg Q) \vee (\uparrow\neg Q \wedge \top\neg P) \\ & \Longleftrightarrow & (\downarrow P \wedge \downarrow Q) \vee (\downarrow P \wedge \bot Q) \vee (\downarrow Q \wedge \bot P) \end{array}$$

The calculation shows that the output rises just when both inputs fall or one falls and the other stays low. Similarly we can calculate the other three germ-ships of the output with the calculus. The calculus formalises the conventional theory of combinational circuits.

Timing issues can also be discussed. For example, the transmission delay d of the rising signal of R in the example can be formulated as

$$\uparrow\neg(P \vee Q); (l = d) \Rightarrow (l = d); \uparrow R$$

and the inertial delay d as

$$\uparrow\neg(P \vee Q); (\llbracket\neg(P \vee Q)\rrbracket \wedge (l = d)) \Rightarrow (l = d); \uparrow R$$

D.2 DesCaRTeS: Telecommunications Systems

DesCaRTeS[40] is an acronym for Design Calculi Research for Telecommunications Systems.

The research project DesCaRTeS is focusing attention on formal techniques, and tools in support of them, to complement SDL for more rigorous approaches to software development in telecommunications. One of its aims is to enhance the possibilities for advanced analysis of SDL descriptions, in particular analysis with respect to time dependent behavioural aspects of the systems described.

In this appendix, first of all, we give a small example to illustrate how time related behavioural aspects of systems can be described in SDL. The translation of the SDL description to process algebra, according to the process algebra semantics of SDL outlined in Appendix C.2, is presented as well. This semantics indicates that there is an alternative way to describe the same behaviour, using process algebra. After that, we outline the use of the process algebra semantics for analysis of time dependent behavioural aspects of systems specified using SDL. This demonstrates that some analysis of time related properties is already possible if we make direct use of the process algebra semantics, but it in a rather clumsy way.

Description using SDL

The example concerns the control component of a simple telephone answering machine. The specification is due to Mauw [70]. In order to control the telephone answering, the control component of the answering machine has to communicate with the recorder component of the answering machine, the telephone connected with the answering machine, and the telephone network. When an incoming call is detected, the answering is not started immediately. If the incoming call is broken off or the receiver of the telephone is lifted within a period of 10 time units, answering is discontinued. Otherwise, an off-hook signal is issued to the network when this period has elapsed and a pre-recorded message is played. Upon termination of the message, a beep signal is issued to the network and the recorder is started. The recorder is stopped when the call is broken off, or when 30 time units have passed in case the call has not been broken off earlier. Thereafter, an on-hook signal is issued to the network. A controller for the answering machine should control the telephone answering in such a way that the telephone answering machine functions as informally stated above. Informally, it should be immediately clear that the behaviour of the controller described in Figure 16 provides for the proper control, except, perhaps, for the time dependent aspects which is described using timers.

[40] This section is written by Kees Middelburg

```
system AnsweringControl
  signal inccall;
  signal endcall;
  signal offhook;
  signal onhook;
  signal beep;
  signal rcvlifted;
  signal playmsg;
  signal endmsg;
  signal startrec;
  signal stoprec;
  signal wtimer;
  signal rtimer;

  signalroute fromnetwork from env to AMC
    with inccall, endcall;
  signalroute tonetwork from AMC to env
    with offhook, onhook, beep;
  signalroute fromtelephone from env to AMC
    with rcvlifted;
  signalroute torecorder from AMC to env
    with playmsg, startrec, stoprec;
  signalroute fromrecorder from env to AMC
    with endmsg;

  process AMC
    start;
      nextstate begin;
    state begin;
      input inccall;
        set(now+10,wtimer);
        nextstate waiting;
    state waiting;
      input endcall;
        reset(wtimer);
        nextstate begin;
      input rcvlifted;
        reset(wtimer);
        nextstate begin;
      input wtimer;
        output offhook via tonetwork;
        output playmsg via torecorder;
        nextstate answering;
    state answering;
      input endcall;
        nextstate end;
      input endmsg;
        output beep via tonetwork;
        output startrec via torecorder;
        set(now+30,rtimer);
        nextstate recording;
    state recording;
      input endcall;
        reset(rtimer);
        output stoprec via torecorder;
        nextstate end;
      input rtimer;
        output stoprec via torecorder;
        nextstate end;
    state end;
      input none;
        output onhook via tonetwork;
        nextstate begin;
  endprocess;
endsystem;
```

Figure 16: Answering machine controller described in SDL

Description using Process Algebra

The translation of the above SDL system description to process algebra, according to the process algebra semantics of SDL sketched in Appendix C.2, is given below. Thus, we also illustrate how time related behavioural aspects of systems can be expressed using process algebra.

The behaviour of the control component of the answering machine is described by the following process expression:

$$\tau_{I \cup \{\mathbf{t}\}}(\lambda_{G_0}(AMC \parallel Env))$$

where the process AMC is recursively defined by the set of equations given in Figure 17. This

$$
\begin{aligned}
AMC &= BEGIN\\
BEGIN &= \underline{\underline{input}}(inccall, \{\,\}, amc)\cdot \underline{\underline{set}}(now+10, wtimer, amc)\cdot WAITING+\\
&\quad\ waiting(amc) :\!\rightarrow \sigma_{\mathsf{rel}}(BEGIN)\\
WAITING &= \underline{\underline{input}}(endcall, \{\,\}, amc)\cdot \underline{\underline{reset}}(wtimer, amc)\cdot BEGIN+\\
&\quad\ \underline{\underline{input}}(rcvlifted, \{\,\}, amc)\cdot \underline{\underline{reset}}(wtimer, amc)\cdot BEGIN+\\
&\quad\ \underline{\underline{input}}(wtimer, \{\,\}, amc)\cdot \underline{\underline{output}}(offhook, amc, \mathsf{env})\cdot\\
&\qquad\qquad \underline{\underline{output}}(playmsg, amc, \mathsf{env})\cdot ANSWERING+\\
&\quad\ waiting(amc) :\!\rightarrow \sigma_{\mathsf{rel}}(WAITING)\\
ANSWERING &= \underline{\underline{input}}(endcall, \{\,\}, amc)\cdot END+\\
&\quad\ \underline{\underline{input}}(endmsg, \{\,\}, amc)\cdot \underline{\underline{output}}(beep, amc, \mathsf{env})\cdot\\
&\qquad \underline{\underline{output}}(startrec, amc, \mathsf{env})\cdot \underline{\underline{set}}(now+30, rtimer, amc)\cdot RECORDING+\\
&\quad\ waiting(amc) :\!\rightarrow \sigma_{\mathsf{rel}}(ANSWERING)\\
RECORDING &= \underline{\underline{input}}(endcall, \{\,\}, amc)\cdot \underline{\underline{reset}}(rtimer, amc)\cdot\\
&\qquad\qquad\qquad \underline{\underline{output}}(stoprec, amc, \mathsf{env})\cdot END+\\
&\quad\ \underline{\underline{input}}(rtimer, \{\,\}, amc)\cdot \underline{\underline{output}}(stoprec, amc, \mathsf{env})\cdot END+\\
&\quad\ waiting(amc) :\!\rightarrow \sigma_{\mathsf{rel}}(RECORDING)\\
END &= \underline{\underline{tt}}\cdot \underline{\underline{output}}(onhook, amc, \mathsf{env})\cdot BEGIN + waiting(amc) :\!\rightarrow \sigma_{\mathsf{rel}}(END)\ .
\end{aligned}
$$

Figure 17: Answering machine controller defined using process algebra

means that the system behaves as the process AMC executed, in parallel with the environment process Env, in state G_0. Besides, the actions in the set $I \cup \{\boldsymbol{t}\}$ are hidden. In the initial state G_0, the system time is zero, AMC's input queue is empty and AMC has no active timers. Env and I are parameters of the semantics, as explained in Appendix C.2. A delay to a future time slice is only possible when an input action has to be performed but the condition $waiting(amc)$ evaluates to true, which means that the input queue of AMC has to be empty. A delay to the next time slice will cause the system time to be incremented by one and timer signals to be put into the input queue of AMC for timers expired at that time.

The two definitions of the process AMC presented in Figures 16 and 17 look alike. The main difference is that the latter definition can be subjected to equational reasoning – using the axioms of the discrete time process algebra used.

Analysis using Process Algebra

The following is one of the time related properties of the telephone answering machine described above that should be respected by the controller's behaviour:

when the off-hook signal is issued to the network, nothing has happened since the detection of the last incoming call and meanwhile 10 time units have passed.

Let $output'(sig)$ be the action that occurs when action $output(sig)$ has been executed in a state, and let Snd be the set of actions $output'(sig)$ where the sender of sig is the environment env. Then the above-mentioned property can be expressed using an equation:

$$\tau_{Snd-\{output'(offhook,amc,\mathsf{env})\}}(S) = S'$$

where S is the meaning of the system description given above, i.e.

$$S = \tau_{I\cup\{\mathbf{t}\}}(\lambda_{G_0}(AMC \parallel Env))$$

and S' is recursively defined by

$$S' = \tau_{Snd}(S) + \underline{\underline{output'}}(inccall, \mathsf{env}, amc) \cdot \sigma^{11}_{\mathsf{rel}}(\underline{\underline{output'}}(offhook, amc, \mathsf{env})) \cdot S' \ .$$

We claim that this equality can be proved by means of the axioms of $\mathrm{ACP}^{\tau}_{\mathrm{drt}}$ and its extensions used (see also Appendix C.2), if we consider the meaning of the system description obtained by taking the environment process $Env^{\mathsf{st}}_{\kappa}$ and the set of actions I^{st}_{κ} for Env and I, respectively. Both $Env^{\mathsf{st}}_{\kappa}$ and I^{st}_{κ} are precisely defined in [12]. If one takes $Env^{\mathsf{st}}_{\kappa}$ for Env, one gets an open meaning corresponding to the viewpoint that the only assumptions about the environment that the system can rely on, are the ones made explicit in the signal route definitions. If one takes I^{st}_{κ} for I, one gets an abstract meaning corresponding to the viewpoint that only the communication of the system with the environment is observable.

This approach to analyse properties of a process is very convenient for properties that concern its behaviour as a whole. However, it may be inconvenient or even impossible to express a property that covers only one detail of the behaviour of a process using an equation. Therefore, we are also studying – in the framework DesCaRTeS – logics to analyse properties of processes expressed using discrete time process algebra (see [9, 10]). This is only one of the research topics that has to be addresses in order to enable better grounded validation of SDL specifications, which is one of the primary aims of DesCaRTeS. Another one is to enable design steps made using SDL to be justified by formal verification.

D.3 CaSIno: Platforms for Systems Implementation & Integration

CaSIno[41] is an acronym for Categories for Systems INtegratiOn!

CaSIno is an acronym for Categories for Systems Integration, reflecting a premature (!) decision as to the tool (category theory) applied to alleviate some of technical problems with the integration of formal and informal methods in software engineering:

- Formal methods allow software be designed with proofs of correctness, in principle free of design mistakes. An example is RAISE – Rigorous Approach to Industrial Software Engineering. The story is well-known: formal methods are perceived in the software industry to be complex, unfriendly, and of little practical use.
- Industry applies informal methods and, under the pressure to deliver products faster (quality in the second place), more and more relies on component-based development, rapidly engineered from libraries of existing components. Increasingly, this integration is not confined to the barriers created by programming languages, incompatible machines, protocols that cannot talk to each other.. It is not even constrained to take place off-line; see intelligent software agents that are able to dynamically explore, and make use of, the services available over the network, some not existing at the time the agent was created! An example is CORBA – Common Object Request Broker Architecture.

The last example has been put forward to provide evidence that the integration of formal (e.g. RAISE) and informal component-based methods (e.g. CORBA) in software engineering is well justified; more – that it is very much needed. The cornerstone of component-based development of software is the separation of component interfaces and implementations. This requires ways to find out, from the interface description only, if one component (a server) provides what another (a client) wants. The current practice of matching clients and servers is based on purely syntactic criteria of service types and takes into account no semantic information, except for describing informally "what the operations of the component are supposed to do". This makes it impossible to interpret such information automatically, leaving the tedious task of exploring interface repositories for the needed services to software designers (off-line integration) or users (on-line integration); the latter clearly contradicting the very idea of the on-line integration! Conclusion: component-based software development requires (semantically) well-founded methods of describing, not only coding, functionality of software.

The project works towards this goal. The basic idea is to extend interface descriptions for software components by axioms which represent the expected behaviour of the components. This will allow for two methods of component-based development:

- Such axioms can in principle allow for proofs that the implementation of the component is indeed correct with respect to its interface; it not only delivers the operations within the

[41] This section is to be written by Tomasz Janowski

types stated but also results that satisfy given properties. This allows for flexible means of matching components, given that properties of the required service, specified loosely, are indeed satisfied by the server.

- Such axioms can also be validated on-line, based on software generated from the interface description which "wraps" the component implementation and monitors all interactions on its interface. It is transparent, unless the interaction violates the axioms, in which case it raises an exception. This low-cost approach does not require proofs about component implementations. The only cost is the formal description of the interface and some penalty in performance due to monitoring software. The monitoring software can be generated automatically.

We want to follow this idea for RAISE and CORBA in particular, with RSL becoming a formal extension of the CORBA IDL (Interface Definition Language). The stages are as follows:

- Produce a model of the CORBA environment in RSL, capture aspects of distribution and heterogeneity and describe how CORBA helps to overcome the problems caused by them.
- Extend the CORBA Interface Definition Language (IDL) by adding axioms to express the promised behaviour of component software, based on the formalisation of the IDL type hierarchy in RSL.
- Provide a further extension of IDL by axioms representing the modal and temporal properties of component software; this will require an extension of RSL itself.
- Off-line verification of component software: translate all components back to RSL and investigate the properties of the integrated modules using RSL tools, as if the components were correct refinements of their IDL descriptions.
- On-line validation of component software: specify formally and then design software that given an extended IDL description will produce: (1) plain IDL and (2) a program that will monitor all interactions of a component with its environment and will raise an exception as soon as it detects a violation of the axioms in the extension. Integrate such software within the CORBA environment.

This project is still in the early stages of its development. Unlike other research projects the staff involved had little previous experience in the area, and it has only involved a small amount of effort. It does, however, deal with issues likely to be relevant to the advanced development projects.

E UNU/IIST's Software Technology Projects

E.0.1 Models to be shown

In special appendix sections E.1 (starting page 103) — E.13 (starting page 190) we present 13 domain models. They range from railways (special appendix section E.1 (starting page 103)), air traffic (special appendix section E.2 (starting page 108)), and metropolitan transport (special appendix section E.3 (starting page 115)), via manufacturing (special appendix section E.5 (starting page 141)), multi-lingual scripts (special appendix section E.6 (starting page 148)), airline business (special appendix section E.7 (starting page 155)), radio telephony (special appendix section E.8 (starting page 160)), and taxation & treasury administration (special appendix section E.9 (starting page 165)), to external debt management (special appendix section E.10 (starting page 170)), financial industries & services (special appendix section E.11 (starting page 174)), decision support systems for sustainable development (involving geo- & demographic information systems [GIS, etc.]) (special appendix section E.12 (starting page 183)), and enterprise resource monitoring & control (special appendix section E.13 (starting page 190)).

This is quite an impressive spectrum. It should, however, in all fairness be said that not all of the underlying UNU/IIST projects has progressed equally deep into each of these areas.

E.0.2 Comments on the Models to be Developed

Special appendix sections E.4 (starting page 122) represents two levels of abstractions of the models given in special appendix sections E.1 (starting page 103) — E.3 (starting page 115). This section (special appendix section E.4 (starting page 122)) illustrates that domains can be modelled in a variety of styles.

In special appendix section E.3 (starting page 115) we present a rather concrete model of bus, metro train and taxi networks and traffic. We just present it to you, in one step. Let us say from the very beginning: *"That is not the way we like to do it!"* So why are we not presenting a model for metropolitan transport the way we think it ought to be done? For at least three reasons:

1. "Back–of–an–Envelope" Sketch

   When initially, i.e. at the very first outset of, contemplating a possibility of tackling a commercial opportunity [or even a scientific investigation] into the matter of a large scale development of [respectively research into] an infrastructure system, it is often a good idea to see how an abstraction might evolve, or to "fix" which is the target of study. Here these individuals and phenomena are streets, with busses and taxis, bus stops and taxi ranks, metro lines and stations, bus and train schedules, etc. So the "rough", "back–of–an–envelope" sketch focus on these — perhaps only to be thrown away. Maybe the sketch can serve as the basis for a first proposal to the client.

2. As an Example "perhaps not to be followed"!

   But these "backof-an-envelope" sketches are usually "too concrete"! Either they are biased wrt. means of implementation and their validation and their correctness proofs. That is: they may "bind" implementors in undesirable ways. (*If the "sketch" identify bus and train schedules as separate kinds of objects, and bus and train nets likewise, then to implement them in the form of one kind of timetables, respectively nets, may become awkward.*)

3. To Contrast with Desirable Models!

   In special appendix sections E.4.1 on page 122 and E.4.3 on page 134 we present two transport models. The first (E.4.1 on page 122) is a first abstraction of the one given in the present section. The latter (E.4.3 on page 134) perhaps represents the best departure point for a series of Domain Theory descriptions of general and special concepts of transportation.

   The 'parameterisable' model of special appendix section E.4.1 (starting page 122) also represents an abstraction of the railway system model of special appendix section E.1 (starting page 103).

E.0.3 On "*Why are we showing You these Models?*"

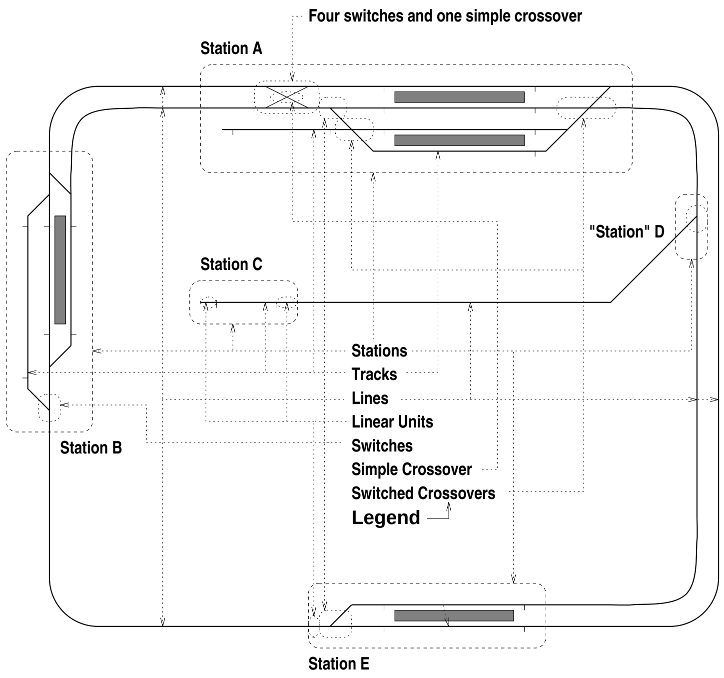
— and *What are we illustrating?*

Briefly we can give the following answers:

- *To show the breadth and depth of* UNU/IIST*'s research & development.*
- *To illustrate what we mean by* Domain Theories.
- *To argue that formal specification can be applied to new areas.*
- *To exemplify styles of modeling.*
- *To discuss modeling techniques.*
- *To show how* Domain *models relate to and contrast with* Requirements.
- *To contrast this again with* Software Architecture & Program Organisation.

  Cf. items 1 on page 69 and 30 on page 69, and footnote 30 on page 69.
- *To encourage further research.*
- *To advocate the method step of* Domain Modeling

Figure 18: A Rail Net



A Railway Net with 5 Stations and 7 Lines

E.1 RaCoSy: Railway Computing Systems

RaCoSy[42] is an acronym for Railway Computing Systems. Another acronym, PRaCoSy stood for PRC (People's Republic of China) RaCoSy!

PRaCoSy was UNU/IIST's first project. It eventually became a relatively large, advanced software development project. The initial problem of PRaCoSy was that of *train dispatch.*

A picture is worth a thousand words! We refer to Figure 18.

Some of the *thousand words* could be:

E.1.1 Railway System Scope

In RaCoSy, UNU/IIST enlarged the Domain Theory scope to cover the "fuller" application domain of railways:

[42] This section is written by Dines Bjørner

1. From long range planning & development of railway resources [acquisition or "down-sizing"][43] and services: *rail net development, new (or disbanded) lines, new (or disbanded) stations, new (or disbanded) train (passenger and freight) services*, etc.;
2. via medium range resource allocation & scheduling: *time-tabling, and staff, line, track, car and traffic allocation & scheduling*, etc.,
3. to daily, operational handling: *passenger ticketing (and travel reservation), freight handling, train dispatch, signalling, client information services (platform displays, freight tracing), shunting, marshaling, rolling stock monitoring*, etc.; and
4. finally ending with statistics gathering (to be fed back to the actions mentioned in items 1 and 2.

Dependent on which individual or combination of the above facets are subject to requirements capture, the railway computing problem belongs to an individual or a combination of the Control, Information, Workpiece, [Clearing] Transaction, or Connection Frames.

E.1.2 Elements of a Domain Theory

The RaCoSy Domain Theory 'unfolds' as follows:

- The Rail Net: cf. Figure 18 on the page before
  - Units (U),[44] with Connectors (C), as the "smallest" elements from which nets, see below, are composed.
  - Paths (P) as pairs of connectors, designating direction of transport flow through a unit, unit States (Σ) as set of currently 'open paths', and flow capability (Ω) as set of potential unit states.
  - Nets (N), as suitably constrained composition of units into lines (L) and stations (S), the former consisting of sequences of linear units and the latter with (line-like) tracks (Trk) and otherwise consisting also of other varieties of connected units: junctions, cross-overs, etc. Tracks are either platform, by-pass, siding, marshaling yard (feeder fan-in and yard fan-out) tracks, or other.[45] Lines connect pairs of stations.[46]
  - Open, closed and train routes (Rt) as sequences of units and/or connectors (of these units).

[43] See special appendix section E.13 (starting page 190).

[44] In figure 18 on the preceding page we only indicate some units, and only "point" to some junctions (switches).

[45] In figure 18 on the page before we only show platform, (one) by-pass, and siding tracks.

[46] In figure 18 on the preceding page we show a situation where, along what is seemingly a line from Station A to Station E there is a switch. Thus the "apparent" line is "broken" into two and the single switch unit "effectively" (and certainly according to Chinese Railway Ministry "rules & regulations") becomes a station (Station D). Station E has no platform. Several stations have sidings (i.e. tracks with no "adjacent" platform.

  - Managed rail nets MR as time dependent rail nets.

  The Rail Net description includes descriptions of the kinds of routes into, out from and through stations from and to lines — enabling implicit descriptions of signalling, route planning and hence line and station management.

- Traffic, Timetables, Schedules, Scheduling & Re-Scheduling:
  - Traffic (TF) as a time dependent (hence managed) rail nets and train positions (TP) constrained so as to preserve certain laws: continuity, progress, etc., but not avoiding train crashes (as such are realities).
  - Timetables (TT) which lists train journies (J). The latter consists of station visits (SV), etc.
  - Schedules (SC) which can be understood as discrete projections of traffics, and scheduling and rescheduling (functions) which constructs and reconstructs ("repairs" current) schedules, the latter in response to observed discrepancies between traffic and schedules. A usually infinite set of acceptable schedules can be constructed from managed rail nets and timetables.
- Shunting, Marshaling Yards and Marshaling:
  - Shunting as the movement of train cars within stations and the decomposition and composition of trains into, respectively from car sequences.
  - Marshaling yards as special (parts of) stations with humps. feeder fan ins, and yard track fan outs.
  - And Marshaling as the fanning in (to the hump) of feeder trains and the merging of their cars (over the hump) into sequences of cars (train trunks or stems).
- Passengers and Freight:
  - Ticket (or travel, journey) reservation and actual journey, travel inquiries, etc.
  - Freight reservation and handling, including tracing.
- Net and Services Development:
  - Development or downgrading of nets and services: insertion (or deletion) of new (existing) facilities [lines, stations, tracks, timetable entries, etc.].
  - Implications for schedules, scheduling, traffic, shunting, marshaling, passengers and freight.
- *&c.*

type
  C, U, P = C×C,

Σ = P-**set**, Ω = Σ-**set**,
N, L, S, Trk,
Rt, TR = {| rt | rt:C^* • wf_Rt(rt) |}
value
obs_N_Ls: N $\rightarrow$ L-**set**, obs_N_Ss: N $\rightarrow$ S-**set**,
obs_N_Us: N $\rightarrow$ U-**set**, obs_L_Us: L $\rightarrow$ U-**set**, obs_S_Us: S $\rightarrow$ U-**set**,
obs_S_Trks: S $\rightarrow$ Trk-**set**, obs_Trk_Us: Trk $\rightarrow$ U-**set**,
open_Rt: Rt $\times$ N $\rightarrow$ **Bool**
type
T, Tn,
MR = {| mr | mr:(T $\rightarrow$ N) • wf_MR(mr) |},
TF = {| tf | tf:(T $\rightarrow$ (N $\times$ TP)) • wf_TF(tf) |},
TP = Tn $\underset{m}{\rightarrow}$ TR
SC = TF-**infset**,
TT = Tn $\underset{m}{\rightarrow}$ J,
J = {| j | j:SV-**set** • wf_J(j) |},
SV = {| sv | sv:(S $\times$ T $\times$ T) • wf_SV(sv) |},
value
TF_on_SC: TF $\times$ SC $\rightarrow$ **Bool**
TF_on_SC(tf,sc) $\equiv$ tf $\in$ sc

Etcetera! This 'etcetera' refers to virtually hundreds of axioms, and predicate, observer, generator and auxiliary functions that expresses the theory — of which we have only shown "the top of the ice berg".

Some of the observer functions are not computable. Take for example *obs_N_Us*. But they can be manifested. You, the railway official show me, the computing systems engineer, your net (n). Together we, you and I, can then "compute" *obs_N_Us(n)*! Cf. figure 18 on page 103.

E.1.3 Requirements: A "Running Map" System &c.

$\boxed{\text{More to come}}$

Report [20] outlines, informally and formally a number of alternative sets of requirements covering a larger part of the spectrum outlined in section E.1.1 on page 103.

E.1.4 Software: A "Running Map" System &c.

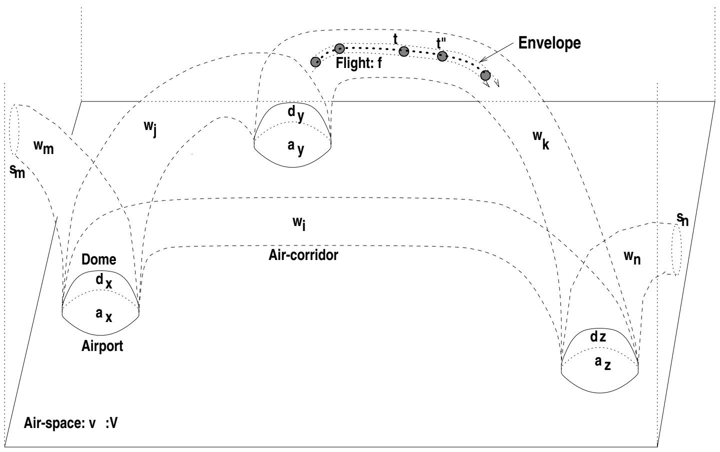
$\boxed{\text{More to come}}$

Report [20] finally outlines, informally and formally a software system architecture that will allow individual software sub-system packages for each of the alternative requirements also outlined in [20].

E.1.5 Discussion

The PRaCoSy and RaCoSy project activities stretches over now four years. Some 50 internal technical reports have been published in addition to a number of published papers [75, 69, 20]

Figure 19: An Air Traffic Space



Air-space, v, with three Airports, three internal and two external Air-corridors, and two "Intersections"
A fragment of a flight, f, is shown, as are discrete envelope positions, two of which are at times t and t"

E.2 ATC'2000: Air Traffic

ATC'2000[47] is an acronym for Air TraffiC ("Year 2000" (!)). The 'C' also hints at *air traffic Control* — a Control Frame problem.

E.2.1 Air Space

Figure 19 may illustrate that *a picture is worth a thousand words:*

Some of the *thousand words* could be:

An air-space (V) contains (α) of a number of internal airports (A), with each such airport having a dome (D) (also known as a terminal area (TMA)), and a number of internal and external air--corridors (W). Internal air-corridors connect (β) (domes of) internal airports, while external air-corridors either connects (γ) pairs of internal and external airports (the latter are not shown in figure 19), or go through (δ) the air-space connecting two external airports (also not shown).

More than one air-corridor may connect two (external or internal) airport (also this is not shown in figure 19). Air-corridors are directional: from one to another airport (but we show only one internal air-corridor for each pair of internal airports).

Thus there may be air-corridors from external airports (A-**set**) to internal or other external airports, but not necessarily vice-versa; and similarly: from internal to (A-**set**) an external

[47]This section is written by Dines Bjørner

airport, but not necessarily vice-versa.

External air-corridors need only be defined within the air-space.

A formalisation is worth an infinite number of pictures!

type V, A, D, W
  AS′ :: v:V xi:A-**set** xo:A-**set**
      g:(A $\overrightarrow{m}$ (D × (A $\overrightarrow{m}$ W-**set**)))
  AS = {| **as** | **as**:AS′ • is_wf_AS(**as**) |}

Types V, D, and W are not mode-led: they are thought of as simple, continuous topological spaces.

The air-space welformedness predicate (is_wf_AS) secures the various constraints implied in the narrative: $\alpha, \beta, \gamma, \delta$, that is: 'connections' of corridors to domes, where appropriate; described corridors start, end or go through the air-space, as appropriate; different airport domes do not overlap; etc.

Given the definition of the air-space (AS) with its spatial volume delineation (V) and corridors (W), it is now possible to define possible *intersection surfaces* between corridors and the air--space volume surface, whether two corridors intersect, etc. It is is also possible to compute a mean length of an internal corridor.

E.2.2 Timetables

Flights (identified by unique flight names (Fn)) are time-tabled, with a flight entry listing journies. A journey contains, amongst others, a list of airport visits. An airport visit informs, amongst others, about flight arrival and departure times ($T \times T$), etc.

type TT, Fn, J, AVs, T
value
  obs_Flights: TT → Fn-**set**
  obs_Journey: Fn × TT $\stackrel{\sim}{\rightarrow}$ J
  obs_AVisits: J → AVs
  obs_AAvDeps: AVs → (A × (T × T))*

Various axioms (or sub-type definitions for *TT, J, AVs,* etc.) properly constrain timetables.

Given an air-space and a timetable one can characterise their conformance:

value
  inv_AS_TT: AS $\times$ TT $\rightarrow$ **Bool**

The timetable should mention at least one airport internal to the air-space; if flight "legs" are listed between internal [or a pair of internal and external] airports then there must be a [not necessarily fully defined) air-corridor in the proper direction between them; the flight time of an internal "leg" should be commensurate with the mean length of some of the connecting internal corridors; etc.

E.2.3 Traffic

Figure 19 on page 108 shows a fragment of one flight's trajectory.

Traffic (*TF*) specifies for each point in time the position, (P), of a number of flights. Given a flight we can speak of its position trajectory *PTraj*: the infinite list of flight positions, starting with an airport and hopefully ending with an airport. The trajectory may "touch" several other airport, i.e. have positions that fall within domes and "glue" onto dome "bottoms" (the tarmac).

Let T now designate a dense set of real valued time points.

type P
  $\mathrm{TF}' = \mathrm{T} \rightarrow ((\mathrm{F} \underset{m}{\overrightarrow{\ \ }} \mathrm{P}) \times \mathrm{AS})$
  $\mathrm{TF} = \{|\ \mathrm{tf} \mid \mathrm{tf{:}TF}' \bullet \mathrm{is\_wf\_TF(tf)}\ |\}$
  $\mathrm{PTraj} = \mathrm{P}^{\omega}$
value
  projTF: F $\times$ TF $\stackrel{\sim}{\rightarrow}$ PTraj

P can be thought of as a topological, dense space. The traffic well-formedness predicate, is_wf_TF, describes appropriate constraints: flight trajectories must be continuous (this also insures the absence of "ghost–flights"[48]); must start in airports (etc., but need not necessarily end in airports!); etc.

Given a well-formed traffic one can define predicates which characterise "near-misses", "crashes" and, given the air-space, also the ground are of the crash. One may think of flights (F) containing all relevant information about the aircraft, the flight name (and number), its velocity, acceleration or deceleration, its mass, fuel reserves, orientation, etc. For each of these one may postulate observer functions. If so, then any flight trajectory can be further checked for conformance: consistency, for example, between flight velocity & acceleration at certain times and suitable next positions!

[48]For all times t and t'' of a flight *fn* such that t is before t'', it must be the case that the flight also has a position at all times t' between t and t''.

E.2.4 Scheduling

A schedule (SC) is a discrete version of projected traffic. Projection may, for example, list only the flight name, but give a larger envelope E within which positions are expected to be contained. One can, as above, but now wrt. envelopes, speak of their trajectory, a finite list of envelopes, as well as a "smoothened" version, *SETraj*: an infinite ("space-fitted") list of envelopes. Finally one can check whether a position trajectory is contained with a smoothened envelope trajectory, and whether suitable fragments of the latter are contained with air-corridors:

type E
  SC′ = T $\overrightarrow{m}$ (Fn −m− > E)
  SC = {| sc | sc:SC′ • is_wf_SC(sc) |}
  ETRaj = E^*
  SETraj = E^ω
value
  smoothen: E−List $\tilde{\rightarrow}$ E^ω
  contain_Trajs: PTraj × SETraj → **Bool**
  contain_SETraj_W: SETraj × W → **Bool**

E, and SETraj are suitable, continuous topological spaces.

Given a real traffic, TF, and a scheduled traffic, SC, one may check whether the real traffic follows the scheduled traffic. Since we are dealing with a Domain Theory you may think of both traffics extending well into the past and into the future. A check may thus reveal the first time t any real flight f is "sufficiently" deviating from its schedule.

value
  check: SC × TF $\tilde{\rightarrow}$ T × F
  check(sc,tf) **as** (t,f) **pre** ... **post** ...

The pair (t,f), if it exists, i.e. if there is a flight discrepancy, designates the first time in traffic tf when a flight deviates from its plan in sc. (t,f) then designates that time and that flight.

value
  sch: TT × AS $\tilde{\rightarrow}$ TF-**infset**

Finally the function sch generates a possibly infinite set of traffics all of which satisfy the given air space and timetable. Etcetera: further functions can then be defined to deal with selecting "optimal" schedules, with rescheduling, etc.

E.2.5 Discussion

The ATC'2000 example affords us to briefly touch upon a number of methodological issues:

0. Summary:

   We have briefly sketched basic elements in a model of a domain of air traffic. The model took as its four most important elements those of *air-space, timetables, traffic* and schedules.

1. Further Domain Intrinsics:

   We can now go on to describing properties of rescheduling, etc.

   The above constitutes an essence, not of transport, see special appendix section E.4 (starting page 122), but of air traffic, we claim. As such the part of the Domain Theory that we have shown belongs to the Intrinsics of air traffic.

   (a) Support Technology:
   Support Technology for air traffic has not been described. Examples of support technologies could be: *(i) air traffic control centres, (ii) air traffic approach (and departure) towers, (iii) air traffic tarmac control,* etc. The first(i) may cover the monitoring (but not "control"[49]) of flights between air traffic towers. The second (ii) monitors & "controls" approaching traffic, i.e. traffic within TMA's (domes) up to the point of landing, and possibly departing traffic as from the point of lift-off. The third (iii) monitors & "controls" aircraft taxiing along runways, taxiways and on the airport tarmac (apron). A description of such supporting technology usually would describe the monitoring and "control", including interaction (communication) protocols between aircrafts and ground monitoring and "control" staff. Included in the description would be cases of erroneous staff handling, failure of the supporting technology (radar, sonar, GPS, radio & video communications, etc.).

   (b) Rules & Regulations:
   Another area which has not been described either are the Rules & Regulations determining for example *scheduling, rescheduling, ground staff to/from aircraft staff protocols,* etc. — the protociols as stipulated by Civil Aviation Authorities (FAA, CAAC, etc.).

2. Requirements:

   Requirements may now proceed, after a suitably "complete" Domain Theory of air traffic has been obtained.

   (a) A General Air Traffic Control Problem:
   Let us assume that the requirements are for some form of air traffic control.

[49]We double quote the term 'control' since "control" can at most give directions to the aircraft, i.e. the pilot, but not really actuate any of its controls!

This implies that the problem belongs to the Control Frame.

Three "extreme" possibilities exists: whither centralised (i.e. ground-based) or de-centralised (i.e. aircraft based) air traffic control.

i. *Centralised Monitoring & "Control":*
   By centralised air traffic control is meant the kind outlined under the item of supporting technology in item 1a on the preceding page.
   This case may either be forced upon us from the very beginning, i.e. tacitly assumed throughout (by no choices being alluded to in any contract!) — in which case we will have included models of that support technology in our Domain Theory (as was made provision for above); or it may be mandated, by the client, as part of the requirements, in which case what was above described as part of the technology support of the domain will instead be described as part of the requirements.
   One Man's Domain is another Man's Requirement!

ii. *De-Centralised Monitoring & "Control":*
   By de-centralised air traffic control we mean a, perhaps "futuristic", situation, in which at least the air traffic control centres and the air traffic control towers are abolished (or at least not to be used by certain aircrafts). Instead they are replaced by "copies" of all necessary functionalities within the cockpit of relevant aircrafts!
   This case may likewise have been either forced upon us from the very beginning, or mandated as part of requirements.
   Now the requirements will describe that new support technology!

iii. "Open Monitoring & Control":
   This case arises when clients do not (wish to) specify whither one or the other technology — but leaves it to implementation!
   Therefore requirements will not mention the form of air traffic control but only which *functions are to be monitored and "controlled".*

These, the *functions* that *are to be monitored and "controlled"* are also to be requirements captured and defined in the centralised and de-centralised cases — as may be many other facilities.

Thus requirements capture & definition proceeds by checking off the usual collection of control frame problem issues: the image of the air-space, timetable, traffic and schedule states to be represented inside the computer; input & updates of state components; etc.

(b) A TCAT Control Problem:

We wish here to make some comments on the much published work, in the US, on TCAT-II.

If our requirements had been less ambitious than sketched above, for example just for the computerized aspects of an aircraft collision avoidance air traffic sub-system control, like TCAT-II, then we would basically have proceeded as in this special appendix section! Before going into the "nitty-gritty" details of the non-software aspects of the TCAT-II supporting technology, we would have first described as much of the domain as we think reasonable — and that is a lot more than is even hinted at in TCAT-II papers! Then we would have proceeded in a stepwise fashion to unfold (i.e. refine) the complexities of the supporting technology, etc. That is: we are far from convinced that the approach currently published as to the technical handling of the TCAT-II problem is up to todays professional standards of software engineering.

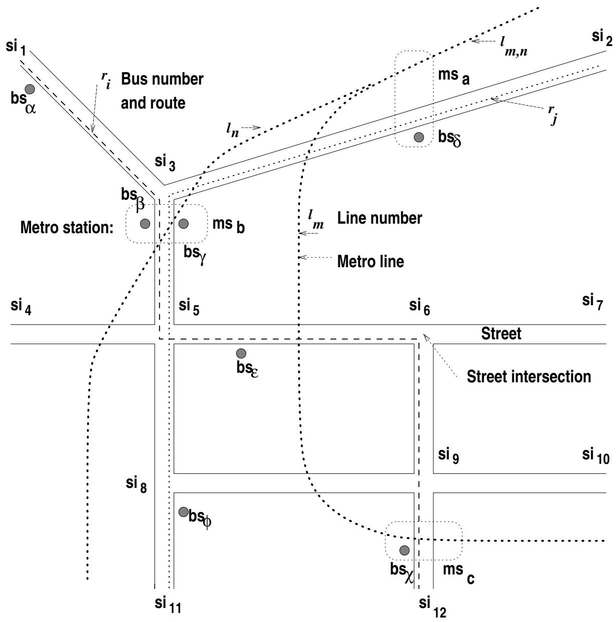
(c) Other Requirements:

Other requirements can be expressed on the basis of the current Domain Theory: requirements for time-tabling software (with the problem becoming a combined Information and Workpiece Multi-Frame problem, etc.

3. Computing Systems Architecture:

Many issues now follow, but all we want to say here is that if requirements were for an "open monitoring & control", then a first issue of computing systems architecture would be to suggest, to the client, one or more information technology solutions to air traffic control, and specify the chosen ones as part of the computing systems, including software architecture.

Figure 20: A Fragment Metropolitan Traffic System Snapshot



Fragment of a Metropolitan Transport System

E.3 MiTraS: Minsk Metropolitan Transport System

MiTraS[50] is an acronym for Minsk Metropolitan Transport System.

We refer to the introductory sections 1.1.1 on page 2 — 1.1.2 on page 2, as well as to item 3 on page 17 of section 3.2.2.

In MiTraS we deal with a combination of streets, busses, metro (underground, subway, regional) trains, and taxis.

With requirements the MiTraS problem comes to represent a Control + Information+ Workpiece + Transaction Clearing Multi-Frame problem. See special appendix section section C.4.4 on page 70.

E.3.1 Towards A Domain Theory

A picture is worth a thousand words! See figure 20.

Some of the *thousand words* could be:

[50] This section is written by Dines Bjørner and Gueorgui Satchok (UNU/IIST Fellow, Minsk, Belarus).

The overall system (*MiTraS*) can be understood as consisting of static street, bus, metro, and taxi subsystems, and a dynamic sub-system which reflects the temporal state of the static subsystems.

The "statics" of streets (*StrSys*) can be characterised by named (*SNm*) (hence multiple, "parallel") fragments between neighbouring street intersections (*Si*) with each fragment (*StFrag*) having a street name (*SNm*), a length (*FraLen*), etc.[51]

The "statics" of the bus system can be characterised by the set of currently available busses (*BId*-**set**), its bus stops (*BsStps*), the bus routes, (*BusRts*), and the bus frequency or time table schedules (*BusSch*). Bus stops can be characterised by their offset position within named street fragments; bus routes by the bus line number and the sequence of bus stops potentially visited.

The "statics" of the train system (*MetSys*) can be characterised by the set of currently available trains (*MId*-**set**), the set of metro stations (*Ms*-**set**), the metro lines (*MeLins*), and the line schedules (*MeSch*). Metro, like bus, lines can be characterised by the sequence of stations stopped at! Metro, like bus, schedules are left further undefined.

The "statics" of the system further relates (*BMCorr*) those bus stops that are closest to metro stations, i.e. for which transfers may take place.

The "statics" of the taxi system (*TaxSys*) can be characterised by the set of taxis currently in use (available or free) (*TId*-**set**), and by the taxi rank (*Tr*) locations wrt. street fragment position.

The dynamics of the system can be characterised by the time-wise position of actually deployed busses, trains and taxis. Bus and taxi positions are either at bus stops — in which case we interpret this as the bus being stopped for passenger alighting or getting off, respectively taxi ranks — in which case we interpret that as the taxi being free. Or bus and taxi positions can be (point) offsets along street fragments, or in street intersections. In either case we interpret this to mean that the busses and taxis are moving (or waiting for traffic to clear up, etc.). A bus at a taxi rank is to be understood as the bus passing the taxi rank and, maybe observing its state. A taxi at a bus stop is to be understood as the taxi passing the bus stop and, maybe, observing its status. Included in the bus stop or taxi rank status is the number of people waiting to board any bus for that stop, respectively one or more taxis.

Thus the "statics" is not quite only statics! Thus we expect to be able to observe for any bus (*BId*), any metro train (*MId*), and for any taxi (*TId*), their states: velocity, acceleration, mass, orientation, power reserves, number of passengers, etc.! Metro stations (*Ms*) may likewise have a number of observable dynamic properties: passenger ticket sales, by time units, for specific trains, etc.

Why do we expect this? Simply because we are speaking of the Domain — for which everything

[51]In this "etc." much is "hidden": a host of desirable topological (cartographic, cadestral, geotechnical, etc.) properties can be "swept" into the three dots: ...!

is possible!

A formalisation is worth an "infiniticity" of pictures!

type

$$
\begin{array}{l}
\text{Si, SNm, FraLen, Bs, Offset, Bn, Ms, Mn, TId, BId, MId} \\
\text{BusSch, MetSch} \\
\text{MiTraS} = \text{Static} \times \Sigma \\
\\
\text{Static} :: \text{rds:StrSys}\ \ \text{bus:BusSys}\ \ \text{met:MetSys}\ \ \text{map:BMCorr}\ \ \text{tax:TaxSys} \\
\text{StrSys} = \text{Si} \ \overrightarrow{m}\ (\text{Si} \ \overrightarrow{m}\ (\text{SNm} \ \overrightarrow{m}\ \text{StFrag})) \\
\text{StFrag} = \text{FraLen} \times \ldots \\
\text{BusSys} = \text{BId-}\mathbf{set} \times \text{BsStps} \times \text{BusRts} \times \text{BusSch} \\
\text{BsStps} = \text{Bs} \ \overrightarrow{m}\ \text{Pt} \\
\text{Pt} \quad = \text{SNm} \times \text{Si} \times \text{Si} \times \text{Len} \\
\text{BusRts} = \text{Bn} \ \overrightarrow{m}\ \text{Bs}^* \\
\text{MetSys} :: \text{trns:MId-}\mathbf{set}\ \ \text{ms:Ms-}\mathbf{set}\ \ \text{mls:MeLins}\ \ \text{msch:MeSch} \\
\text{MeLins} = \text{Mn} \ \overrightarrow{m}\ \text{Ms}^* \\
\text{BMCorr} = \text{Bs} \ \overrightarrow{m}\ \text{Ms} \\
\text{TaxSys} = \text{TId-}\mathbf{set} \times \text{TxRnks} \\
\text{TxRnks} = \text{Tr} \ \overrightarrow{m}\ \text{Pt} \\
\\
\Sigma \quad = \text{T} \rightarrow (\text{Static} \times (\text{Bus}\Sigma \times \text{Metro}\Sigma \times \text{Taxi}\Sigma)) \\
\text{Bus}\Sigma \quad = \text{BId} \ \overrightarrow{m}\ \text{BTPos} \\
\text{BTPos} = \ \text{Bs} \mid \text{Tr} \mid \text{Pt} \mid \text{Si} \\
\text{Metro}\Sigma \ = \text{MId} \ \overrightarrow{m}\ (\text{Ms} \times \text{Offset} \times \text{Ms}) \\
\text{Taxi}\Sigma \ = \text{TId} \ \overrightarrow{m}\ (\text{Tr} \mid \text{BTPos})
\end{array}
$$

A Domain Theory can now be further elaborated. It could characterise the flow of passengers within and across the various sub-systems, adherence to schedules, co-ordination of schedules and of real traffic, etc.

E.3.2 Towards Requirements

Example alternative requirements to computerised support of complementary facets of the metropolitan transport can now be established. We exemplify some which relate to co-ordination among the two or the three vehicle types: busses, trains and taxis.

1. MiTraS/RaLNaS^2 — *A 'Route & Line' 'Net & Schedule' design, planning & development System:*

Synopsis :

Based on past statistics and future (polled) Traffic Flow estimates MiTraS/RaLNaS[2] is to provide computerised support for the design, planning and the management of the development ([de-]construction) of MiTraS services: planning new bus or metro lines and route schedules (or the removal of existing ones), effecting such changes (i.e. the smooth phasing in, or out, of lines or routes), etc.

Narrative :

- Traffic Flow:
  By a traffic flow (*Flow*) we may understand a collection of data which describes, for adjacent, non-overlapping time intervals, how many people are moving between suitable origins and destinations. By origins and destinations we mean (street related) points.
- Line & Route Design:
  By line & route design we mean a function, larp, which given the present bus & metro net decides on which, if any, existing metro lines and bus routes to retain, which, if any, to put out of service (terminate), and which new ones, if any, to add.
  Line & route design is to be based on the following Metro Line & Bus Route Planning Rules (*OPR*) — and here follows a detailed description of these rules!
  The outcome of this kind of design is, (*LRP*), a set, lns, of the lines and routes to be terminated and a set of the new lines, (*NewMet*), and routes, (*NewBus*). The latter may involve new metro stations [or bus stops] either along existing stretches between existing stations (exi [, resp. str]), or may, for metros, imply the construction of new rails (new)! These rails (*Rails*) may be described topologically. (But we refrain!)
  The design outcome may finally outline required resources (*Res*) (monies, time, labour, equipment, materials, etc.) needed to actually implement the design.
- Planning:
  By planning we mean a function (plan) which given the design results and given an outline (*Outlin*) of available and estimations of avalailability and quantities of additional resources, derives a resource allocation and schedule (i.e. time interval *TI* related) plan (*RASP*) for (the located pairlL) deployment of resources (*R*). These resources relate to specific metro stations, new rails, bus stops, etc. This relation (*Task*) is also given in the plan.
- Development:
  By development (devt) we mean the actual deployment, over time, to the implementation of the plan.
- *&c.:* By 'etcetera' we mean further, extensive narratives.

Models :

type [design]
  TI = {| (f,t) | (f,t):T×T • f<t |}
  Flow = TI $\overrightarrow{m}$ (Or $\overrightarrow{m}$ (De $\overrightarrow{m}$ **Nat**))
  Or, De = Pt

$$
\begin{array}{l}
\quad \mathrm{LRP} \;\; = (\mathrm{Mn}|\mathrm{Bn})\textbf{-set} \times \mathrm{NewMet} \times \mathrm{NewBus} \times \mathrm{Res} \times \mathrm{Plan} \\
\quad \mathrm{NewMet} :: \mathrm{lns}{:}(\mathrm{Mn} \xrightarrow[m]{} \mathrm{Ms}^*) \\
\qquad\qquad\qquad \mathrm{exi}{:}(\mathrm{Ms} \xrightarrow[m]{} (\mathrm{Ms} \times \mathrm{Offset} \times \mathrm{Ms})) \\
\qquad\qquad\qquad \mathrm{new}{:}((\mathrm{Ms}{\times}\mathrm{Ms}) \xrightarrow[m]{} \mathrm{Rail}) \\
\quad \mathrm{NewBus} :: \mathrm{lns}{:}(\mathrm{Bn} \xrightarrow[m]{} \mathrm{Bs}^*) \\
\qquad\qquad\qquad \mathrm{str}{:}\mathrm{BsStps} \\
\quad \mathrm{Rail},\ \mathrm{Res} \\
\textbf{value} \\
\quad \mathrm{larp}{:}\ \mathrm{Static} \times \mathrm{Flow} \stackrel{\sim}{\to} \mathrm{LRP} \\
\\
\textbf{type}\ [\,\mathrm{planning}\,] \\
\quad \mathrm{R},\ \mathrm{L},\ \mathrm{Outlin} \\
\quad \mathrm{RASP} = (\mathrm{TI} \xrightarrow[m]{} (\mathrm{R} \xrightarrow[m]{} \mathrm{L})) \times \mathrm{Task} \\
\quad \mathrm{Task} = (\mathrm{Ms}|\mathrm{Bs}|\mathrm{Rail}) \xrightarrow[m]{} \mathrm{R} \\
\textbf{value} \\
\quad \mathrm{plan}{:}\ \mathrm{LRP} \times \mathrm{OutLin} \stackrel{\sim}{\to} \mathrm{RASP} \\
\\
\textbf{value}\ [\,\mathrm{development}\,] \\
\quad \mathrm{devt}{:}\ \mathrm{RASP} \stackrel{\sim}{\to} (\mathrm{T} \to (\mathrm{Static} \times \mathrm{RASP}))
\end{array}
$$

The design and planning parts of MiTraS/RaLNaS[2] are of a typical Information & Workpiece Multi-Frame problem, whereas the development part has additional facets of a Control Frame problem.

2. MiTraS/BaTCoS — *A Bus⊕Taxi Co-ordination System:*

Synopsis :

As busses pass taxi ranks bus drivers can report to a co-ordinating agent rough estimates of waiting, potential taxi passengers; and as taxis pass bus stops they can likewise report on waiting, potential bus passengers. Such reporting may lead to insertion of additional busses and taxis, and to re-routing of available taxis. Finally bus passengers may be able to order taxis to meet them at specified bus stops.

Narrative : *&c.!*

Model : *&c.!*

The bus and taxi coordination system may also include considerations of satisfying the dynamics of passenger flows with a set of on-route vehicles. Unpredictable events may cause temporary deviations from schedules and require rescheduling.

MiTraS/BaTCoS primarily represents a "Control" & Clearing Transaction Multi-Frame problem.

3. MiTraS/CITaRS — *A Commuter Inquiry & Ticketing & Reservation System:*

Synopsis :

Commuters, i.e. potential passengers, whether for bus, metro or taxi transport , wish to inquire as to how cheapest, or fastest, or with fewest transfers, etc., to commute from one point to another; to obtain prices (or estimates, if involving taxis) for contemplated (i.e. such) journies; or to even "book" these journies (including reserving specific departures or ordering taxis at specific points).

Narrative : *&c.!*

Model : *&c.!*

MiTraS/CITaRS primarily represents an Information & Clearing Transaction Multi-Frame problem.

4. *&c.*

E.3.3 Towards An Information Technology

The alternative requirements: MiTraS/BaTCoS and MiTraS/CITaRS, gave implementation a freedom to decide on support technology. Again various kinds can be envisaged. Support technologies differ with respect to. to either of:

1. Centralised Monitoring:

   A centralised transaction clearing centre receives, processes and dispatches messages concerning:

   - MiTraS/BaTCoS: bus stop states, taxi rank states, taxi reservations, etc.
   - MiTraS/CITaRS: inquiries & reservations

2. De–centralised Monitoring:

   A distributed set of loosely communicating transaction clearing centres receives, processes and dispatches messages concerning:

   - MiTraS/BaTCoS: bus stop states, taxi rank states, taxi reservations, etc.
   - MiTraS/CITaRS: inquiries & reservations

3. *&c.*

The support technologies may range from street sensors (& actuators) and traffic television cameras & monitors, via radio-telephone, to GPS (global positioning systems), etc.

E.3.4 Discussion

The "static" and dynamic state model that we presented was rather concrete. More abstract ones can be given. Since the problem(s) to be solved had to do with co-ordination among the various transport sub-systems, rather than the monitoring & control of the individual sub-systems, their models need not have followed the form of the railway system model presented in special appendix section E.1 on page 103. Or, perhaps we should have modelled both the street and the rail network in terms of respective units? Many good reasons can be given for doing exactly that.

We choose to present the present rather "mechanistic" model just for the sake of letting the three specific transport examples (viz.: RaCoSy, ATC'2000, and MiTraS: special appendix sections E.1–E.3) be modelled in different ways!

E.4 General Transportation Systems

The previous sections (E.1–E.3) covered different types of transport systems: railways, airline and general metropolitan (busses, metro, taxis). Common to them were notions such as traffic (i.e. movement), time tables, schedules, and transport nets (whether rails, air-corridors, or streets).

E.4.1 A Graph Model of Transport

The[52] three models of special appendix sections E.1 (starting page 103) — E.3 (starting page 115) (railways, air traffic and metropolitan transport) can be summarised in one, common model. In this model to be briefly shown in this section one and the same parameterisable "base" model can be instantiated to either of the systems shown in special appendix sections E.1 (starting page 103) — E.2 (starting page 108), or to combinations of such, as shown for land-based transports in special appendix section E.3 (starting page 115).

L: Space

L stands for spatial locations. The point to be made here is that we do not model L. m stands for a **metric**, for example a function which to pairs of locations give their distance D.

type
  L, D
value
  m: L$\times$L $\to$ D

This model of space shall be seen as fully 'orthogonal' to, i.e. independent of our subsequent model of transport nets as 'graphs'.

G: The Graph of Streets, Rail Lines, Routes & Air-lanes

The networks (nets) of streets, rail lines, canal & river routes, and air-lanes, is modelled as graphs of ways and junctions (edges and nodes, respectively). At this level the network graphs are considered un-directed.

[52] This section is written Dines Bjørner and is based on material researched by Chris George, UNU/IIST, and by Dr. Dr. Gueorgui Satchok, Center of IT, Belarussian State University of Informatics and Radio-electronics, Minsk, Belarus

Basically we have: W stands for a set of ways along which transport can take place (linear units in the RaCoSy model, air lanes in the ATC'2000 model, and, for example, street segments in the MiTraS model). J stands for the junctions at which ways "meet" [are joined] (non-linear units in the RaCoSy model [i.e. switches, cross-overs, etc.], airports in the ATC'2000 model, street intersections and metro stations in the MiTraSmodel). We model graphs abstractly, as a sort, and presently introduce observers of graph nodes (junctions) and edges (ways), and (given an edge) we postulate that we can also observe its zero, one or two junctions. Zero if the edge is not part of a graph, one if the edge is looping, and two if the edge "connects" two distinct junctions — as would be normal.

type
  G, W, J
value
  obs_G_Ws: G $\rightarrow$ W-**set**
  obs_G_Js: G $\rightarrow$ J-**set**
  obs_W_Js: W $\rightarrow$ J-**set**
axiom
  $\forall$ g:G, w:W $\bullet$ w $\in$ obs_G_Ws(g) $\Rightarrow$
    obs_W_Js(w) $\subseteq$ obs_Js(g) $\wedge$ **card** obs_W_Js(w) $\leq$ 2

R: Routes

Given a graph we can speak about all finite length routes (R) (i.e. sequences of ways) through the un-directed graph, of cyclic routes, etc. (We could define similar notions for junction routes.)

type
  R = W*
value
  paths: G $\rightarrow$ R-**set**
  is_cyclic_R: R $\rightarrow$ **Bool**
  is_G_R: G $\times$ R $\rightarrow$ **Bool**
  is_G_R(g,r) $\equiv$ r $\in$ paths(g)

M: Graph to Space Maps

Junctions can be mapped onto locations, and ways onto discrete or dense ('continuous') sets of locations:

value
  map_J_L: J $\rightarrow$ L
  map_W_Ls> W $\rightarrow$ L-**infset**

P: Paths

Paths (P) designate means of getting through a junction from one way to another. You may think of a path as a pair ($W \times W$) of ways. A state (Σ) of a junction is a possibly empty, finite set of paths and can be interpreted as those directions (from one way to another, connected way) which are open (for movement) at a given time. The set of all possible junction states is referred to as Ω.

type
  $P' = W \times W$
  $P \ = \{| \ (w,w') \mid (w,w'){:}P' \bullet$ **card**(obs_WJs(w) isect obs_WHs(w')) = 1 $|\}$
  Σ = P-**set**
  Ω = Σ-**set**
value
  obs_J_Σ: J $\rightarrow$ Σ
  obs_J_Ω: J $\rightarrow$ Ω
axiom
  $\forall$ j:J, σ:Σ
    • obs_J_Σ(j) $\in$ obs_J_Ω(j)

TS: Transport Stops

Transport stops TS designate places where transport vehicles may make scheduled stops to allow passengers to enter or alight. Transport stops are points along [directed] ways — which, as locations are offset (some distance) along the way between two junctions.

type
  TS
value
  obs_G_TSs: G $\tilde{\rightarrow}$ TS-**set**
  obs_W_TSs: W $\tilde{\rightarrow}$ TS-**set**
  obs_TS_W: TS $\tilde{\rightarrow}$ W
  obs_TS_L: TS $\tilde{\rightarrow}$ L
  obs_TS_Pos: TS $\tilde{\rightarrow}$ $W \times J \times D \times J$

axiom

$\forall$ g:G $\bullet$
  $\forall$ j,j′:J $\bullet$ j,j′ $\in$ obs_G_Js(g) $\Rightarrow$
    $\exists$ w:W $\bullet$ w $\in$ obs_G_Ws(n) $\wedge$ {j,j′} = obs_W_Js(w) $\wedge$
  $\forall$ ts:TS $\bullet$ ts $\in$ obs_G_TS(n) $\Rightarrow$
    $\exists$ w:W $\bullet$ w $\in$ obs_G_Ws(n) $\wedge$ w = obs_TS_W(ts),
$\forall$ ts:TS $\bullet$ $\exists$ w:W, j,j′:J, d:D $\bullet$
  let (w,j,d,j′) = obs_TS_Pos(ts) **in**
  obs_TS_W(ts) = w $\wedge$ ts $\in$ obs_W_TSs(w) $\wedge$
  {j,j} = obs_W_Js(w) $\wedge$ d = m(l,l′) **end**

Interpretations

- Way Interpretations:

  A way between two junctions is undirected. Had we modelled:

  value
    obs_W_Js: W $\rightarrow$ J$\times$J
    obs_W_Js(w) **as** (j,j′) ...

  then we might be said to model a way directionally: from j to j'. Now the above 'oracle' function yields either of the sets: $\{\}$, $\{j\}$, or $\{j, j'\}$ – where $j, j' : J, j \neq j'$. All we can say is that w connects j and j' — in either direction. Whether transport may actually be allowed to move in both, some or any direction depends could be modelled through the states of the adjoining junctions! That is, if:

  $\exists$ g:G, j,j′:J, w,w′,w″:W $\bullet$
    {w,w′,w″} $\subseteq$ obs_G_Ws(g) $\wedge$ {j,j′} = obs_W_Js(w)
      $\Rightarrow$ (w′,w) $\notin$ $\bigcup$ obs_Ω(j)
      $\wedge$ (w,w″) $\notin$ $\bigcup$ obs_Ω(j′)

  then w is a [perhaps temporarily] closed way! But in what sense is it closed. Could it not be that it is only closed to enter from the outside? Yes indeed! To make sure the above also models inability to leave the way to the outside we must add:

      $\wedge$ (w,w′) $\notin$ $\bigcup$ obs_Ω(j)
      $\wedge$ (w″,w) $\notin$ $\bigcup$ obs_Ω(j′)

  And still we cannot, through this kind of modelling, express that no movement is allowed within certain ways.

- Air Space Interpretations:

  In the above interpretation you may substitute ways for streets or airways.

  Now: what are airports? To arrive at a model let us observe that one part of the general model of transport nets models the airspace notions of airways while another part models the ground-based runways (for take-off or landing), the taxiing ways and the tarmac (apron), with junctions between the air and runways, between runways and taxiing ways, and between taxiing ways and the tarmac lanes. Gates are being modelled as junctions.

- Railway Interpretations:

  Finally, as previously mentioned, certain junctions model switches, other junctions model simple, and yet other switched cross-overs. Ways are thus used to only model linear units.

N: Transport Nets

A network, as defined above, may consist, and usually does, of a number of "isolated" sub--networks. These are themselves "full-blown", but 'singular' networks. A singular network consists of connected ways and junctions. That is: there are means of traversing the network, in or "against" the opened direction of ways, from any junction to any other. A 'multiple' network consists of two or more 'singular' networks where no two 'singular' (or, as we shall henceforth call them, sub-) networks share any junctions.

type
  N
value
  obs_N_Gs: N $\rightarrow$ G-**set**

- Interpretation:

  The interpretation of sub-networks is that they each serve one distinct class of vehicle types.

  One sub-network may serve mini, local, shuttle, express and regional busses. Another sub--network may serve shuttle, metro, sub-urban, and regional trains. Yet another network may serve cog-wheel transport. Others may serve air traffic, or river, canal, and coastal (etc.) ship (boat, etc.) transport. *&c.*

Let us assume that a network $n : N$ consists of several distinct sub-networks: $n_1, n_2, \ldots, n_m : N$ networks. Now the geography of a network includes information (NX) as to which transport stops of one sub-network provides "easy, convenient" (i.e. ticket "authorised") transfer to transport stops of another network. In general we can "convert" a general network N into a named set

(MN) of singular sub-networks SN, and for each pairing of these we can indicate such transfer stops.

From primarily presenting an abstract types (sorts) model with observers we continue, now "switching" to concrete, model-oriented types. If you do not like the "bias" which may thus be introduced we can always "convert" the below concrete types into appropriate pairs of sorts and observers.

$$
\begin{array}{l}
\textbf{type } \mathrm{NNm} \\
\quad \mathrm{SN} = \{|\ \mathrm{g} \mid \mathrm{g{:}G} \bullet \mathrm{is\_singular\_G(g)}\ |\} \\
\quad \mathrm{MN}' = \mathrm{NNm} \ \overrightarrow{m} \ \mathrm{SN} \\
\quad \mathrm{MN} = \{|\ \mathrm{mn} \mid \mathrm{mn{:}MN}' \bullet \mathrm{are\_disjoint(mn)}\ |\} \\
\quad \mathrm{NX} = \mathrm{NNm} \ \overrightarrow{m} \ (\mathrm{TS} \ \overrightarrow{m} \ (\mathrm{NNm} \ \overrightarrow{m} \ \mathrm{TS}\text{-}\textbf{set})) \\
\textbf{value} \\
\quad \mathrm{is\_singular\_G{:}\ G} \to \textbf{Bool} \\
\quad \mathrm{are\_disjoint{:}\ MN} \to \textbf{Bool} \\
\quad \mathrm{are\_disjoint(mn)} \equiv \\
\quad\quad \forall\ \mathrm{nm,nm'{:}NNm} \bullet \{\mathrm{nm,nm'}\} \subseteq \textbf{dom}\ \mathrm{mn} \wedge \mathrm{nm} \neq \mathrm{nm'} \\
\quad\quad\quad\quad\quad\quad\quad \Rightarrow \mathrm{mn(nm)} \neq \mathrm{mn(nm')} \\
\quad\quad \wedge\ \textbf{let}\ \mathrm{gs} = \textbf{rng}\ \mathrm{mn}\ \textbf{in} \\
\quad\quad\quad \sim\exists\ \mathrm{g,g'{:}G} \bullet \mathrm{g} \in \mathrm{gs} \wedge \mathrm{g'} \in \mathrm{gs} \wedge \mathrm{g} \neq \mathrm{g'} \Rightarrow \\
\quad\quad\quad\quad \mathrm{obs\_G\_Js(g)\ isect\ obs\_G\_Js(g')} = \{\} \wedge \\
\quad\quad\quad\quad \mathrm{obs\_G\_Ws(g)\ isect\ obs\_G\_Ws(g')} = \{\}\ \textbf{end} \\
\quad \mathrm{obs\_N\_SN{:}\ N} \to \mathrm{SN}\text{-}\textbf{set} \\
\quad \mathrm{obs\_N\_SN\_NNm{:}\ N} \times \mathrm{SN} \to \mathrm{NNm} \\
\quad \mathrm{obs\_N\_NX{:}\ N} \to \mathrm{NX}
\end{array}
$$

Γ: Network Dynamics

Over time the junctions of a network change current as well as feasible, state; ways incident upon a junction may also change; new ways and new junctions may be added to the network, and old removed from the network — all in response to network development or the breakdown (hence temporary or permanent removal of ways). The breakdown of a junction is recorded as the state of that junction remaining closed (no through paths) for a longer period. A dynamic network is a continuous function from time into networks such that at any two infinitesimal close times the corresponding network differ only by such changes that guarantee monotonic network changes.

$$
\begin{array}{l}
\textbf{type} \\
\quad \Gamma = \{|\ \mathrm{tn} \mid \mathrm{tn{:}(T} \to \mathrm{N)} \bullet \mathrm{wf\_}\Gamma\mathrm{(tn)}\ |\} \\
\quad \Delta = \mathrm{T} \to (\mathrm{NNm} \ \overrightarrow{m} \ \mathrm{SN})
\end{array}
$$

where Δ suggests a model for the separate sub-dynamics of singular networks.

Ψ: Transport Routes and Vehicle Space & Dynamics

Named transport routes *TRS* indicate route names, well-formed sequences of scheduled stops, and the type of the vehicles that can serve the routes. A sequence is well-formed if there exists singular net with a sequence of connected ways and junctions such that the route stops can be found along that sequence.

A given transport sub-system, in addition to the above, consists of a set of individually named vehicles (*VNm*) each (temporarily) assigned to a route.

Finally, at any moment in time, a number of vehicles are on or off some route (whether allocated or not, whether of the type assigned to the route, etc.). Indeed, some may even be off transport, even just ways!

$$
\begin{array}{l}
\textbf{type}\ \mathrm{RNm},\ \mathrm{VNm} \\
\quad \mathrm{TRS}' = \mathrm{RNm} \ \overrightarrow{_{m}}\ (\mathrm{TS}^* \times \mathrm{VTy}) \\
\quad \mathrm{TRS}\ \ = \{|\ \mathrm{trs} \mid \mathrm{trs{:}TRS}' \bullet \mathrm{wf\_TRS(trs)}\ |\} \\
\quad \mathrm{VS}\ \ \ = \mathrm{VNm} \ \overrightarrow{_{m}}\ \mathrm{RNm} \\
\quad \mathrm{AVS} \subset \mathrm{VNm} \\
\quad \Psi' = \ \mathrm{T} \rightarrow ((\mathrm{N} \times (\mathrm{VNm} \ \overrightarrow{_{m}}\ \mathrm{L})) \times \mathrm{AVS}) \\
\quad \Psi\ \ = \{|\ \psi \mid \psi{:}\Psi' \bullet \mathrm{wf\_}\Psi(\psi)\ |\} \\
\textbf{value} \\
\quad \mathrm{obs\_VTy{:}}\ \mathrm{VNm} \rightarrow \mathrm{VTy}
\end{array}
$$

Observe that transport routes and vehicle allocations are not part of the vehicle dynamics state Ψ. (The vehicles that move need not be among those allocated.)

RTT: Route Timetables

For every route there is a timetable. The timetable abstracts the arrival and departure times at traffic stops along the route.

$$
\begin{array}{l}
\textbf{type} \\
\quad \mathrm{RTT} = \mathrm{RNm} \ \overrightarrow{_{m}}\ (\mathrm{TS} \ \overrightarrow{_{m}}\ (\mathrm{T}{\times}\mathrm{T}))
\end{array}
$$

Given a network, given a reservoir of appropriate vehicles, and a time table, one can now define the possibly infinite set of transport dynamics which satisfy the timetable:

value
schedule: $N \times VS \times RTT \overset{\sim}{\to}$ **Ψ-infset**

PFTT: Passenger Flow Timetables

A number of views can be applied to a concept of passenger flow:

- Real Flow:

  An ideal observer could record flow of people between any pair of locations. If ℓ, ℓ' are two locations and at a given time t exactly q people are on their way from origin ℓ to destination ℓ', then the recording would say so! One could also group these idealised observations by time interval $\tau\iota$ with the meaning that q people decided during that time interval to start from origin ℓ heading for destination ℓ'.

type
$\mathrm{TI} = \{| \ (b,e) \mid (b,e){:}T \times T \bullet b \leq e \ |\}$
$\mathrm{PFTT} = \mathrm{TI} \to ((L \times L) \ \underset{m}{\to} \ \mathbf{Nat})$

- Polled or Estimated Flow Statistics:

  Another interpretation, for example use, of the above is for prediction. Now the records show statistical estimates of expected flow of people willing to be transported between traffic stops close travel origins and destinations.

- Accommodated Flows:

  A final interpretation as follows: given a particular net, given a particular reservoir of vehicles of appropriate types, and a timetable, one can describe the (most likely infinite) set of flows that can be accommodated by the system.

Etcetera!

We decide to end our parameterisable model of transport here. In fact: we have only shown some of the important notions of such a model and some of the "derivatives", i.e. "excursions" that can be made from such models. A next step is to suitably "package" the model into a lattice of — in our case — RSL classes etc.

E.4.2 A Stepwise Unfolded — Can/Cannot Do Transport Model

Elements of the domain of city traffic are being formalised, driven by different applications of the model. All applications have to do with calculating routes between two locations in the street network: any route, the shortest route, the shortest route obeying traffic regulations, the shortest route obeying traffic regulations and avoiding traffic congestion, ... At each stage we point out possibilities and limitations, the latter motivating future refinements of the model.

Network

Assume locations around a city (street junctions) and connections between such locations (street sections). A network is a non-empty set of connections.

```
type
   Location, Connection
value
   net: Connection-set • net ≠ {},
   start, finish: Connection → Location
```

A route is a sequence of adjacent connections in the network:

```
type
   Route = {| cl:Connection* • iswf(cl)|}
value
   iswf: Connection* → Bool
   iswf(cl) ≡ elems cl ⊆ net ∧
      (∀ i:Nat • i ∈ inds cl ∧ i>1 ⇒ finish(cl(i−1))=start(cl(i)))
```

This model is good for finding all routes between two locations:

```
value
   is_route: Location × Location × Route → Bool
   is_route(l1,l2,r) ≡
      if r=⟨⟩
      then l1=l2
      else let c=hd r in
               start(c)=l1 ∧ is_route(finish(c),l2,tl r)
            end
      end
```

It is not good for finding the shortest route.

Distances

Assume that each connection is assigned a length:

value
  length: Connection $\to$ **Nat**,

Then each route can have its length calculated:

value
  length: Route $\to$ **Nat**
  length(r) $\equiv$ **if** r=$\langle\rangle$ **then** 0 **else** length(**hd** r)+length(**tl** r) **end**

This model is good for finding the shortest route between two locations:

value
  is_shortest_route: Location $\times$ Location $\times$ Route $\to$ **Bool**
  is_shortest_route(l1,l2,r) $\equiv$
    local
      value
        is_shortest: Location $\times$ Location $\times$ Route $\to$ **Bool**
        is_shortest(l1,l2,r) $\equiv$
          ($\forall$ r$'$:Route $\bullet$ is_route(l1,l2,r$'$) $\Rightarrow$ length(r$'$)$\geq$length(r))
    in
      is_route(l1,l2,r) $\wedge$ is_shortest(l1,l2,r)
    end

It is not good to find the shortest route that respects traffic regulations.

Regulations

Assume vehicles in the network, their movement subject to the restrictions on where to enter the traffic and what turns to take on junctions:

```
type
   Vehicle
value
   entry: Vehicle → Connection-set
   entry(v) as cs post cs ⊆ net,

   turn: Vehicle × Connection ~→ Connection-set
   turn(v,c1) as cs
      post (∀ c2:Connection •
         c2 ∈ cs ⇒
         c2 ∈ net ∩ entry(v) ∧ start(c2)=finish(c1))
      pre c1 ∈ net ∩ entry(v)
```

This model is good to find the shortest regular route:

```
value
   is_shortest_regular_route: Vehicle × Location × Location × Route → Bool
   is_shortest_regular_route(v,l1,l2,r) ≡
      local
         value
            is_regular: Vehicle × Route → Bool
            is_regular(v,r) ≡
               case r of
                  ⟨⟩ → true,
                  ⟨c⟩ → c ∈ entry(v),
                  ⟨c⟩^s → hd s ∈ turn(v,c) ∧ is_regular(v,tl r)
               end
      in is_shortest_route(l1,l2,r) ∧ is_regular(v,r) end
```

This model is not good to avoid congested routes.

Congestion

Assume that vehicles in traffic are positioned at connections while moving between two adjacent locations, and define congestion as a maximal number of vehicles that could be present at a given connection:

```
type
   Traffic = {|t:Connection →m Vehicle-set • dom t=net|}
value
   congestion: Connection → Nat
```

This model is good for finding the shortest, regular, non-congested route:

```
value
  is_shortest_regular_noncongested_route:
    Vehicle × Traffic × Location × Location × Route → Bool
  is_shortest_regular_noncongested_route(v,t,l1,l2,r) ≡
    local
      value
        is_noncongested: Traffic × Route → Bool
        is_noncongested(t,r) ≡
          if r=⟨⟩ then true
          else card t(hd r) < congestion(hd r) ∧
                 is_noncongested(t,tl r) end
    in is_shortest_regular_route(v,l1,l2,r) ∧ is_noncongested(t,r) end
```

It is not good to find the fastest, non-congested regular route, etc.

E.4.3 Towards A Theory of Transport Systems

The[53] model of the previous section is a reflection of a more theory oriented model. It will be briefly surveyed in this section.

We wish, based on abstractions of movement and resources[54], to more abstractly investigate fundamental transportation concepts.

- **Caveat:**

  This section is at present, August 21, 1997, incomplete. It lacks all the explanatory text.

1. An Evolutionary Model — EM:

   There is an abstract notion of (transport) state — and we do not say what it is. States are only interesting if, however, we postulate a concept of transitions. *transition* is a predicate which applies to a pair of states (i.e. is a binary relation) and expresses whether such a pair of states "offer" a possible transition. *Evolution* defines a possibly infinite set of states ordered by *transition* τ. Some evolutions are not possible, for some reason which we do not explicitly define (*'rho* ρ).

   type
   $\quad \mathrm{EM} = \Sigma \times \tau \times \Theta \times \rho$

   where:

   - Σ is an abstract (state) set
   - τ is a partial order on Σ
   - *Evolution* Θ is the set of all subsets of Σ totally ordered by τ
   - ρ is a property on Θ

   scheme
   $\quad$ EVOLUTIONARY_MODEL =
   $\quad\quad$ **class**
   $\quad\quad\quad$ **type**
   $\quad\quad\quad\quad \Sigma$
   $\quad\quad\quad\quad \Theta = \Sigma$**-infset**
   $\quad\quad\quad$ **value**
   $\quad\quad\quad\quad \tau\text{: } \Sigma \times \Sigma \to \mathbf{Bool}$
   $\quad\quad\quad$ **axiom**

[53] This section is based on material [74] researched by Dr. Nikolai S. Nikitchenko (Department of Theory of Programming, Kyiv [Kiev] Taras Shevchenko University, Ukraine) and is is written by Dines Bjørner.

[54] Resources may be mobile, for vehicles, passengers or freight; or stationary, for rails, air-space, streets, etc.

[Partial Order]
$\forall\ \sigma,\sigma',\sigma'':\Sigma \bullet ((\tau(\sigma,\sigma') \wedge \tau(\sigma',\sigma'')$
$\Rightarrow \tau(\sigma,\sigma'')) \wedge \tau(\sigma,\sigma) \wedge (\tau(\sigma,\sigma') \wedge \tau(\sigma',\sigma)) \Rightarrow \sigma = \sigma')$
type
$\Theta = \{|\ \sigma\mathrm{s}:\Theta \bullet \forall\ \sigma,\sigma':\Sigma \bullet \sigma \in \sigma\mathrm{s} \wedge \sigma' \in \sigma\mathrm{s} \Rightarrow \tau(\sigma,\sigma') \vee \tau(\sigma',\sigma)\ |\}$
end

2. General Transport Models:

(a) Resources:

type
R
value
take: $\mathrm{R} \times \mathrm{R} \times \Sigma \rightarrow$ **Bool**

(b) Fundamental Movement Abstraction:

value
moving: $\mathrm{R} \times \Theta \rightarrow$ **Bool**
axiom
[Moving]
$\forall\ \mathrm{r}:\mathrm{R} \bullet\ \forall\ \theta:\Theta \bullet$
$\mathrm{moving}(\mathrm{r},\theta) \equiv$
$(\exists\ \mathrm{r}':\mathrm{R} \bullet\ \exists\ \sigma,\sigma':\Sigma \bullet \sigma \in \theta \wedge \sigma' \in \theta \wedge \mathrm{take}(\mathrm{r},\mathrm{r}',\sigma) \wedge \sim \mathrm{take}(\mathrm{r},\mathrm{r}',\sigma'))$

(c) General Transport Model (GTM):

i. The Σ State:

type
$\mathrm{GTM} = \Sigma \times \tau \times \Theta \times \rho \times \mathrm{R} \times \mathrm{take}$

ii. The "Types":

scheme
GENERAL_TRANSPORT_MODEL =
extend EVOLUTIONARY_MODEL **with**
class
type
R
value
take:$\mathrm{R} \times \mathrm{R} \times \Sigma \rightarrow$ **Bool**
end

(d) The Main Transport Problem:

value
intention:R $\times$ Θ $\rightarrow$ **Bool**

type
Solution =
$\{|\ \theta{:}\Theta \bullet \sim \rho(\theta) \wedge (\forall\ r{:}R \bullet \text{intention}(r,\theta))\ |\}$

3. Spatial Transport Model:

   (a) Movement Refutation:

   value
   movement_ρ:R $\times$ Θ $\rightarrow$ **Bool**

   axiom
   [Movement_ρ]
   $\forall\ \theta{:}\Theta \bullet \forall\ r{:}R \bullet\ \text{movement\_}\rho(r,\theta) \Rightarrow \rho(\theta)$

   (b) Immovable Resources:

   value
   immovable:R $\rightarrow$ **Bool**
   axiom
   [Immovable]
   $\forall\ r{:}R \bullet\ \forall\ \theta{:}\Theta \bullet \text{immovable}(r) \wedge \text{moving}(r,\theta) \Rightarrow \text{movement\_}\rho(r,\theta)$

   (c) Passive Resources:

   value
   passive:R $\rightarrow$ **Bool**
   axiom
   [Passive]
   $\forall\ r{:}R \bullet \forall\ \theta{:}\Theta \bullet \text{passive}(r) \wedge$
   $(\exists\ \sigma,\sigma'{:}\Sigma \bullet (\sigma \in \theta \wedge \sigma' \in \theta) \wedge$
   $(\exists\ r'{:}R \bullet \text{immovable}(r') \wedge \text{take}(r,r',\sigma) \wedge \sim \text{take}(r,r',\sigma')) \wedge$
   $(\forall\ \sigma''{:}\Sigma \bullet (\tau(\sigma,\sigma'') \wedge \tau(\sigma'',\sigma') \vee \tau(\sigma',\sigma'') \wedge \tau(\sigma'',\sigma)) \Rightarrow$
   $(\forall\ r''{:}R \bullet \text{take}(r,r'',\sigma'') \Rightarrow \text{immovable}(r'')))) \Rightarrow \text{movement\_}\rho(r,\theta)$

   (d) Prohibited Places:

   value
   prohibited_place:R $\times$ R $\rightarrow$ **Bool**
   axiom
   [Prohibited_place]
   $\forall\ r{:}R \bullet \forall\ \theta{:}\Theta \bullet (\exists\ r'{:}R \bullet \exists\ \sigma{:}\Sigma \bullet$
   $\text{prohibited\_place}(r,r') \wedge \sigma \in \theta \wedge \text{take}(r,r',\sigma)) \Rightarrow \text{movement\_}\rho(r,\theta)$

(e) Subroutes & Same Routes:

value
  subroute:R $\times$ Θ $\times$ Θ $\rightarrow$ **Bool**
axiom
  [Subroute]
  $\forall$ r:R • $\forall$ θ,θ':Θ •
    subroute(r,θ,θ') $\equiv$
      ($\exists$ f:$\Sigma \rightarrow \Sigma$ • $\forall$ σ,σ':Σ •
        (($\sigma \in \theta \wedge \sigma' \in \theta$) $\wedge$ (f(σ) $\in \theta' \wedge$ f(σ') $\in \theta'$) $\wedge$
          τ(σ,σ') $\Rightarrow$ τ(f(σ),f(σ'))) $\wedge$ ($\forall$ r$'$:R • take(r,r$'$,σ) $\equiv$ take(r,r$'$,f(σ))))
value
  same_route:R $\times$ Θ $\times$ Θ $\rightarrow$ **Bool**
axiom
  [Same_route]
  $\forall$ r:R • $\forall$ θ,θ':Θ •
    same_route(r,θ,θ') $\equiv$ subroute(r,θ,θ') $\wedge$ subroute(r,θ',θ)

(f) Refuted Routes:

value
  refuted_route:R $\times$ Θ $\rightarrow$ **Bool**
axiom
  [Refuted_route_stab]
  $\forall$ r:R • $\forall$ θ,θ':Θ •
    same_route(r,θ,θ') $\Rightarrow$ (refuted_route(r,θ) $\equiv$ refuted_route(r,θ'))
axiom
  $\forall$ r:R • $\forall$ θ:Θ • refuted_route(r,θ) $\Rightarrow$ movement_ρ(r,θ)

(g) A Spatial Transport Model (STM):

type
  STM = Σ $\times$ τ $\times$ Θ $\times$ R
    $\times$ take $\times$ movement_ρ $\times$immovable
    $\times$ passive $\times$ same_route $\times$ refuted_route

4. Hierarchies of Transport Models

(a) Timed Transport Model:

value
  later: Σ $\times$ Σ $\rightarrow$ **Bool**
axiom
  [Later]
  $\forall$ σ,σ',σ'':Σ •
    later(σ,σ) $\wedge$ (later(σ,σ') $\wedge$ later(σ',σ'') $\Rightarrow$ later(σ,σ'')) $\wedge$
      (τ(σ,σ') $\wedge$ later(σ',σ) $\Rightarrow$ later(σ,σ'))

(b) Controlled Transport Models:

value
  nice: $\Theta \to$ **Bool**
  continuation: $\Theta \times \Theta \to$ **Bool**
  uncontrolled_θ: $\Theta \to$ **Bool**
axiom
  [Continuation]
  $\forall\ \theta,\theta':\Theta \bullet$
    continuation$(\theta,\theta') \equiv$
      $(\forall\ \sigma:\Sigma \bullet \sigma \in \theta \Rightarrow \sigma \in \theta') \wedge$
      $(\forall\ \sigma':\Sigma \bullet \sigma' \in \theta' \wedge (\exists\ \sigma:\Sigma \bullet \sigma \in \theta \wedge \tau(\sigma',\sigma)) \Rightarrow \sigma' \in \theta)$
  [Uncontrolled_θ]
  $\forall\ \theta:\Theta \bullet$ uncontrolled_$\theta(\theta) \equiv$
    $(\forall\ \theta':\Theta \bullet \sim \rho(\theta') \wedge$ continuation$(\theta,\theta') \Rightarrow \sim$ nice$(\theta'))$

5. Top-Down Model Construction:

(a) A Refutation Model (RM):

type
  RM = Object $\times\ \rho$

where:

- *Object:* is an abstract set,
- ρ*:* is a property on *Object*.

scheme
  REFUTATION_MODEL =
    class
      type Object
      value
        ρ: Object $\to$ **Bool**
    end

(b) A Transition Model (TM):

type
  TM = $\Sigma \times \tau \times$ Transitions $\times\ \rho$

where:

- Σ is an abstract set,
- τ is a binary relation on Σ,

- *Transitions* is the set of all subsets of *'tau*,
- ρ is a property on *Transitions*.

scheme
TRANSITION_MODEL =
class
type Σ
value
τ: $\Sigma \times \Sigma \to$ **Bool**
type
Transitions =
$\{|$ pairs:$(\Sigma \times \Sigma)$**-infset** $\bullet$ $\forall$ (σ,σ'):$\Sigma \times \Sigma$ $\bullet$ $(\sigma,\sigma') \in$ pairs $\Rightarrow \tau(\sigma,\sigma')$ $|\}$
value
ρ:Transition $\to$ **Bool**
end

6. Bottom-Up Model Construction:

   Let:

   type
   EM $= \Sigma \times \tau \times \Theta \times \rho$
   EM$' = \Sigma' \times \tau \times \Theta \times \rho$

   be evolutionary models.

   Let:

   value
   hom: $\Sigma \to \Sigma'$

   be a total function.

   This function, *hom*, is extended in the usual way on the set of all subsets of Σ.

   Then:

   axiom
   [H1]
   $\forall$ σ,σ':Σ $\bullet$ $\tau(\sigma,\sigma') \Rightarrow \tau(\text{hom}(\sigma),\text{hom}(\sigma'))$
   [H2]
   $\forall$ θ:Θ $\bullet$ $\sim\rho(\theta) \Rightarrow \sim\rho(\text{hom}(\theta))$

7. "Transport" Logics:

(a) General Transport Logic (GTL):

type
Σ, Θ, R
value
θ: Θ
τ: $\Sigma \times \Sigma \rightarrow$ **Bool**
ρ: $\Theta \rightarrow$ **Bool**
take: R $\times$ R $\times$ $\Sigma \rightarrow$ **Bool**
belong: $\Sigma \times \Theta \rightarrow$ **Bool**
equality: $=$

(b) Θ:

axiom
[Partial Order]
$\forall\ \sigma,\sigma',\sigma'':\Sigma\ \bullet$
$((\tau(\sigma,\sigma') \wedge \tau(\sigma',\sigma'') \Rightarrow \tau(\sigma,\sigma'')) \wedge \tau(\sigma,\sigma)) \wedge$
$(\tau(\sigma,\sigma') \wedge \tau(\sigma',\sigma) \Rightarrow \sigma = \sigma')$
[Total_Order]
$\forall\ \theta:\Theta \bullet \forall\ \sigma,\sigma':\Sigma \bullet \sigma \in \theta \wedge \sigma' \in \theta \Rightarrow \tau(\sigma,\sigma') \vee \tau(\sigma',\sigma)$
[Extensionality]
$\forall\ \theta,\theta':\Theta \bullet (\forall\ \sigma:\Sigma \bullet \sigma \in \theta \equiv \sigma \in \theta') \Rightarrow \theta = \theta'$

(c) Separation:

[Separation]
$\forall\ \theta:\Theta \bullet \exists\ \theta':\Theta \bullet\ \forall\ \sigma:\Sigma \bullet \sigma \in \theta' \equiv \sigma \in \theta \wedge \mathrm{P}(\sigma)$

(d) Unitable:

value
unitable:$\Theta \times \Theta \rightarrow$ **Bool**
axiom
[Unitable]
$\forall\ \theta,\theta':\Theta\ \bullet$
unitable$(\theta,\theta') \equiv (\forall\ \sigma,\sigma':\Sigma \bullet (\sigma \in \theta \wedge \sigma' \in \theta) \Rightarrow \tau(\sigma,\sigma') \vee \tau(\sigma',\sigma))$
[Union]
$\forall\ \theta,\theta':\Theta\ \bullet$
unitable$(\theta,\theta') \Rightarrow (\exists\ \theta'':\Theta \bullet \forall\ \sigma:\Sigma \bullet \sigma \in \theta'' \equiv \sigma \in \theta \vee \sigma \in \theta')$

E.5 MI^{2}CI: A Manufacturing Industry Systems

With growing pressure to sustain global competition, to deliver products of higher quality, at lower cost and with shorter time-to-market, perhaps even to fulfill individual customer requirements, many manufacturing firms are forced to reconsider the ways they run their business. With increased expertise to design, engineer and manufacture complex products, companies are also forced to work more tightly together, perhaps sharing or outsourcing some of their business processes within an extended (virtual) enterprise. Facing such global trends require dedicated, empowered people provided with up-to-date information and modern tools to support their decisions. Development of such tools is the main focus of this project.

E.5.1 Introduction

MI^{2}CI[55] is an acronym which stands for: Manufacturing Industry Information and Command Infrastructure. Started in 1994, the project has gone through many stages to evolve into a long-term research and development project [54].

- Research concerns methods of specifying, describing, engineering and re-engineering of manufacturing enterprises, all firmly based in formal domain models and extending similar (but formal) methods for software.
- Development is to produce a set of generic models, in all major aspects of an enterprise, a language to help specify, write and combine such models, and a set of tools to support writing, analysis and modification of such enterprise descriptions.

We are concerned about decision-supporting tools, comprising languages to describe aspects of an enterprise, methods of using these languages and related software, be based on solid grounds and achieve the fullest possible help from computerised presentation, reasoning and analysis. This asks that particular descriptions of an enterprise, are all given precise mathematical meaning (semantics) that can be understood and manipulated by computers. This is contradictory, however, to making the concepts understandable to human users, in particular to domain experts. Here is where domain-specific languages come to help, with syntax capturing concepts familiar to domain experts, and formal semantics understood by computers. There will be three such languages, to express:

1. objectives (business goals),
2. structure (enterprise resources) and
3. behaviour (business processes) or an enterprise.

[55] This section is written by Tomasz Janowski

Thus seen, an enterprise consists of a set of business processes executed on the set of enterprise resources to fulfill the business goals. For instance, to manufacture and deliver given quantity of a product (business process) using shop-floor and logistics systems (resources), with low failure-rate and rapid delivery (business goal). All three languages are given formal semantics in terms of domain-specific models, capturing the life-cycle of a product (from marketing, design, engineering, production, delivery, to marketing again), enterprise itself (marketing, production, finances and administration), even a virtual enterprise, made of resources owned by other enterprises.

E.5.2 Domain Modelling and Analysis, An Example

This section exemplifies development and analysis of enterprise models, in three parts:

1. a model of an enterprise,
2. a set of such models as a model for a virtual enterprise and
3. structure-preserving abstraction between models of a single and a virtual enterprise.

Modelling

Consider an enterprise which holds the stock for various products and is able to: buy more products from the market, sell products to the market and transform products internally, as depicted in Figure 21. We let the transformation proceed according to the common bill of products: what are immediate sub-products of a given product (if any) and how many are needed to assemble a single unit of this product.

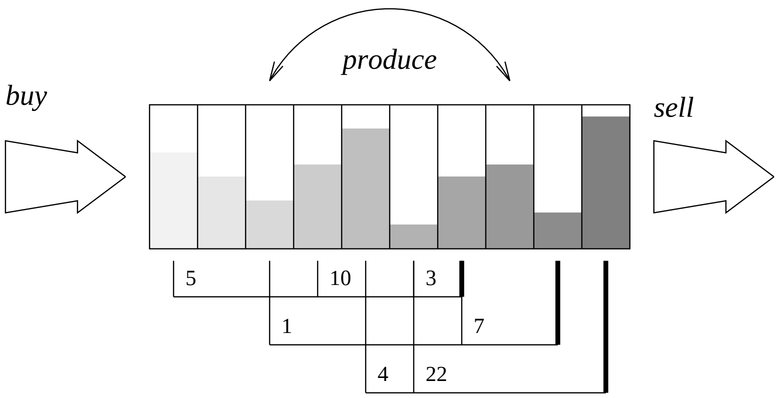


Figure 21: A model of a manufacturing enterprise.

Formally, we first define a type of products. The type is a sort (its structure not elaborated), reflecting the fact that products are primitive to our interests. A bill of products is then a well-formed map, as captured in the predicate *iswf*, from products to multi-sets of products (a set allowing for multiple copies of its elements): given a product the map returns its immediate sub-products and their quantities. The multi-set itself is represented as a map from products to natural numbers, all formalised below.

$$
\begin{array}{l}
\mathbf{type} \\
\quad \mathrm{Product}, \\
\quad \mathrm{Bill}' = \mathrm{Product} \underset{m}{\rightarrow} (\mathrm{Product} \underset{m}{\rightarrow} \mathbf{Nat}), \\
\quad \mathrm{Bill} = \{|\ \mathrm{b} : \mathrm{Bill}' \bullet \mathrm{iswf(b)}\ |\}
\end{array}
$$

The predicate *iswf* will select the maps that could reasonably represent a bill of products, defined in terms of auxiliary predicates: *isprod*, a product is in the bill; *isimsub*, an immediate sub-product relation; *issub*, a sub-product (immediate or not) relation.

$$
\begin{array}{l}
\mathbf{value} \\
\quad \mathrm{isprod} : \mathrm{Bill}' \times \mathrm{Product} \rightarrow \mathbf{Bool}, \\
\quad \mathrm{isimsub, issub} : \mathrm{Bill}' \times \mathrm{Product} \times \mathrm{Product} \xrightarrow{\sim} \mathbf{Bool} \\
\mathbf{axiom\ forall}\ \mathrm{b}' : \mathrm{Bill}', \mathrm{p, q} : \mathrm{Product} \bullet \\
\quad \mathrm{isprod(b',p)} \equiv \mathrm{p} \in \mathbf{dom}\ \mathrm{b}', \\
\quad \mathrm{isimsub(b',q,p)} \equiv \mathrm{q} \in \mathbf{dom}\ \mathrm{b'(p)}\ \mathbf{pre}\ \mathrm{isprod(b',p)}, \\
\quad \mathrm{issub(b',q,p)} \equiv \mathrm{isimsub(b',q,p)} \vee \\
\qquad (\exists\ \mathrm{r} : \mathrm{Product} \bullet \mathrm{isprod(b',r)} \wedge \mathrm{isimsub(b',r,p)} \wedge \mathrm{issub(b',q,r)})\ \mathbf{pre}\ \mathrm{isprod(b',p)}
\end{array}
$$

A well-formed bill of products will not allow a product which is a sub-product of itself or a sub-product which is not registered in the bill:

$$
\begin{array}{l}
\mathbf{value} \\
\quad \mathrm{iswf} : \mathrm{Bill}' \rightarrow \mathbf{Bool} \\
\quad \mathrm{iswf(b')} \equiv (\forall\ \mathrm{p, q} : \mathrm{Product} \bullet \mathrm{isprod(b', p)} \Rightarrow \\
\qquad \sim \mathrm{issub(b', p, p)} \wedge (\mathrm{isimsub(b', q, p)} \Rightarrow \mathrm{isprod(b', q)}))
\end{array}
$$

Suppose, for simplicity, that all enterprises share the bill of products, ***bill***:

$$
\begin{array}{l}
\mathbf{value} \\
\quad \mathrm{bill} : \mathrm{Bill}
\end{array}
$$

For each product of interest, the enterprise will contain information about the amount on stock and ability to assemble from sub-products, as given in the bill. The enterprise is then the well-formed map from products to the records of their stock and transformation capacity; the map is well formed if all products are recorded in the bill.

type

Record :: stock : **Nat** $\leftrightarrow$ re_stock trans : **Bool** $\leftrightarrow$ re_trans,
Ent$'$ = Product $\overrightarrow{m}$ Record,
Ent = {| e$'$: Ent$'$ • **dom** e$'$ $\subseteq$ **dom** bill |}

Operating on the enterprise are functions for buying and selling products, incrementing or decrementing the stock, and for transforming products from sub-products, decrementing some stocks (for sub-products) and incrementing other (for the final product).

value

buy, sell, prod : Ent $\times$ Product $\times$ **Nat** $\stackrel{\sim}{\rightarrow}$ Ent

axiom forall e : Ent, p : Product, n : **Nat** •

buy(e,p,n) $\equiv$ e†[p$\mapsto$re_stock(stock(e(p))+n,e(p))] **pre** p $\in$ **dom** e,
sell(e,p,n) $\equiv$ e†[p$\mapsto$re_stock(stock(e(p))$-$n,e(p))] **pre** p $\in$ **dom** e $\wedge$ stock(e(p)) $\geq$ n,
prod(e,p,n) $\equiv$ buy(e,p,n)†[q$\mapsto$sell(e,q,n$*$bill(p)(q))(q) | q:Product • isimsub(bill,q,p)]
  pre p $\in$ **dom** e $\wedge$ trans(e(p)) $\wedge$
    ($\forall$ q : Product • isimsub(bill,q,p) $\Rightarrow$ stock(e(q)) $\geq$ n$*$bill(p)(q))

Composition

A simple view of a virtual enterprise is that it consists of a set of enterprises, statically interconnected according to what products they are able to buy and sell, as in Figure 22.

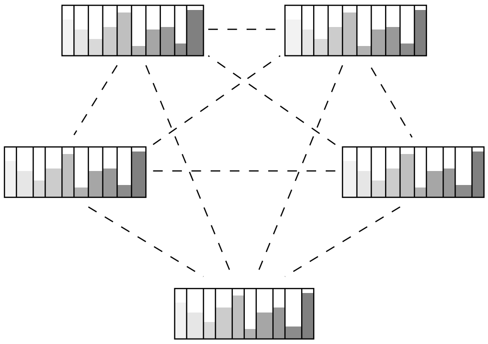


Figure 22: A model of a virtual enterprise.

Formally, a virtual enterprise is defined as a set of enterprises with operations for new enterprises to join the set, for existing ones to leave and for the transfer of products between two enterprises (one seller, one buyer), both members of the set.

type
  VEnt = Ent-**set**
value
  join, leave : Ent $\times$ VEnt $\xrightarrow{\sim}$ VEnt,
  trans : Ent $\times$ Ent $\times$ Product $\times$ **Nat** $\times$ VEnt $\xrightarrow{\sim}$ VEnt
axiom forall e, e1, e2:Ent, v:VEnt, p:Product, n:**Nat** •
  join(e,v) $\equiv$ v $\cup$ {e} **pre** e $\notin$ v,
  leave(e,v) $\equiv$ v\{e} **pre** e $\in$ v,
  trans(e1,e2,p,n,v) $\equiv$ v \ {e1,e2} $\cup$ {sell(e1,p,n),buy(e2,p,n)}
    pre {e1,e2} $\subseteq$ v $\wedge$ p $\in$ **dom** e1 $\cap$ **dom** e2 $\wedge$ stock(e1(p)) $\geq$ n

Abstraction

Consider a function *abstract* by which a virtual enterprise would be seen as a single enterprise, only more complex inside. Such a function would provide a useful tool for studying the interactions of a virtual enterprise with its environment and for scaling up the concept (Figure E.5.2). It takes a virtual enterprise and returns an enterprise which stock equals the sum of all component stocks and which capacity to transform products equals capacities of all its components:

value
  abstract : VEnt $\rightarrow$ Ent
  abstract(v) **as** f **post**
    ($\forall$ p:Product • p $\in$ **dom** f $\Rightarrow$
      ($\exists$ e : Ent • e $\in$ v $\wedge$ p $\in$ **dom** e) $\wedge$
      stock(f(p))=sum({stock(e(p)) | e:Ent • e $\in$ v $\wedge$ p $\in$ **dom** e}) $\wedge$
      trans(f(p))=dis({trans(e(p)) | e:Ent • e $\in$ v $\wedge$ p $\in$ **dom** e }))

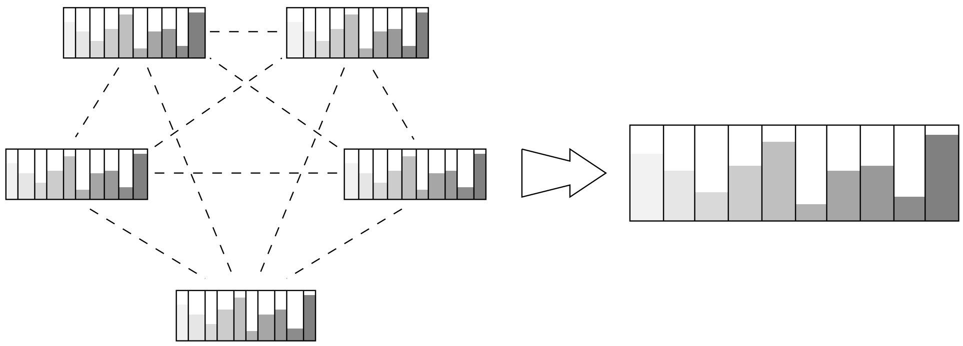


Figure 23: Abstraction from the virtual enterprise.

We need to justify that abstraction didn't ignore some salient features of the model and there exists a correspondence between operations on the abstract model and on components of the concrete one, by showing the following diagrams to commute:

$$
\begin{tikzcd}
vent \arrow[r, "abstraction"] \arrow[d, "buy/sell"] & ent \arrow[d, "buy/sell"'] \\
vent' \arrow[r, "abstraction"] & ent'
\end{tikzcd}
\qquad
\begin{tikzcd}
vent \arrow[r, "abstraction"] \arrow[d, dotted, "trans"] & ent \arrow[dd, "prod"'] \\
\cdot \arrow[d, "prod"] & \\
vent' \arrow[r, "abstraction"] & ent'
\end{tikzcd}
$$

For buying or selling products, the effect after applying abstraction is the same as if we first apply abstraction and then buy/sell from the resulting enterprise. This is similar for production, except we may need a sequence of transactions between enterprises before production within one of them can take place, matching production within a virtual enterprise.

E.5.3 Conclusions

There are several stages that we pursue in domain modelling for manufacturing:

- Broad (informal) study of the issues in manufacturing industry, as they appear in the intra-enterprise, inter-enterprise and inter-market contexts. The emphasis is on decision-making within marketing, administration, finance and production — see [**62**] mentioned in [54].
- Intra-enterprise modelling. Formalisation of the structure and operations within a manufacturing enterprise, considering the basic concepts in production, marketing, finances and engineering. A clear separation between the enterprise resources (four Ms: men, machines, materials and money) and the operations on them — see [**63**] mentioned in [54].
- Inter-enterprise modelling. Consider an enterprise which is made entirely of operations provided by other enterprises. An integrating concept is an electronic trader (yellow pages for manufacturing) or traders. Show formally that this virtual enterprise can be seen as a single enterprise, only more complex inside — see [**69**] mentioned in [54].
- Application of inter-enterprise modelling. The first area is marketing decision-making – formalise the marketing mix (product, place, price and promotion) and capture the choice of the mix as an optimisation problem — see [**92**] mentioned in [54]. The second area is concurrent engineering – utilise concurrency within a virtual enterprise and decompose design tasks based on the formal notation to write down design requirements and design decisions.

We now start the second phase of the project – a calculus for business process re-engineering. This will properly build on the earlier stage, especially in defining the semantics of languages comprising the calculus: to describe enterprise resources and business processes and goals.

E.6 MultiScript: Multi-Lingual Script System

MultiScript[56] is an acronym whose designation should be clear from the title of this section.

MultiScript started, in 1992, with a number of so-called MoTraS (for Mongolian Traditional Script) project proposals — and some project work [35]. Finally as from September 1995 the current project took off [46].

E.6.1 Basic Concepts

The basic idea of both the MoTraS proposals and the current MultiScript project can be summarised:

- Domain Theory & Analysis:

  To understand, describe, include formalise, the domain of composite multi-lingual texts. That is texts such as:

  - LtRC-WTtB: *Mongolian* — which occur[57] in sequences of left-to-right columns with words proceeding top-to-bottom within columns;
  - RtLC-CTtB: *Old Chinese (Japanese, etc.)* — which occur in sequences of right-to-left columns with characters proceeding top-to-bottom within columns;
  - TtBR-WCRtL: *Arabic (Hebrew, Kurdish, etc.):* — which occur in sequences of top--to-bottom rows with words (and word characters) proceeding right-to-left within rows;
  - TtBR-WCLtR: *English (Vietnamese, etc.):* — which occur in sequences of top-to-bottom rows with words (and word characters) proceeding left-to-right within rows;

- Requirements Capture & Definitions:

  To understand, describe, including formalise, requirements for:

  - System: the computerised input, editing, formatting (incl. presentation) of multi-lingual documents (i.e. texts containing one or more of the four "display" directions).
  - UNICODE: MultiScript requirements to UNICODE.

- Software Design:

  To understand and document, informally and formally:

  - System: A software architecture, a program organisation and a prototype implementation (i.e. code) for (most likely a subset of) the required system.

[56] This section is to be written by Richard Moore and Myatav Erdenechimeg

[57] By 'occur' we mean: "are normally written and thus normally read" (i.e. displayed)

– UNICODE & Glyphs: Exploratory (experimental, prototype) implementations of Mongolian encodings in UNICODE etc.

MultiScript Requirements thus destine it as a rather special version of a Workpiece Frame problem:

- Templates:

  The various rules governing the character and sign glyphs (etc.) and the text formatting of the MultiScript languages (i.e. scripts), form Templates.

- Designs:

  Largest single script text fragments form Designs.

- Aggregates:

  And combinations of two or more distinct script fragments form Aggregates.

That is: MultiScript has no intention of moving in the direction of Machine Translation — but any MultiScript implementation should represent its data in a transparent manner such as to facilitate any Connection to other researchers Machine Translation systems!

E.6.2 Background

There are several thousand languages in use in the world today, for an increasing number of which some software support is available. For many of the European families of languages, such support is relatively easy to add on to existing Roman[58] text and document processing systems, often requiring little more than the addition of a few extra characters to the font and the assignment of these characters to keys on the keyboard. For other languages, notably the oriental and Arabic families, things are much more difficult for several reasons.

First, these languages tend to have their own character sets which are not simple extensions or modifications of the English character set. This means that existing fonts generally cannot be used and whole new fonts have to be designed to support these languages.

Second, many of these languages have character sets which are much larger than those of the European languages, sometimes incorporating thousands of distinct characters. For such languages there are clearly not sufficient keys on the keyboard to allow a simple mapping from a single key to a single character as is done on an English keyboard. Instead more complicated input methods are required, often involving the use of multiple keystrokes or some sort of graphical interface to generate a single character.

[58] Here 'Roman' means any language whose character set consists of the basic English alphabet, possibly modified by the addition of accents to some letters or extended with such accented letters.

Finally, many of these languages are not traditionally written in the same direction as the European languages, some, particularly those of the Arabic family, being written horizontally but right to left instead of left to right, others being written vertically. In fact this latter group is itself subdivided into languages in which the lines[59] of characters are ordered right to left (for example Chinese, Japanese and Korean) and those in which they are ordered left to right (for example Mongolian).

Where the software support for these languages has been provided by extending some existing Roman-based software, the extension has tended to be concentrated on the first two aspects, namely on the provision of the extra character sets (fonts) and the modification or extension of the keyboard input mechanism, and the question of directionality has mainly been ignored. To a large extent this is of course to be expected, as without the ability to display the characters of a language or to input them into the computer in some way the question of the reading and writing direction is largely academic. As a consequence, however, people wishing to use these systems to process non-Roman texts, particularly those of the oriental languages, have generally been forced to do so using the standard European directionality. This is unnatural and can make reading and writing the text extremely difficult[60].

Software support which was primarily designed to support non-Roman languages often does allow its users to write and read documents in the appropriate "natural" direction and sometimes allows the user to choose the reading and writing direction of a document. However, even such systems are not truly multi-lingual as they do not allow this direction to change within the document.

E.6.3 A Domain Theory

An extensive study of a wide range of different multi-lingual documents reveals that there are essentially two different ways in which texts in different languages, possibly with different reading and writing directions, can be intermixed in a single document. In the first type, the parts of the text with the different languages and/or orientations are clearly separated.

In the second type, parts of one language are embedded in the other, so that, for example, a sentence in one language is "interrupted" by insertions in another language. If both languages have the same natural orientation this is used by both, but there are again two different basic ways in which the insertion can happen in cases where the two languages have different natural orientations.

First, the embedded language can retain its own natural orientation. Alternatively, the embedded language loses its natural orientation and instead adopts that of the language in which it

[59] The word 'line' in the sense of 'lines of text' is used generically to refer to both horizontal and vertical text. It should not be taken to imply that the text is horizontal. The word 'column' is also used to describe vertical text.

[60] nettirw hsilgnE daer ot rehtie deirt sah ohw enoyna sa yna ni deedni ro) sdrawkcab hsilgnE etirw ot ro sdrawkcab !etaicerppa ylidaer dluohs (noitcerid dradnats-non rehto

is embedded. In this case the embedded language is often written with its characters rotated when one of the natural orientations is horizontal and the other vertical.

We define a *frame* to be a piece of text written in a single language[61] and within which the reading and writing direction does not change. In more physical terms a frame can also be thought of as a rectangular area (on a computer screen or on a piece of paper) within which the text is displayed or printed.

The orientation of a frame is determined by its *entry point* and its *line stream*. The entry point defines the position at which reading or writing begins, and is situated either at the top left-hand corner or the top-right-hand corner of the frame. We are not aware of any languages in which reading and writing start at the bottom of the frame, so it is sufficient to define the entry point simply as either left or right.

The line stream defines the direction in which the text is read or written and is either horizontal or vertical. Consecutive lines of text are aligned perpendicular to the line stream and proceed downwards in the case of horizontal text and horizontally away from the entry point in the case of vertical text.

Thus, for example, a frame whose entry point is left and whose line stream is horizontal corresponds to horizontal lines of text with each line being read or written from left to right (away from the entry point) and with the lines themselves ordered vertically downwards, of which English text is an example. Arabic, which is written horizontally from right to left, has the same horizontal line stream but its entry point is right. Traditional Chinese text is written vertically in columns with the columns progressing from right to left, so it has a frame whose entry point is right but whose line stream is vertical. And Mongolian, which is written in columns which progress from left to right, also has vertical line stream but its entry point is left.

The contents of a frame can be either ordered or unordered, or may indeed consist of an ordered part and an unordered part. In the ordered part, the position of each character is determined by the characters preceding it, whereas in the unordered part there is no logical connection between its various pieces. An example of an unordered frame might be a page of a newspaper, which essentially consists of a collection of logically unrelated, or perhaps weakly related, articles which can effectively be positioned at will by the people producing the newspaper.

The ordered part of a frame is essentially a list of pieces of text, each with some associated formatting information, because different pieces of text may appear in different formats (different font, typeface, colour, with rotated characters, etc.) without changing the reading and writing direction. Each such piece of *formatted text* is then composed of some formatting information and a list of *tokens*, where a token may be either a *character* (printing or non-printing) or a frame. This latter allows a frame to have pieces of text with different reading and writing directions embedded in it.

[61] More strictly perhaps, which can be generated using the characters of a single font. We can clearly produce, say, German text with English insertions using only a German font (and keyboard) because the English character set is a subset of the German.

The unordered part of the contents of a frame represents objects which have no logical connection with the ordered part or with each other and which can be positioned at will when the document is displayed or printed. Each of these is just a frame, and in terms of the display or printing of the document represents an area of the frame which its ordered component is not allowed to fill. These "holes" may either contain text themselves or may simply be empty, perhaps representing the location of images. We refer to these frames as *free frames*.

A document is then just a frame whose orientation is determined by the document's main language.

E.6.4 A Formal Model of Multi-lingual Documents

Based on the analysis presented in Section E.6.3, we can develop a formal model in RSL which describes general multi-lingual documents.

In RSL we can define the properties of a frame (entry point and line stream) via type definitions:

type Entry_Point == left | right,

Line_Stream == horizontal | vertical

Here we use variant definitions to simply list all possible values of the type. We then combine these definitions into another type, Orientation, which specifies all the properties related to the directionality of a frame, using a record type:

type Orientation :: entry : Entry_Point
stream : Line_Stream

A frame itself is then composed of an orientation, a list of pieces of formatted text, and a list[62] of its free frames, the last two representing its ordered and unordered contents respectively.

type Frame :: orient : Orientation
string : FormattedText^*
freeFrames : Frame^*

Formatted text is made up of formatting information and a list of tokens, where each token may be either a character or a frame as represented by the union type in the definition below.

[62] Although this means that the free frames are in fact ordered in the model, this order is used purely as a convenience and is ignored when the contents of the frame are displayed.

The types Character and FormatInfo are defined as sorts, i.e. they are abstract types, the precise details of which are unimportant for the specification. Character will clearly include the character sets of the various languages we wish to include in our documents as well as various formatting (control) characters used for defining the structure and the layout of the document when it is printed or displayed. FormatInfo will include such things as the font, the typeface, and the rotation of the characters.

type FormattedText :: format : FormatInfo
chars : Token*,

Token = Character | Frame,

Character, FormatInfo

Finally, a document is represented simply as a frame:

type Document :: frame : Frame

Concepts such as the position of a "cursor" within a document, and functions for editing and displaying documents can then be defined as extensions to this model.

E.6.5 On Requirements

The most fundamental requirement of the MultiScript software system is that it should be possible to use it to create, modify and display multi-lingual documents such as those described above.

One way of achieving this would be to design a completely new system based around the model presented in Section E.6.4, but this is somewhat wasteful because there are already many systems which offer good support for certain languages both in terms of providing fonts covering the appropriate character sets and in providing a good method of inputting those characters. Indeed, although each tool in itself may only support a small set of different languages, taken as a whole they cover a very wide range. The main drawback of these systems is that they generally do not support the whole range of different reading and writing directions and character rotations which are required to support truly multi-lingual documents. Indeed many insist that all the languages they support should be written and read in the same sense, and they can thus be said rather to support only the different alphabets (or character sets) of the languages rather than the languages themselves.

Another drawback is that many of the tools use different encoding schemes for their character sets, which makes them mutually incompatible as they stand. Neither do the majority of them provide support for document processing (e.g. defining the structure of a document) as opposed to simple text processing.

Rather than trying to re-implement useful functionality that is already available in existing tools, however, the MultiScript system should provide an environment within which a wide range of existing tools can be integrated and which in addition allows the reading and writing directions of pieces of text to be changed and permits the rotation of the characters it contains. More specifically, it should allow lines of text to be written vertically, with the lines ordered top to bottom, or horizontally, with the lines ordered either left to right or right to left, and it should permit characters to be displayed either rotated 90° in either direction or upside down. It should also allow both the font associated with some piece of text and the size of the frame in which the text is displayed to be changed, and perform any hyphenation needed as a result of this reformatting.

The system should also provide its own functions for editing and creating multi-lingual documents and for displaying them.

E.6.6 On Software

We plan to implement a prototype (demonstrator) MultiScript system in the next phase of the project.

E.6.7 Discussion

The MultiScript project has been running for just over one and a half years. A paper on the project has been presented at the 1997 conference on the Computer Processing of Oriental Languages, and one UNU/IIST report ([75]) has been prepared and another is in preparation. Several internal technical documents have also been written.

E.7 ABC'2000:

ABC'2000[63] is an acronym for Airline Business Computing.

The ABC'2000 Requirements destine it to become an Information + Workpiece Multi-Frame problem.

E.7.1 Airline Business: Informal Analysis

Just as with any business, the airline business can be viewed from several different aspects and at several different levels. First, one might consider what might be called an *operational* view of the business, looking at the internal functioning of the business in terms of its processing of resources into products. Both resources and products may be either services, as in the case of the airline business where the main product is the transportation of people and/or goods from one location (airport) to another, or material objects. The resources of an airline business include its aircraft and its staff.

A second view of a business, which might be termed the *organisational* view, looks at the interactions between the business as a whole and the outside world, namely its suppliers and customers, where suppliers covers both suppliers of materials and suppliers of services. In the airline business, suppliers include, for example, the catering companies which provide meals for passengers and the maintenance engineers who keep the aircraft airworthy, while customers are passengers and cargo transport agencies.

One could also consider what might be called the *management* view of a business, dealing with the financial and planning aspects of the business and with ensuring that the necessary resources are available to enable the business to provide the products required by its customers. Viewed in this respect, the airline business is very much like any other business.

We concentrate here on the operational view, considering resources and products and how the former are converted into the latter.

E.7.2 Operational View: Airline Resources

An airline requires many resources to operate, the main ones being those which are used directly in providing the products of the business (the flights), namely aircraft and staff. Physically an aircraft is defined by its make and model (for example, Airbus A32S, Boeing 747-300), and by its cabin and hold configurations (passenger and cargo capacities for different classes of service). The flight history of an aircraft is also important as it determines not only its useful lifetime but also its servicing and maintenance requirements. Servicing and maintenance must be performed

[63] This section is to be written by Richard Moore

by engineering staff who are qualified to work on the particular type (make and model) of aircraft.

Airline staff include air crews (pilots, co-pilots, cabin staff, etc.), ground crews (check-in staff, gate staff, etc.), sales staff, administrative staff and management personnel. Staff are defined by their personal data (name, sex, date of birth, etc.) and their technical skills and qualifications. Only staff with the appropriate technical skills and qualifications can be assigned to particular tasks.

Other supporting resources, such as fuel (both for aircraft and for other vehicles), passenger tickets and cargo bills, in-flight catering and entertainment, airport vehicles, maintenance facilities, and airport facilities, are used by the main resources and thus contribute only indirectly to the products. Generally speaking, most of these are provided by external sources, whether on a day-to-day basis (in the case of fuel, in-flight entertainment and catering supplies, maintenance facilities and airport facilities) or infrequently (in the case of airport vehicles, property and office equipment). As far as the airline is concerned, therefore, these may be considered simply as raw materials which are purchased ready made. As a result, the airline only has control over how many of what type of each are available at any time, the airline management system ensuring that the supply is always sufficient.

Cash is a resource which can be converted into other resources (by purchasing resources) or which other resources can be converted into (by selling resources). It is also obtained by selling products to customers. Passenger tickets and cargo bills are given to customers in exchange for cash and are presented back to the airline when the customer redeems his purchase.

Resources have various properties, the most significant one as far as an airline is concerned being that its main resources move from one location to another in the process of providing a flight. Some of the flight support resources also have this property, for example aircraft fuel: a certain amount of fuel is loaded onto an aircraft before a flight and a reduced amount remains at the end of the flight and is therefore available as a resource at the flight's destination.

Another important property of a resource is its availability at a given time, or more strictly for a given time interval. For example an aircraft and its crew have to be available for the duration of a flight they are to be used on. An aircraft may be unavailable if, for example, it is undergoing maintenance. Staff may be unavailable if they are sick or on vacation.

The allocation of resources at the correct locations and for the required time intervals (scheduling) is one of the functions of the airline management.

E.7.3 Operational View: Airline Products

There is only one product, the transportation of passengers and cargo from one place to another[64], in other words a *flight*. Geographically, a flight is defined by its departure and arrival points, departure and arrival times, and flight route. Physically, a flight requires an aircraft and its crew, and may additionally have either or both of passengers and cargo freight allocated to it. The crew should be suitably qualified to perform the flight, and the passenger and cargo allocations should not exceed the aircraft capacity.

E.7.4 A Formal Model of Aircraft and Flights

To specify an aircraft we need to define types which describe not just the basic aircraft but also the notions of class of service for passengers and cargo, and cabin and hold configurations. We use sorts to define class of service for passengers and cargo separately (allowing for the fact that they might be different). We then model the cabin and hold configurations of the aircraft as mappings from the appropriate class of service to *PassengerCapacity* and *CargoCapacity*, abstract types representing the space available for seats and cargo for each class. The total configuration of the aircraft is then represented by the type *AircraftConfig*, which is a record consisting simply of two components, one recording the cabin configuration, the other the hold configuration.

$$
\begin{array}{l}
\textbf{type} \\
\quad \text{Aircraft}, \\
\quad \text{SeatClass}, \\
\quad \text{CargoClass}, \\
\quad \text{PassengerCapacity}, \\
\quad \text{CargoCapacity}, \\
\quad \text{CabinConfig} = \text{SeatClass} \underset{m}{\rightarrow} \text{PassengerCapacity}, \\
\quad \text{HoldConfig} = \text{CargoClass} \underset{m}{\rightarrow} \text{CargoCapacity}, \\
\quad \text{AircraftConfig} :: \text{cabinConfig} : \text{CabinConfig} \;\; \text{holdConfig} : \text{HoldConfig}
\end{array}
$$

We introduce more types to represent flights, airports (for the departure point and destination of the flight), employees (for airline staff), passengers (for customers on the flight), and cargo (for cargo on the flight). In addition we define the type *Traveller* as the union of the types *Passenger* and *Employee*. This allows us to take account of the fact that airline staff may be travelling on a flight although not working as part of the crew. Finally, for convenience we define a type *Crew* to be a set of employees.

[64] We could, of course, consider passengers and cargo separately but there is no advantage in doing so.

type
  Flight,
  Airport,
  Employee,
  Passenger,
  Cargo,
  Traveller = Passenger | Employee,
  Crew = Employee-**set**

Further record types represent the geographical attributes of the flight, specifically its departure point and destination (FlightRoute), and physical resources allocated to the flight, that is its aircraft, crew, passengers and cargo (FlightResources). In the latter, the aircraft is represented by some (unique) aircraft identifier, a member of the abstract type AircraftId. In the definition of the state of the system we impose the constraint that all aircraft belonging to an airline must have distinct identifiers.

type
  AircraftId,
  FlightRoute :: departure : Airport destination : Airport,
  FlightResources ::
    aircraft : AircraftId
    crew : Crew
    passengers : (Traveller $\underset{m}{\rightarrow}$ SeatClass)
    cargo : (Cargo $\underset{m}{\rightarrow}$ CargoClass)

Finally, the type ABC represents an airline business, and observer functions on this type return the aircraft, employees and flights of the airline. Both an aircraft and an employee have an associated location, which is an airport. Flights are split into two parts: current flights, which represents those flights which are currently in progress, and future flights, which represents those flights which are scheduled to take off at some time in the future.

type ABC

value
  flights : ABC $\rightarrow$ Flight-**set** $\times$ Flight-**set**,
  fleet : ABC $\rightarrow$ (Aircraft $\underset{m}{\rightarrow}$ Airport),
  staff : ABC $\rightarrow$ (Employee $\underset{m}{\rightarrow}$ Airport)

E.7.5 Discussion

The ABC'2000 project ran for just over one year, from September 1995 to September 1996, though discussions are underway with a view to resuming the work. A paper on the project was presented at the 1996 Asia Pacific Software Engineering Conference, and one UNU/IIST report ([74]) was prepared. Several internal technical documents were also written.

E.8 DiMulTS: A Philippine Radio Telephone System

DiMulTS[65] is an acronym for Digital Multiplexed Telephone System.

The DiMulTS Requirements destine it to become a Control & Connection Multi-Frame problem.

This project was done in the period May 1995 to February 1996 by Roderick Durmiendo from the Advanced Science and Technology Institute (ASTI) under the Department of Science and Technology of the Republic of the Philippines, together with Chris George.

E.8.1 Introduction

Software for telecommunication systems such as that which controls a digital exchange demands a high-degree of reliability. The various requirements that need to be considered account for most of the complexity in the software development of such systems. One must for instance ensure correct behaviour of a service despite the limited network resources and in addition ensure the correct allocation of these resources. These requirements, however, can be separated using a step-wise approach in the development. Step-wise development involves formulating an abstract specification that captures the high level requirements of a system, and validating it against those requirements. Subsequent steps refine the specification to reveal certain aspects and address more detailed requirements; each step is verified against the previous specification and also validated against the more detailed requirements.

The project is concerned with a "distributed exchange". There is a "central" component of the exchange connected by trunk cables to the public network. Connected to the central exchange by microwave links are a number of "remote" exchanges each serving a local community. So a subscriber in remote community A can connect directly to another subscriber in the same community through their remote exchange. He can connect to another subscriber in Manila, say, on the public network, by the microwave link between his remote exchange and the central exchange. He can also connect to another subscriber in a different community B via two microwave links, from A to the central exchange and from there to B. The distributed exchange effectively has a "star" configuration. It is illustrated in figure 24.

The aim of the project was to develop a specification of the software in the central and remote exchanges. This specification was then used as a basis of the development of the actual system. As part of this the proposed communication protocols were to be validated.

The work was done in a series of steps with validation and verification at each stage.

[65]This section is to be written by Chris George

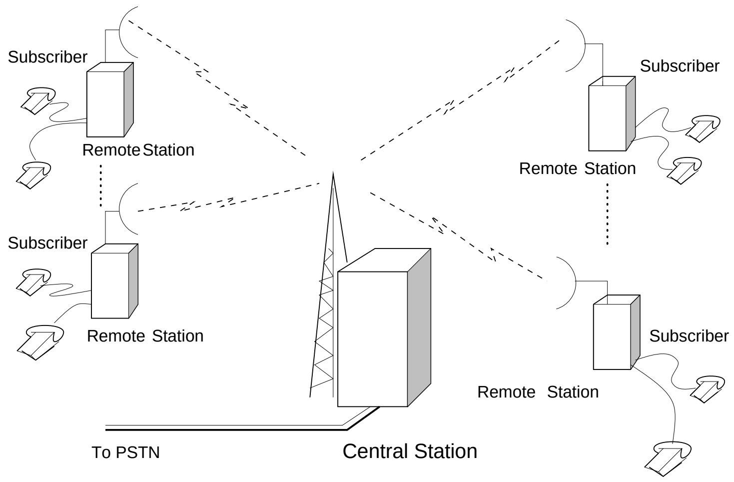


Figure 24: The DiMulTS System

E.8.2 Initial specification

As far as the subscribers are concerned the distribution of the exchange should be invisible (except, perhaps, for a more limited capacity than the normal public network). So the first specification was of a single exchange providing connections and disconnections — the Plain Old Telephone Service (POTS) — between an homogeneous set of subscribers.

A typical axiom from this specification is [*phones_put_down_ax*]. It says that putting down a phone that was previously not on its cradle will change the state to *on_hook*. When putting down a phone, there are two cases where the state of another phone will change. The first is when the other connected phone becomes busy. The second is when the ringing phone goes back to being *on_hook*.

$$
\begin{array}{l}
\textbf{axiom} \\
\quad [\,\mathrm{phones\_put\_down\_ax}\,] \\
\qquad \forall\ \mathrm{no} : \mathrm{T.No},\ \mathrm{s} : \mathrm{State} \bullet \\
\qquad\quad \sim \mathrm{on\_cradle}(\mathrm{no}, \mathrm{s}) \Rightarrow \\
\qquad\quad (\mathrm{phones}(\mathrm{next}(\mathrm{put\_down}(\mathrm{no}, \mathrm{s}))) \equiv \\
\qquad\qquad \textbf{let}\ \mathrm{m} = \mathrm{phones}(\mathrm{s})\ \textbf{in} \\
\qquad\qquad\quad \textbf{case}\ \mathrm{m}(\mathrm{no})\ \textbf{of} \\
\qquad\qquad\qquad \mathrm{dial\_tone} \rightarrow \mathrm{m} \dagger [\,\mathrm{no} \mapsto \mathrm{on\_hook}\,], \\
\qquad\qquad\qquad \mathrm{connected}(\mathrm{no}') \rightarrow \\
\qquad\qquad\qquad\quad \textbf{if}\ \mathrm{no} = \mathrm{no}'\ \textbf{then}\ \mathrm{m} \dagger [\,\mathrm{no} \mapsto \mathrm{on\_hook}\,] \\
\qquad\qquad\qquad\quad \textbf{else if}\ \mathrm{m}(\mathrm{no}') = \mathrm{ringing}(\mathrm{no})\ \textbf{then} \\
\qquad\qquad\qquad\qquad\quad \mathrm{m} \dagger [\,\mathrm{no} \mapsto \mathrm{on\_hook},\ \mathrm{no}' \mapsto \mathrm{on\_hook}\,] \\
\qquad\qquad\qquad\qquad \textbf{else}\ \mathrm{m} \dagger [\,\mathrm{no} \mapsto \mathrm{on\_hook},\ \mathrm{no}' \mapsto \mathrm{busy}\,] \\
\qquad\qquad\qquad\qquad \textbf{end end},
\end{array}
$$

busy → m † [no ↦ on_hook]
end end)

This specification needed to be validated. This was done by positing as an RSL **theory** a number of basic properties of POTS. For example, a subscriber dialling his/her own number will result in a "busy" state:

[dial_busy2]
in PHONE_0 ⊢
∀ s, s′ : State, dig : T.Digits •
T.is_no(dig) ∧ phones(s)(dig) = dial_tone ∧ s′ = dial(dig, dig, s) ⇒
phones(s′)(dig) = busy

Nine such properties were posited and proved of the initial specification.

E.8.3 First development step

Refinements typically involve decomposition of the state, and the first refinement here was in this category. The state of the system was decomposed into a collection of phones and a network providing connections between them.

Connections in fact correspond to the line-cards in exchanges: one for each phone. They may be free, connected or in the process of being connected:

type
Connection,
Line_state ==
free | connected(conn: Connection) | connecting(conn: Connection)

A phone is either on- or off-hook, and in the latter case has a "tone":

type
Ph_state == off_hk(Tone) | on_hk,
Tone == busytn | dialtn | data

Only some combinations of phone and connection are allowed; blank entries in table 25 are prevented by a subtype condition.

	off_hk(busy)	off_hk(dial)	off_hk(data)	on_hk
free				on_hook
connected(cn)	busy	dial_tone	connected(no′)	ringing(no′)
connecting(cn)			connected(no′)	

Figure 25: Possible combinations of Line_state and Ph_state

The states and functions of the initial model were defined in terms of these new types and the development verified by proving the new specification implemented the original one.

E.8.4 Developing the network

The module specifying the phones is now complete; from here on the problem is only that of developing the network into central and remote exchanges plus the microwave links.

The network is developed in two stages. First a number of "trunk" connections (central-remote) and a number of "pstn" connections (central to public switched telephone network) are introduced; the Connection type is decomposed. By also classifying numbers as possibly belonging to remote exchanges it is possible to specify what resources are needed for a particular connection, such as one trunk plus one pstn for the subscriber calling his aunt in Manila.

The development step was verified by proving implementation.

E.8.5 Introducing distribution

The specification still has a single monolithic mapping of phones to the phone cards (represented by Line_state). But in fact, for the remote subscribers, the phone cards will be located locally, so a new type is introduced:

$$\text{Rid} \xrightarrow[m]{} (\text{No} \xrightarrow[m]{} \text{Line\_state})$$

where "Rid" is the identifier of a remote exchange.

The network can now be decomposed into a number of modules: REMOTE for the collection of remote exchanges (which can later be decomposed further into an array of individual exchanges), TRANSMISSION for the microwave links, and CENTRAL for the central switch.

TRANSMISSION is not to be implemented: it is a specification of the microwave links and their associated protocols. It therefore acts as a specification of our *assumptions* about these links. the rest of the specification can then, in theory, be proved correct with respect to these assumptions. We assumed no loss of messages, and no overtaking on connections between the centre and a particular remote. These are ideal, but in practice can be realised most of the time by some low-level protocol. It was decided not to try to model unrecoverable faults in transmission, to keep the specification as simple as possible. Note that there is no assumption that transmissions to or from different remotes will take the same time.

There are only two frequencies available for microwave transmissions, one in to the centre and one out from it. Outward messages are broadcast to all remotes, inward ones are merged. The TDM/TDMA protocols used involve sending "frames" of information which are divided into a number of "slots", called trunks. It is essential for correct operation that at any one time each trunk is allocated to at most one remote exchange. So making connections involve the request and allocation of trunks, and careful management of them.

E.8.6 Verification

Proving the distribution step correct (by showing implementation) was beyond the capacities of the RAISE justification tool. It was tried instead to prove "critical" properties of it. These were the one mentioned above (each trunk allocated at most one remote) and a "no loss" property that each trunk is either free (and known as such by the centre), being allocated (by a message in transit), being freed (again by a message) or allocated (and known to a remote). Only part of the proof of these properties has so far been completed, but stating these properties formally and proving some of the lemmas involved showed problems in the originally intended design, which were corrected.

E.8.7 Conclusion

The work exposed some problems in the original design and enabled them to be corrected. It exposed some limitations in the tools in that the proofs done were very long and those not done perhaps beyond their capacity. But it did increase very considerably the confidence in the final design. It is described in more detail in a research report [67].

The other interesting aspect of this work is that it was done purely applicatively, in line with the RAISE method. The transformation to concurrent components can be done after the design of the system, in particular its decomposition into separate modules. Doing this development applicatively allows one to concentrate on the issues of data design without being concerned with the extra complications arising from imperative and concurrent features. The proofs were difficult in an applicative context; they would be much more complicated in an imperative, concurrent one.

E.9 MoFIT: Vietnam Ministry of Finance IT

MoFIT[66] is an acronym for Ministry of Finance Information Technology.

The MoFIT Requirements destine it to become an Information and Workpiece Multi-Frame problem.

This project is being done with fellows from Vietnam: four from the Ministry of Finance (MoF), one from the Institute of Information Technology (IoIT) in Hanoi, and one from Hanoi University.

E.9.1 Introduction

The project started with discussions between UNU/IIST and the MoF in September 1995 on the basic requirements for a substantial project in Vietnam, supported by the World Bank, to generate a national Financial Information System. A two-week course on RAISE was presented to people from MoF and IoIT in January 1996, and five fellows came to UNU/IIST in April 1996.

The first group of fellows had worked on software for the taxation system, and this system was described first in some detail, both as a narrative written in English and as a formal specification in RSL. The specification splits quite neatly into two parts, taxation accounting and taxation structure.

E.9.2 Taxation accounting

The notion of an account was developed from that done initially during the course as an exercise on external debt accounting, described in the next section E.10. Taxation accounting is complicated in that taxes are based on three components: the current taxation rules, defined by laws laid down by the Government, information about the kinds of activities and hence the kinds of taxation to be paid by taxpayers; and information periodically collected from these taxpayers about their income, turnover, profits etc. These three kinds of information appear separately in the specification, making it easy to see their role and convenient to update them separately.

Taxation is also complicated by the fact that taxation is assessed and paid for periods, and these periods may differ for different kinds of tax. Each kind of tax must be accounted separately, and payments used to pay any arrears from previous periods as well as liabilities for the current period. Arrears attract interest. There is also the possibility of tax exemptions being granted.

The specification of this proved very complicated, and raised issues initially overlooked, like how to handle the perhaps unlikely but certainly possible case of overpayment. To gain more

[66] This section is to be written by Chris George

confidence in the specification it was decided to prototype the accounting system. This was done by making concrete some things that were completely abstract in the main specification, like dates and periods, mechanically translating the specification into C++, and testing it against test cases and expected results generated by other members of the team. This uncovered a few trivial errors but also showed an undocumented assumption (that transactions would be input in date order) that was added as a precondition, and one conceptual error in the specification, to do with how demands for debts from previous periods are dated. The correction was not very complicated, but this is the kind of error that it is important to find early in development.

The prototype ran very slowly (which is not a criticism — specifications show what must be done, and they need not be executable at all). It was decided as an exercise to try some optimisations of the specification. Three approaches were tried and they produced dramatic improvements in efficiency of the generated code. Knowledge gained from such an experiment can be used later in the development towards a real system of the main specification.

E.9.3 Taxation system structure

In Vietnam there is a national, "general" taxation office. Under this there is an office in each province, and under the provincial offices there are district offices. So there is a three-level hierarchy, which has a regular tree structure. Larger businesses are dealt with at the provincial level, smaller ones at the district level. So two levels are involved in accounting (and the same generic accounting module was used by both). Both the province and general office are involved in collecting and combining or summarising reports from lower levels. The structure is illustrated in figure 26.

Reports are described very abstractly. A report has a format Report_form and a number of Lines, determined by the format. Each line in a report has an amount. Two reports may be summarised to generate a third, provided they have the same format, which will also be shared by the result. The amount in each line of the result is the sum of the corresponding lines in the input reports.

type
  Report, Report_form, Line
value
  form : Report $\rightarrow$ Report_form,
  is_in : Line $\times$ Report_form $\rightarrow$ **Bool**,
  amount : Line $\times$ Report $\rightarrow$ T.Amount,
  summary : Report $\times$ Report $\stackrel{\sim}{\rightarrow}$ Report
axiom
  [form_summary]
  $\forall$ rp1, rp2 : Report •
    form(summary(rp1, rp2)) $\equiv$ form(rp1) **pre** form(rp1) = form(rp2),
  [amount_summary]

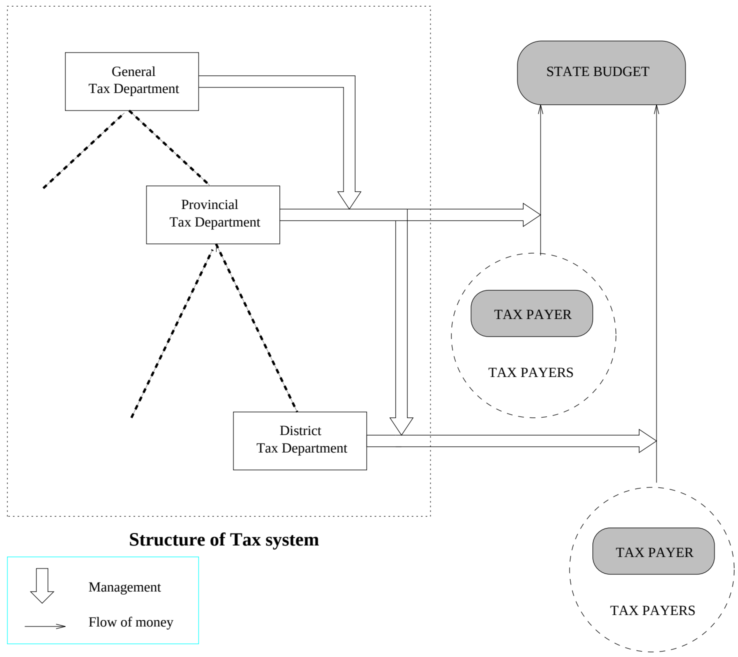


Figure 26: Tax system structure

$\forall$ l : Line, rp1, rp2 : Report •
  amount(l,summary(rp1, rp2)) $\equiv$ amount(l,rp1) + amount(l,rp2)
  pre form(rp1) = form(rp2)

The specification does not enumerate different kinds of reports. It provides a system in which a format can be given (e.g. arrears of profit tax in periods 1, 2 and 3) and a criteria for the lines (e.g. manufacturers of chemicals). The rest is automatic.

E.9.4 Security

It is important that taxation information is both confidential and accurate, and that the opportunities for corruption are minimised. Hence the data in the system needs to be protected against unauthorised disclosure, alteration or destruction. This is done currently by defining various groups of users and various access modes to the data.

Reflecting this in an applicative specification would involve considerable simple but tedious changes in passing the identities of users from the top level functions down to the database level where their rights would be checked. It would be more easily added later in the imperative version.

We also investigated the possibility of providing a "security shell" at the outermost level. But this would involve a careful analysis of what data was read or written by each top level function, and this uncovers some deeper problems. For example, a person collecting a summary report about chemical companies might not be allowed access to individual records. But in creating a summary report in which only totals for provinces, say, appear the report generating algorithm will need to read data for individual companies. Does this mean the person creating the report is "reading" this data? This needs more investigation.

E.9.5 Policy

There was also some investigation of possible changes in the taxation system. For example, as the Vietnamese economy develops there is likely to be an introduction of personal income tax instead of taxing only enterprises; tax rules devised to encourage particular industries may have to be changed to fit in with international rules and agreements; value added tax may be introduced. Hence the specification needs to be abstract enough to cover such possibilities.

E.9.6 Budget and treasury systems

The taxation system is only a part of the picture. The national budget system is involved in drawing up, propagating and monitoring budgets for all government departments (including taxation). The treasury system is involved in handling all money flows. For example, taxation payments are made to a treasury office and reported from there to the taxation system. Both the budget and treasury systems have the same hierarchical structure as the taxation system (with an additional commune level for the budget system). These were specified in a similar manner to the taxation system, but in less detail.

E.9.7 External aid and loans

External debt and loan systems only exist at the general, national level but have to deal with the foreign sources of the aid or loans (which may take the form of goods as well as money in various currencies). These are being specified, again in outline.

E.9.8 Integration

The five systems described above — taxation, budget, treasury, external aid and external loans — are each separate specifications. We need to find a way of integrating them while reusing as much as possible of the existing specifications. This involves a number of things:

- The hierarchical, nested specifications of the first three need to be "unpacked" so that it is possible, for example for the treasury office in a particular district to communicate amounts of taxes paid to the taxation office of the same district.
- The functional dependencies between the hierarchical layers need to be relaxed into some message passing mechanism. It is not the case that a taxation office in a province directly calls the report generation functions of its district offices in order to produce its summary report. Rather it must send messages to them and wait for replies. The specification has to be loose enough to allow for various and changing forms of actual communication, which will depend on the physical locations of offices and the equipment they may have. Similar communication mechanisms need to be described for communication "horizontally" between systems.
- The specification of the communication system needs to establish some common format of data between different systems.

It seems possible to do this by adding an extra component, a communication medium with a set of addresses for all the different systems and offices within them. This specification, while originally single and monolithic, could be decomposed later into a number of different systems reflecting the different mechanism in use between different offices. Then the existing modules of the different systems can be extended by functions to send, receive and act on messages. This work is just starting but seems feasible. It seems to be a general method for starting with tightly structured hierarchical systems and developing them smoothly into distributed, more loosely connected systems.

This project is described in more detail in a UNU/IIST research report [81].

E.10 EdMaCS: External Debt System

EdMaCS[67] is an acronym for External Debt Monitoring & Control System.

The EdMaCS Requirements destine it to become an Information and Workpiece Multi-Frame problem.

E.10.1 Introduction

This project arose from an exercise done as part of the course in Formal Software Development given by UNU/IIST at the Vietnamese Ministry of Finance (MoF) in January 1996. The exercise was to specify formally the description in [7] of the accounting principles to be used in recording and reporting on a country's external debt.

The exercise was intended to be both:

1. an exercise in using the formal specification language RAISE;
2. an example of how to use the writing of a formal description as a means of analysing requirements written in natural language (in this case English). The document being analysed is not specifically a set of requirements for a software system but a definition of "Accounting Principles", and as such could be expected at least to explain what data is to be recorded.

E.10.2 Accounts

Partly for pedagogic reasons, the exercise started by constructing a basic account, with functions to *open* an account, *deposit* and *withdraw* money, and *close* it.

First we define some basic types that we expect to use universally — *Money* and *Date*:

type
  Money = **Int**,
  Date
value
  $\leq$: Date $\times$ Date $\rightarrow$ **Bool**

[67] This section is written by Chris George.

Money is conveniently modelled as **Int** (since we will do arithmetic with it). **Real** is not necessary since we can use the lowest value actually recorded as the basic unit (e.g. cents rather than dollars, or even fractions of cents if we need to compute, say, daily amount of interest with sufficient accuracy). *Date* could be modelled as **Nat** (counting in days) but we choose to be more abstract. A more complete approach would include the axioms making $\leq$ a total order, which can be found as a standard specification in [51].

The basic meaning of "account" is related to "counting", to calculation. But it has also come to mean an explanation: when someone accounts for your money they explain what they did with it. Our model is of an account as a story — a sequence of events with a beginning (*open*), a middle, and (eventually) an end (*close*). We call events "transactions" and assign them "kinds". *open* and *close* are special kinds of transaction: an account must begin with an *open* and may contain a *close* only as its final event:

```
type
  Transaction ::
    date : Date
    kind : Transaction_kind
    amount : Money,
  Transaction_kind == open | close | _,
  Account = {| trl : Transaction* • is_account(trl) |}
value
  is_account : Transaction* → Bool
  is_account(trl) ≡
    trl ≠ ⟨⟩ ∧
    is_date_ordered(trl) ∧
    (∀ tr : Transaction, i : Nat •
        i ∈ inds trl ∧ tr = trl(i) ⇒
        ((kind(tr) = open) = (i = 1)) ∧
        (kind(tr) = close ⇒ i = len trl)),

  is_date_ordered : Transaction* → Bool
  is_date_ordered(trl) ≡ ...
```

A *Transaction* is a *record* with three components: a date, a kind, and an amount. *Transaction_kind* is a *variant* type: it contains two constants *open* and *close* but leaves it possible for more to be defined. An *Account* is a list of transactions satisfying the predicate *is_account*, i.e. it is a list of transactions which

- is not empty
- is ordered by date
- starts with an *open*

- if it has a close, finishes with it.

The full specification ACCOUNT defines the functions *open*, *deposit*, *withdraw* and *close*, and also a function *balance* to compute the balance on any date (using all transactions). There is a further function *balance_kind* to compute the total of transactions of a particular kind. This is not of obvious use at present but turns out later to be useful.

There are also functions *can_deposit*, *can_withdraw* and *can_close* that are used as preconditions for the corresponding functions *deposit* etc., but are not completely defined. This leaves it open to us to specialise accounts in various ways. Thus withdrawal might be limited by a minimum balance, or by an agreed maximum overdraft. Some savings accounts have maximum balances, giving a restriction on deposits. And it turns out that a zero balance is not necessarily a sufficient or even an appropriate condition for closure.

An example of a specialisation of ACCOUNT is INTEREST_ACCOUNT which can be used for accounts that attract interest. This is created by "extending" ACCOUNT: in object-oriented terminology we would say that INTEREST_ACCOUNT inherits ACCOUNT.

E.10.3 Loans

We model a *Loan* as a record with three components — a *schedule*, an *actual* account and other *info*rmation. The idea is that the *schedule* records the plan for the loan: when money will be made available and when capital and interest should be repaid. The *actual* account records all the relevant actual events:

```
type
  Loan ::
    schedule : Schedule
    actual : Account
    info : Loan_info,
  Schedule = {| s : Account • is_schedule(s) |},
  Loan_info
value
  is_schedule : Account → Bool
  is_schedule(sa) ≡
    (∀ tr : Transaction •
      tr ∈ elems tl sa ⇒
        case kind(tr) of
          commit → amount(tr) > 0,
          amortise → amount(tr) < 0,
          interest → amount(tr) < 0
        end)
```

Both *schedule* and *actual* are specialisations of ACCOUNT. For the schedule we allow only the kinds *commit*, *amortise* and *interest* to be included with *open* and *close*, and *is_schedule* requires that commitments are positive and both amortisation and interest payments are negative.

For *actual* we have the kinds *disburse*, *cancel*, *write_off*, *amortise* and *interest*.

Loan_info is left as a *sort*, an abstract type. It is just a place holder for other information that we may need about a loan.

Example This is what a balance sheet for a hypothetical loan might look like.

Date	Schedule		Actual		AoF	SoD	LCDO	Arrears	
	Kind	Amount	Kind	Amount				P	I
1	open	0	open	0	0	0	0	0	0
1	commit	100	disburse	70	30	70	100	0	0
2			cancel	−30	0	70	70	−30	0
100	amortise	−100			0	70	70	70	0
100	interest	−2			0	70	70	70	2
105			write-off	−20	0	50	50	50	2
110			amortise	−50	0	0	0	0	2
110			interest	−2	0	0	0	0	0
110			close	0	0	0	0	0	0

AoF is *availability of finance*, SoD is *stock of debt* and LCDO is *loan commitment/debt outstanding*. This represents our understanding of the requirements in [7].

The loan module only gives data for one loan. A debt management system would extend this further by having a collection of such loans, labelled in some way, with functions for collecting and reporting statistics on individual, all or categories of loans. This remains to be done.

E.11 RuBanCS: Financial Service Industry

RuBanCS[68] is an acronym for Russian Bank Computing System.

Varieties of RuBanCS requirements may predicate either Control, Information, Workiece, Connection or Clearing Transaction, including Multi-Frame problems.

E.11.1 The RuBanCS Project: UNU/IIST + ISP/RAS + FoRuBa

RuBanCS covers a UNU/IIST project joint with Institute of Systems Programming of the Russian Academy of Sciences (ISP/RAS) and the Federation of Russian Banks (FoRuBa), both in Moscow, the Russian Federation.[69] The RuBanCS project has as initial aims & objectives to build Domain Theory models of a general, not necessarily only Russian, finance industry and its services [37], so that such models can be used as the basis for vocational training at all levels: top, middle and low level management as well as "floor" (teller, back-room, etc.) staff. Subsequent aims & objectives is to Requirements Capture specific needs for computerized training, modeling and operations support for specific Russian Banks [78]. Final aims & objectives of the current RuBanCS project is to enable FoRuBa in collaboration with ISP/RAS to procure trustworthy financial service industry software — possibly through contracting Russian, own software development.

E.11.2 "What is a Bank?"

This is really the question we wish to answer[70]. The classical "bare-bones" of being a bank can be summarized: A bank is a financial services institution offering deposits subject to withdrawal on demand and making loans of a commercial or business nature. But as the next section points out: that was the classical view of a bank!

E.11.3 Overview: Institutions & Services

A financial industry consists, in "old" terms, of the following categories of institutions:

- Commercial (credit ([incl. savings &] loan) and demand deposit) Banks,
- Investment (underwriting initial public offerings [IPO]) Banks,

[68] This section is written by Dines Bjøner

[69] The EBRD (European Bank for Reconstruction & Development) and The World Bank are both very active in supporting the creation and proper operation of a Russian Finance Industry. Currently there are over 2,000 banks in the Russian Federation with Moscow being the home of at least 800 such new, fledgling banks!

[70] Cf. Section of same name pages 4–5 of [78].

- Insurance Companies (life, health, accident, theft, fraud, risk, etc.),
- Securities Brokerage (brokers & dealers) and Stock & Bond Exchanges,
- Clearing Houses (for checks, credit card slips, securities, etc.),
- Asset (Portfolio) Management Firms,
- Information & Advisory Companies,
- *&c.*

Increasingly the past, regulated boundaries between such institutions become blurred, with liberalization of the financial services market giving rise to the "bundling" of services and merger of different category institutions.

Their services span from:

1. currency exchange;
2. establishment and termination of accounts, deposits (of salaries, checks, insurance claims, etc.) into and withdrawal (including payments of loan interests and principal, regular utility invoices, insurance premia, credit card claims, etc.) from accounts, discounting commercial notes, safe-keeping of values, offering equipment leasing, etc. — as in commercial banking, including savings and (mortgage) loans, finance companies, etc.;
3. origination: advice & initial public offerings, underwriting, etc. — as in investment banking;
4. via establishment and termination of life, health, accident, theft, fraud, risk and other forms of insurance contracts, payment of premia, submission and processing of claims, and honouring claims, benefits and paying returns — as in the insurance industry;
5. to quoting, buying and selling securities ([preferred] stocks, [convertible] bonds, warrants, collaterised mortgage obligations, index options, futures, etc.) — as in the securities industry;
6. and the clearing of checks, credit card slips, securities and foreign exchange — as in the clearing transaction industry (cf. Connection Frame page 82).

Etc. for remaining services (portfolio management and information & advice).

States & Operations

Abstraction is "a must" if one is to clear one's way through this "wast jungle" of "packaged" services!

First we identify (idealized) notions of states for each of some of the industry categories:

E.11.4 Banking

The Domain Theory (but very condensed) narrative, which follows, closely "mirrors" the (equally abbreviated) formal model below. The reader is encouraged to "pair" the text with the formulas!

1. The State:

   A bank possesses resources, *Res*, maintains a current financial statement, *FiSt*, operates in time, T, maintains customer, U, contracts, *Ctrc*, which for various conditions, C, specify various values, *VAL*, depending on the [contracts & obligations] environment, *Env*, applicable at the time (i.e. in the appropriate time interval, *TI*).

   A bank further maintains, *Acts*, and as part of its state, Σ, accounts for each uniquely identified user (depositor or lender). Each such account records the balance history, the prevailing contract, and the prevailing set of composite conditions, *Cond*. These latter record a current [contracts & obligations] environment, the past such environments, and, as determined by the contract, a set of functions and predicates which are applied whenever a customer specific service action, A, is performed. The customer balance, *Bals*, records a time-stamped history of the account. For every time a customer service action, or list of actions, have been performed an entry is made into the balance (ledger). The entry records the current balance value, *VAL*, the action which caused the item in the entry list, and the report generated as a result of the action performed.

   Service actions may trigger additional service actions, to be performed "now, or later"! The bank state component Ψ records all outstanding customer actions together with the time (or time interval) at which they are to be performed.

   type Res, FiSt, TI, U, Ctrc, T, C, VAL

   $$
   \begin{array}{lcl}
   \text{Bank} & = & \Sigma \times \Psi \\
   \Sigma & = & \text{Acts} \times \text{Res} \times \text{FiSt} \times \text{Env} \\
   \Psi & = & \text{TI} \underset{m}{\to} \text{U} \underset{m}{\to} \text{A-}\mathbf{set} \\
   \text{Acts} & = & \text{U} \underset{m}{\to} (\text{Bals} \times \text{Ctrc} \times \text{Cond}) \\
   \text{Bals} & = & \text{T} \underset{m}{\to} (\text{VAL} \times \text{A} \times \text{Rept})^* \\
   \text{Cond} & = & \text{Env} \times \text{Envs} \times \text{FoPs} \\
   \text{Envs} & = & \text{TI} \underset{m}{\to} \text{Env} \\
   \text{Env} & = & \text{C} \underset{m}{\to} \text{VAL} \\
   \text{FoPs} & = & \text{FPn} \underset{m}{\to} (\text{Env} \overset{\sim}{\to} (\text{Bals} \overset{\sim}{\to} (\text{VAL}|\mathbf{Bool}))) \\
   \text{A} & = & \text{Bank} \overset{\sim}{\to} (\text{FPn-}\mathbf{set} \overset{\sim}{\to} \text{Bank})
   \end{array}
   $$

2. The Service Actions:

   Examples of service actions that may trigger further actions are: (i) Establishment of a loan contract and withdrawal (i.e. actually taking up) the loan (in full or in parts) are service actions. The latter trigger future actions which serve to remind the borrower to pay interest and principal and (later) to check that so has been done, etcetera. (ii) Establishment of a savings or demand deposit account, as well as deposits and withdrawals from the account are service actions. The former may trigger an action to regularly inform the depositor of the current account balance, whereas the latter may lead to overdrafts, say above a credit ceiling, in which case a reminder (to secure balance within limits) action must be issued. Etc.

We may illustrate how action triggering determines basic bank operations. The (sometimes non-deterministic) functions Nxt etc. as shown below, "cycles" the banking state through a first time interval's set of immediate actions as well as those triggered actions that were to be executed immediately!

value

$$
\begin{array}{ll}
\text{Nxt:} & \text{Bank} \xrightarrow{\sim} (\Psi \times \Psi) \\
\text{Sel\_TI:} & \Psi \xrightarrow{\sim} \text{TI} \\
\text{Sel\_U:} & (\text{U} \underset{m}{\Rightarrow} \textbf{A-set}) \xrightarrow{\sim} \text{U} \\
\text{I\_0:} & (\text{U} \underset{m}{\Rightarrow} \textbf{A-set}) \xrightarrow{\sim} \text{Bank} \xrightarrow{\sim} \text{Bank} \\
\text{I\_1:} & \textbf{A-set} \xrightarrow{\sim} \text{Bank} \xrightarrow{\sim} \text{Bank} \\
\text{I\_2:} & \text{A} \xrightarrow{\sim} \text{Bank} \xrightarrow{\sim} \text{Bank} \\
\text{Merge:} & \Psi \times \Psi \xrightarrow{\sim} \Psi \\
\text{V:} & \text{A} \xrightarrow{\sim} \text{Bank} \xrightarrow{\sim} \text{Bank}
\end{array}
$$

Nxt(σ,ψ) $\equiv$
  if ψ = [ ]
    then (σ,[ ])
    else let ti = Sel_TI(ψ) **in**
        I_0(ψ(ti))(σ,$\psi\backslash${ti})
  end **end**

Sel_TI(ψ) **as** (t,t$'$)
  pre (t,t$'$) $\in$ **dom** ψ
  post $\sim\exists$ (t$''$,t$'''$):TI •
        (t$''$,t$'''$) $\in$ **dom** ψ
        $\wedge$ (t$''$ < t) $\vee$ ((t$''$=t) $\wedge$ (t$'''$<t$'$))

I_0(actm)(b)(ti) $\equiv$
  if actm = [ ]
    then b
    else let u = Select_U(actm) **in**
        I_0(actm$\backslash${u})(I_1(actm(u))(b))
        end
  end

Sel_U(uactm) **as** u
  pre uactm $\neq$ [ ]
  post u $\in$ **dom** uactm

I_1(acts)(σ,ψ) $\equiv$

$$
\begin{array}{l}
\quad \textbf{if } \mathrm{acts} = [\,] \\
\qquad \textbf{then } (\sigma,\psi) \\
\qquad \textbf{else let } \mathrm{act} \in \mathrm{acts} \textbf{ in} \\
\qquad\qquad \mathrm{I\_1}(\mathrm{acts}\backslash\{\mathrm{act}\})(\mathrm{I\_2}(\mathrm{act})(\sigma,\psi)) \\
\qquad\qquad \textbf{end} \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{I\_2}(\mathrm{act})(\sigma,\psi) \equiv \\
\quad \textbf{let } (\sigma',\Delta\psi) = \mathrm{V}(\mathrm{act})(\sigma,\psi) \textbf{ in} \\
\quad (\sigma',\mathrm{Merge}(\psi,\Delta\psi)) \textbf{ end}
\end{array}
$$

$$\mathrm{Merge}(\psi,\Delta\psi) \equiv \ldots$$

$$\mathrm{V}(\mathrm{act})(\sigma,\psi) \equiv \ldots$$

The most "interesting" function is, of course, V, which specify the effect of executing an action! Since we have not detailed the set of service commands and how they relate to actions we shall not detail V. But essentially we must first clear up our mind as to the overall action scheme before we can proceed to the smaller details. Then they become almost trivial!

E.11.5 Securities Services

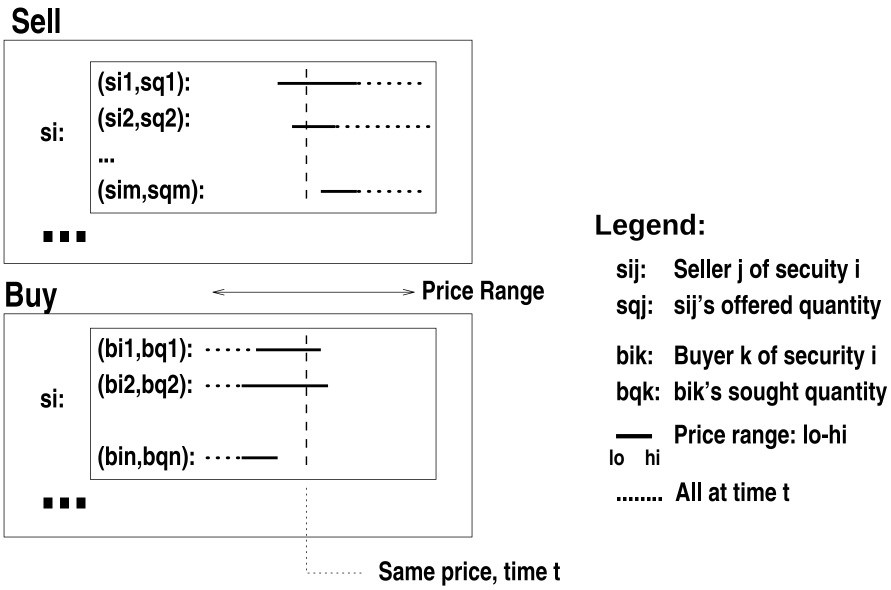
The securities industry consists of client (buyers and sellers), traders (brokers and dealers), and the market. The market can be understood as the willingness to buy, sell and trade securities for example via the exchanges. Cf. item 5 page 175.

1. The State:

   Usually buying and selling is on behalf of clients who provide access to funds through bank accounts, who may safe keep securities in their bank, etc. Hence we need include the banks in the securities state. The traders each have their own state (Υ) which we do not further elaborate. Traders are either brokers (who buy and sell at market conditions and as otherwise instructed by clients) or dealers (who provide quotes to potential buyer or seller client, and who buy or sell at quoted prices). The market is here abstracted as the pair of the current securities (stock, etc.) prices asked and bid, as well as the past history and future expectations of such prices. Banks and traders are uniquely identified, and clients provide reference U to their money or securities safe guard accounts. Asked, quoted and bid prices fall within a spread.

   $$
   \begin{array}{l}
   \textbf{type } \Upsilon, \mathrm{BId}, \mathrm{TId}, \mathrm{SId}, \mathrm{U}, \mathrm{Qty}, \mathrm{Price} \\
   \quad \Omega \qquad\quad = \mathrm{Banks} \times \mathrm{Traders} \times \mathrm{Market} \\
   \quad \mathrm{Banks} \quad\; = \mathrm{BId} \xrightarrow[m]{} \mathrm{Bank}
   \end{array}
   $$

Figure 27: Asks/Bids Spread



$$
\begin{array}{lcl}
\text{Traders} & = & \text{TId} \xrightarrow[m]{} \Upsilon \\
\text{Market} & = & \text{AskBid} \times (\text{T} \stackrel{\sim}{\to} \text{AskBid}) \\
\text{AskBid} & = & \text{Ask} \times \text{Bid} \\
\text{Ask} & = & \text{SId} \xrightarrow[m]{} (\text{Qty} \xrightarrow[m]{} (\text{Slr} \times \text{Sprd})) \\
\text{Bid} & = & \text{SId} \xrightarrow[m]{} (\text{Qty} \xrightarrow[m]{} (\text{Byr} \times \text{Sprd})) \\
\text{Slr,Byr} & = & \text{FId} \times \text{U} \times \text{U} \\
\text{Sprd} & :: & \text{lo:Price} \;\; \text{hi:Price} \\
\text{FId} & = & \text{BId} \mid \text{TId}
\end{array}
$$

The market is worth looking at. At any moment several buyers and several sellers are bidding, respectively asking for the purchase, respectively sale of various securities at various price spreads, Usually buyers expect purchases within lower spreads than sale expectations of sellers. Quotes and composite purchase/sale transactions are based, respectively effected on the base on the composite "picture" as represented by current, past and expected future prices. Figure 27 attempts to illustrate this for just a single security and at a specific (for example the current) moment!

2. The Service Designators:

   Request for quotes, and orders or offers of purchases and sales, are abstracted in the form of Service Action commands:

$$
\begin{array}{lcl}
\text{SAct} & = & \text{BD} \times \text{SCmd} \\
\text{BD} & = & \text{Fid} \times \text{U} \\
\text{SCmd} & = & \text{BSel} \mid \text{BBuy} \mid \text{DQuo} \mid \text{DBuy} \mid \text{DSel}
\end{array}
$$

Title /* Paper or Encrypted Copy of Title */
Order = SId × Qty × TPri
TPri == nil | Sprd

BBuy = Byr × Order
BSel = Slr × Order × Title

DQuo = Opt × Order /* Usually TPri is nil */
DBuy = Byr × Order /* Usually TPri is nil */
DSel = Slr × Order × Title /* Usually TPri is nil */
Opt == nil | Byr | Slr

Initial capital letters B and D refers to brokers, respectively dealers. Security name (for ex. IBM) by SId, etc.

3. The Service Actions:

We just sketch the signature of the broker and dealer buying and selling services, and of the dealer quotes. The buy and deal services involve the actual transfer of title to bought, respectively sold securities.

value

E_BBuy: BBuy $\stackrel{\sim}{\to}$ Ω $\stackrel{\sim}{\to}$ (Ω × T × Price × Title)
E_BSel: Bsel $\stackrel{\sim}{\to}$ Ω $\stackrel{\sim}{\to}$ (Ω × T × Price)
E_DQuo: DQuo $\stackrel{\sim}{\to}$ Ω $\stackrel{\sim}{\to}$ (Price × T)
E_DBuy: DBuy $\stackrel{\sim}{\to}$ Ω $\stackrel{\sim}{\to}$ (Ω × T × Price × Title)
E_DSel: DSel $\stackrel{\sim}{\to}$ Ω $\stackrel{\sim}{\to}$ (Ω × T × Price)

"The rest is almost trivial!" For anybody who has dealt with semantic definitions of imperative programming languages, to complete the above is almost a mere matter of hard work. The "thinking" aspect comes in when describing exactly which security offerings (asks) and requests (bids) to select for exchange. Here we resort to a pre/post style of definition since that "function" is highly heuristic. Formalizing is easy: it is algorithmitizing certain operations that is impossible.

In other words: we may succeed, as we claim we have, in presenting (an even computationally) precise model of the state of securities systems — whether dealing with [preferred] stocks, [convertible] bonds, warrants, collaterised mortgage obligations, index options, futures — but to expect a computable model for the act of choosing the moment for actual purchases and sales is probably not "in"!

E.11.6 Transaction Clearing

The clearing transaction frame presented in special appendix section C.4.4's item C.4.4 explains most of what we now wish to cover. The transactions referred to now has to do with the usually batched clearing of check, credit card slips, and securities.

1. The State:

   Pairs of (external) financial institutions — that of the buyer (sender), respectively seller (receiver) of the financial instruments (checks, credit card slip, etc.) — usually have fixed have agreements as to who (the internal, clearer party) *Clears* the bought instruments (*CreDeb*).

   type BId, TId
   $\Phi \quad = \Theta \times \mathrm{Clears}$
   $\Theta \quad = \mathrm{FId} \underset{m}{\rightarrow} (\mathrm{Bank}|\Upsilon)$
   $\mathrm{Clears} \quad = (\mathrm{FId} \times \mathrm{FId}) \underset{m}{\rightarrow} \mathrm{ClrIns}$
   $\mathrm{ClrIns} \quad = (\mathrm{T} \underset{m}{\rightarrow} \mathrm{CreDeb}) \times ...$
   CreDeb /* Credit + Debit Slips, Checks, Securities, ... */

2. The Service Designators:

   Although the two external parties to usually send, accordingly receive batches of financial instruments to, respectively from the clearer, abstractly speaking we can view the clearing of financial instruments individually. Each is therefore seen as a single instrument TransAction which names the sender, the receiver and the specific instrument (as part of the *TCmd* part of the sent, respectively received message!

   type U, TCmd
   $\mathrm{TAct} \quad = (\mathrm{FId} \times \mathrm{U}) \times \mathrm{TCmd} \times (\mathrm{FId} \times \mathrm{U})$

3. The Service Actions:

   Please refer to Figure 14 on (page 82).

   The three three below Valuation functions describe pertinent actions of the sender, clearer, and receiver.

   The Elaboration function describes the overall clearing transaction.

   value
   $\mathrm{V\_r}\colon \mathrm{TCmd} \overset{\sim}{\rightarrow} (\Phi \overset{\sim}{\rightarrow} (\mathrm{T} \overset{\sim}{\rightarrow} (\Theta \times \mathrm{T} \times \mathrm{Rept})))$
   $\mathrm{V\_c}\colon \mathrm{TCmd} \overset{\sim}{\rightarrow} (\mathrm{Rept} \overset{\sim}{\rightarrow} (\mathrm{ClrIns} \overset{\sim}{\rightarrow} (\mathrm{T} \overset{\sim}{\rightarrow} (\mathrm{ClrIns} \times \mathrm{T} \times \mathrm{Rept}))))$
   $\mathrm{V\_i}\colon \mathrm{TCmd} \overset{\sim}{\rightarrow} (\mathrm{Rept} \overset{\sim}{\rightarrow} (\Phi \overset{\sim}{\rightarrow} ((\mathrm{T}\times\mathrm{T}) \overset{\sim}{\rightarrow} (\Theta \times \mathrm{T} \times \mathrm{Rept}))))$

$$\text{E: TCmd} \xrightarrow{\sim} (\Phi \xrightarrow{\sim} (\Phi \times \text{Rept}))$$

E(tc)(o,cs)(t) $\equiv$
0. **let** ((fi,_,_),_,(fi′,_,_)) = tc **in**
1. **let** (by,t′,r) = V_r(tc)(o(fi))(t) **in**
2. **let** (ci′,t″,r′) = V_c(tc)(r)(cs(fi,fi′))(t′) **in**
3. **let** (by′,t‴,r″) = V_i(tc)(r′)(o(fi′))(t′,t″) **in**
4. ((o + [fi↦by,fi′↦by′],ci′),t‴,r^r′^r″)
end end end end
pre {fi,fi′} $\subseteq$ **dom** o $\wedge$ ...

Lines 1. and 2. describe the preparation for sending and the local clearer processing, respectively. Together they describe the sent message in terms of the 'parameter transfer': t′,r,tc. The pair of lines 2. and 3. similarly describe the received message in terms of the 'parameter transfer': t′,t″,r′,tc. Line 3. further describes the reply through values t‴, r″. Finally line 4. expresses the update of states in respective institutions: fi, fi′ and c by the local valuations as well as further, overall results of the entire clearing process, namely tits "closing" time t‴, and the composite of the three individual valuation reports r,r′,r″.

E.11.7 Discussion

All of the above models describe a Domain Theory for individual financial institutions, and only for their day-to-day operations.

Other models, not shown, model their combination into "larger-and-larger" financial service institutions, as well as, again in separate models, and again not shown, the Domain Theory aspects of decisions of strategic, tactics and operations management.

These *decision theories* identify the indicators whose fluctuations determine (i) interest rates and other account conditions, (ii) risk taking or avoiding actions, (iii) portfolio management decisions, etc., etc. Value generation [37] is not done by just operating the financial institutions, but by carefully monitoring the behaviour of the market and adjusting marketing and sales, services and contract conditions as well as internal operations and fees accordingly.

E.12 DSSfSD: Software for Sustainable Development

DSSfSD[71] is an acronym for Decision Support Systems for Sustainable Development.

For a brief synopsis we refer to item 12 page 19 (section 3).

UNU/IIST is administratively planning (incl. seeking partners and funds for) and technically preparing (i.e. exploring development approaches of the DSSfSD project. Presently the DSSfSD concept includes reliance on (free or paid) access to a global collection of public and private geo- (GIS) and demographic (DIS), as well as other (more conventional) information systems.

As such DSSfSD is an Information, Workpiece & Clearing Transaction Multi-Frame problem.

E.12.1 Towards A Domain Theory

Development

Development, D, is about changing one set of resources, R, into another set:

type R

value
  D: $R^* \xrightarrow{\sim} R^*$

Examples of resources could be agricultural and urban land, monies, labour, equipment, time, etc. An example of development could take agricultural zone land and "develop" it into urban zone land: with streets, sewage system, etc. Before development some amount of monies were available (credit). After development that amount has gone considerably down. It may even be zero or negative (debit)!

Resources, Attributes and Indicators

With resources we associate attributes, A, such as various mineral types, water, currency, credit, debit, etc. Each attribute defines a range of values, V. For a given resource, for example agricultural land, the water attribute value may range between 0% (desert) and 100% per surface unit of land (flood)! Each attribute, when associated with a particular instance of a resource, also defines a desirable indicator, I, range of values. For rice production land the water attribute

[71] This section is written by Dines Bjørner

indicate may for example be a time dependent function which for the planting season may be new 90%, and which nearer to harvest decreases, say towards 15%. To every specific resource, R, we can thus associate a set of a attributes, and for each of these we can associate a pair of actual and desirable indicators:

$$
\begin{array}{l}
\textbf{type } \mathrm{R, A, V, I} \\
\quad \mathrm{AII} = \mathrm{A} \underset{m}{\rightarrow} (\mathrm{I} \times \mathrm{I}) \\
\quad \mathrm{RAII} = \mathrm{R} \underset{m}{\rightarrow} \mathrm{AII}
\end{array}
$$

With a resource we can associate its "origin", O, and one resource may be said to be "basically the same" as another resource if they both have the same "origin":

$$
\begin{array}{l}
\textbf{type } \mathrm{O} \\
\\
\textbf{value} \\
\quad \mathrm{origin: R \rightarrow O} \\
\quad \mathrm{same\_origin: R \times R \rightarrow \textbf{Bool}} \\
\quad \mathrm{same\_origin(r,r') \equiv origin(r) = origin(r')}
\end{array}
$$

One can characterize analysis functions which determine resource attributes and their current values:

$$
\begin{array}{l}
\textbf{value} \\
\quad \mathrm{obs\_As: R \rightarrow A\textbf{-set}} \\
\quad \mathrm{obs\_V: R \times A \stackrel{\sim}{\rightarrow} V}
\end{array}
$$

We can think of characterizing modeling functions which, in the context of the resource's environment, ENV, and desired objectives, Ω, help determine reasonable estimates for desirable indicators:

$$
\begin{array}{l}
\textbf{type } \mathrm{ENV}, \Omega \\
\textbf{value} \\
\quad \mathrm{model: R \stackrel{\sim}{\rightarrow} (ENV \stackrel{\sim}{\rightarrow} (\Omega \stackrel{\sim}{\rightarrow} AII))}
\end{array}
$$

We leave the definition of concrete types for ENV and Ω to a more detailed treatment of the current problem.

We can also think of analysis which checks whether the current (at time T) values of attributes lie within actual or desirable indicators:

```
value
   in_range: V × I × T → Bool

   analyze_a: R × RAII × T → Bool
   analyze_a(r,raii,t) ≡
      let aii = raii(r) in
      ∀ a:A • a ∈ dom aii
         let (ai,_) = aii(a),
               v = obs_V(r,a) in
         in_range(v,ai,t)
      end end

   analyze_a: R × RAII × T → Bool
```

Etcetera for analyze_d.

Equities and Sustainable Development

By an equity, E, we understand a, for example fuzzy or Boolean, time dependent relation between a pair of same origin resources, for example an undeveloped and a developed resource. We cover only the Boolean case.

```
type
   TI = {| (b,e) | (b,e):T×T • b≤e |}
   E = R × R × AII × TI ~→ Bool
```

To determine whether such a relation is within an acceptable, but fuzzy measure, every equity relation refers to appropriate resource attribute indicator pairs.

Sustainable development is now development which keeps changed resource values within desired range:

```
value
   SD = RAII × E × TI ~→ R-set

axiom
   ∀ sd:SD
      ∀ raii:RAII, e:E, ti:TI • (raii,e,ti) ∈ dom sd ⇒
         let rs = sd(raii) in
         ∀ r:R, r′:R •
```

$$
\begin{array}{l}
\quad\quad \mathrm{r} \in \mathbf{dom}\ \mathrm{raii} \wedge \mathrm{r}' \in \mathrm{rs} \wedge \mathrm{same\_origin(r,r')} \Rightarrow \\
\quad\quad \mathbf{let}\ \mathrm{aii} = \mathrm{raii(r)}\ \mathbf{in}\ \mathrm{e(r,r',aii,ti)} \\
\quad \mathbf{end\ end}
\end{array}
$$

Discussion

The reader should have observed that we deal with "real" resources, not representations of them. Our analysis and modeling functions and relations are therefore to be thought of as those performed in the biological etc. laboratories on samples of — i.e. "standing in" for — the real resources.

Such is the "nature" of Domain Analysis: "deal" with the "real" thing, albeit by modeling it! So it really is not the real things but a possibly not computable "representation" of it. That is: some of the suggested analysis and modeling functions and predicates may not be computable!

E.12.2 Towards Requirements

Requirements now has to "project" part or all of the above modeled domain onto a computable representation.

Resource Representation

Instead of resources we speak of resource representations *RR*. And, when speaking of such we must provide somehow a "mapping" between resource representations and some "variant", *VR*, of the real resource. A variant, if the resource is a physical one, could be a combination of (perhaps satellite) photos of the resource, its location, a map thereof, deeds to it, etc. Given a real resource, a variant resource, and a resource representation, predicates are therefore assumed to exist which decides whether a variant resource "stand for" the real thing, whether a resource representation and a variant designate the same resource, etc. Functions between real resources etc. are not computable as we cannot represent the real thing inside the computer.

$$
\begin{array}{l}
\mathbf{type}\ \mathrm{R,\ VR,\ RR} \\
\quad \mathrm{same\_R\_VR}: \mathrm{R} \times \mathrm{VR} \rightarrow \mathbf{Bool} \\
\quad \mathrm{same\_R\_RR}: \mathrm{R} \times \mathrm{RR} \rightarrow \mathbf{Bool} \\
\quad \mathrm{same\_VR\_RR}: \mathrm{VR} \times \mathrm{RR} \rightarrow \mathbf{Bool}
\end{array}
$$

A requirement may now be to implement *VR, RR* and *same_VR_RR,* as well as (direct and inverse (i) database) catalogues of these:

type
RCT = VR $\overrightarrow{m}$ RR-**set**
iRCT = RR $\overrightarrow{m}$ VR

Other requirements may be to support catalogues of attributes and indicators of resource representations (i.e. computerized versions of *RAII* etc.) as well as various various analysis and modeling functions,

type
RAIIC = RR $\overrightarrow{m}$ (A $\overrightarrow{m}$ (I $\times$ I))

And so forth.

The Required System

After further requirements capture we can summarize a system state that deals with some or all of the computable facets of equities and sustainable development.

Discussion

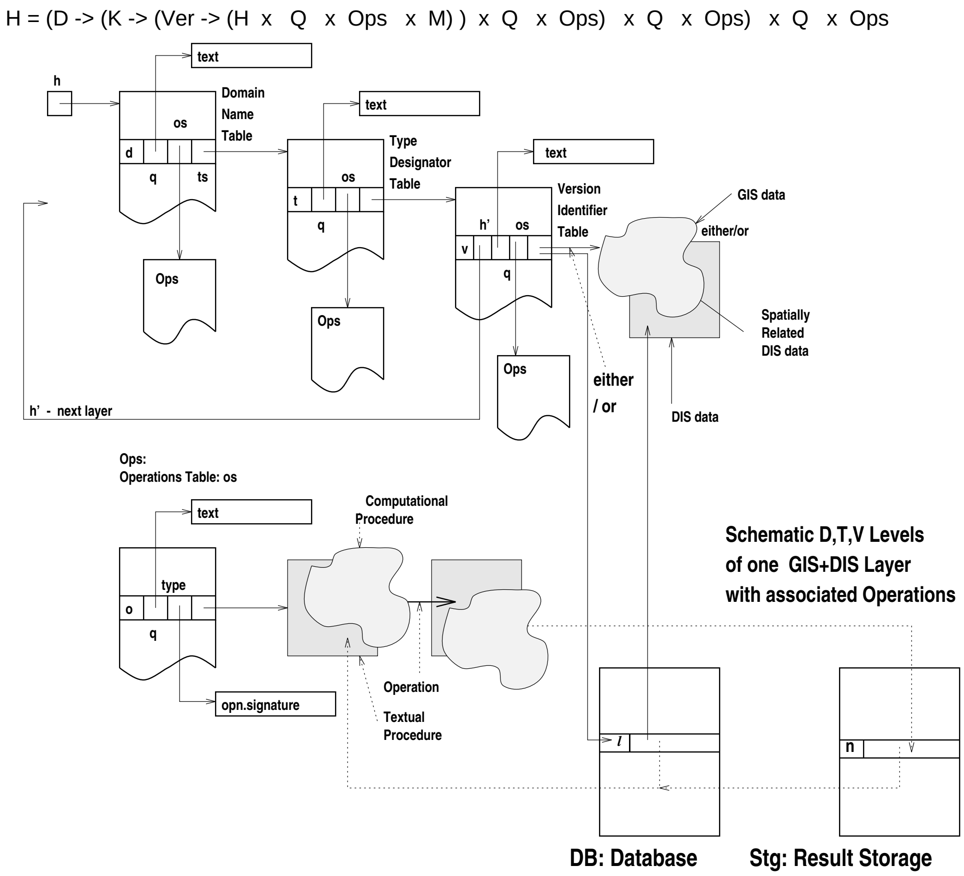
The problem is: we have been very abstract about the "real" representations of resources. In this example we assume that decisions concerning that are part of Software Design, not Requirements Capture — see item 6b comment in section 5.1.

E.12.3 Towards Software

Federated GIS + DIS *&c.* System

A major step towards the Software Architecture design is therefore — in following up the observation of the immediate subsection (section E.12.2) — we may therefore suggest the following *Federated GIS + DIS* &c. architecture component.

A picture is worth a thousand words.



A Hierarchical GIS+DIS Federated Information System

The picture is intended to indicate how compositions of several $VR + RR$'s can be "stored" in a hierarchical structure. This structure allows access, through — for example geographical — 'domain' names D[72], composite information 'kinds' K[73] and 'versions' Ver[74], to GIS maps, or DIS tables, etc. M. For each level of 'domains' there may typically be 'sub-domains'[75]; hence recursion through H. What is indicated as pointers could be Internet links. In this way the GIS+DIS federation allows the connection of world wide information. For each level of 'domains', 'kinds', and 'versions' we indicate access to unary or n–ary functions Ops that can be applied, by users, to single or composite data M as selected through navigation (cf. stack-top $Stack$ top DTV of the formal model below).

But the picture above is insufficient and may be misleading. So we formalize the structures shown — thereby generalizing and abstracting it.

[72] For GIS, or even DIS data examples 'domain' names could be: Continent names.

[73] For GIS data 'kind' could be: geotechnic, geodetic, hydrologic, surface minerals, cadestral, cartographic, etc.; while for DIS data example could be: export, import, crime, income, tax, health, etc.

[74] 1990, 1995, ...

[75] For example Europe, with European countries being sub-domains, and their provinces or states being further sub-sub-domains, etc.

A set of formulas is worth an infinity of pictures!

type

$$
\begin{array}{l}
\text{D, K, Ver, Q, Res\_Typ, Res\_VAL, VAL, N, Data} \\
\text{S} = \text{H} \times \text{DB} \times \text{Stg} \times \text{Stack} \\
\text{Stack} = \text{DTV}^* \\
\\
\text{H} = (\text{D} \underset{m}{\rightarrow} \\
\qquad\quad (\text{K} \underset{m}{\rightarrow} \\
\qquad\qquad (\text{Ver} \underset{m}{\rightarrow} \\
\qquad\qquad\quad (\text{M} \times \text{H} \times \text{O} \times \text{Q})) \\
\qquad\qquad \times \text{Ops} \times \text{Q}) \\
\qquad\quad \times \text{Ops} \times \text{Q}) \\
\quad \times \text{Ops} \times \text{Q} \\
\\
\text{M} = \text{L} \times \text{F} \\
\text{F} = \text{Data} \xrightarrow{\sim} \text{Data} \\
\text{DB} = \text{L} \underset{m}{\rightarrow} \text{Data} \\
\text{Stg} = \text{N} \underset{m}{\rightarrow} (\text{Res\_VAL} \times \text{Stack} \times \text{DTV} \times \text{On}) \\
\\
\text{DTV} = \text{D} \mid \text{D}{\times}\text{T} \mid \text{D}{\times}\text{T}{\times}\text{V} \\
\text{OTup} = ((\text{A}^* \times \text{K}^*) \times \text{Res\_Typ}) \\
\text{OFct} = (((\text{VAL}^* \times \text{Data}^*)|\text{Q}) \xrightarrow{\sim} \text{Res\_VAL}) \\
\text{Ops} = \text{On} \underset{m}{\rightarrow} \text{OTyp} \times \text{OFct} \times \text{Q}
\end{array}
$$

We show this only to indicate to the reader the direction Software Design may be taking after Requirements Capture.

E.12.4 Discussion

We have gone into some detail on a possible Domain Theory for Decision Support Systems for Sustainable Development, and we have briefly hinted at issues of related Requirements Capture & Definition and of Software Design.

The DSSfSD project is an attempt to help clear up vague notions of *decision support systems* and of federated information systems, including those of *heterogeneous databases.*

E.13 ReMaCS: Resource Monitoring & Control System

ReMaCS[76] is an acronym for Resource Monitoring & Control System.

ReMaCS requirements predicate it as primarily an Information & Workpiece Multi-Frame problem.

ReMaCS views this subject only from the point of view of resources: Strategies for their conversion, including acquisition and "down-sizing"; tactics for their allocation and scheduling; and operational handling: i.e. their deployment. In the below Domain Theory exposition we abstract away from types of resources.

E.13.1 Allocation & Scheduling

By a resource $r : R$ we mean such things as monies, material, equipment, labour, etc. — but we single time, $t : T$, out as a special, separately treated resource.

A set of resources $rs : RS$ can be allocated, $ra : RA$, to a set of locations, $l : L$, and such allocations can be scheduled, $ras : RAS$, to be available: either free for use or occupied during certain time intervals, $ti : TI$:

type r:R, t:T, l:L
  rs:RS = R-**set**
  ls:LS = L_set
  ti:TI = $\{|$ (b,e) | (b,e):T×T • b≤e $|\}$
  ra:RA = RS $\overrightarrow{m}$ LS
  ras:RAS = TI $\overrightarrow{m}$ (RS $\overrightarrow{m}$ LS)

E.13.2 Production

To produce certain products, $p : P$, say a machining industry manufactured mechanical part, or an airline business passenger-flight service, requires a production task consisting of many actions, $a : A$ which can be thought of as organized according to some graph, $g : G$. Edges $e : E$ into, $i : I$, out from, $o : O$, and between, $b : B$, nodes designate, α, flow actions of semi-produced products of various kinds (types) and in fixed quantities, q :**Nat**, $k : K$. Nodes $n : N$ designate, α, transformation actions on goods. At any time production of one particular product can be pictured by the state, $\sigma : \Sigma$ of the production singling out a set of labeled nodes and edges.

type p:P, a:A, e:E, n:N, k:K

[76] This section is written by Dines Bjørner

$$
\begin{array}{l}
\mathrm{g{:}G} = \mathrm{i{:}(N} \xrightarrow[m]{} \mathrm{(E} \xrightarrow[m]{} \mathrm{FLOW))} \\
\qquad \times\ \mathrm{b{:}(N} \xrightarrow[m]{} \mathrm{(N} \xrightarrow[m]{} \mathrm{(E} \xrightarrow[m]{} \mathrm{FLOW)))} \\
\qquad \times\ \mathrm{o{:}(N} \xrightarrow[m]{} \mathrm{(E} \xrightarrow[m]{} \mathrm{FLOW))} \\
\qquad \times\ \alpha\mathrm{{:}((N|E)} \xrightarrow[m]{} \mathrm{A)} \\
\mathrm{FLOW} = \mathrm{K} \xrightarrow[m]{} \mathbf{Nat} \\
\mathrm{gs{:}GS{:}\ P} \xrightarrow[m]{} \mathrm{G} \\
\\
\sigma{:}\Sigma = \mathrm{(N|E)}\textbf{-set} \\
\psi{:}\Psi = \Sigma\textbf{-set} \\
\\
\mathrm{Trace} = \mathrm{T} \rightarrow \Sigma \\
\mathrm{Traces} = \mathrm{T} \rightarrow (\mathrm{P} \xrightarrow[m]{} \Psi)
\end{array}
$$

$gs : GS$ denotes the (fixed) set of production plans for all products. *trace:Trace* (indirectly) designate the time dependent set of flow and transformation actions for the production of one product. *traces:Traces* a similar thing for all product productions.

E.13.3 Order Processing

Transformation and flow actions require and consume possibly mobile resources that must be allocated and scheduled according to production plans, $\pi : \Pi$, as determined by existing or expected orders, $\omega : \Omega$, from customers, $c : C$, of specific products, in specific quantities, and delivered within specific time intervals, and given the current state of the overall production system, including availability of resources. $on : On$ designate customer order numbers.

type On

$$
\begin{array}{l}
\omega{:}\Omega = (\mathrm{C} \xrightarrow[m]{} (\mathrm{On} \xrightarrow[m]{} (\mathrm{TI} \xrightarrow[m]{} (\mathrm{P} \xrightarrow[m]{} (\mathbf{Nat} \times \mathrm{L}))))) \\
\pi{:}\Pi = \mathrm{RAS} \times \mathrm{GS} \times \mathrm{Traces}
\end{array}
$$

E.13.4 Performance & Estimates

An enterprise can record its past performance wrt. past customer orders and their delivery, and can estimate their future expectations. Together these statistics form a basis, together with all available, i.e. current resources, for strategic management.

type

$$
\begin{array}{l}
\mathrm{PP,\ FE} = (\mathrm{TI} \xrightarrow[m]{} (\mathrm{P} \xrightarrow[m]{} \mathbf{Nat})) \\
\mathrm{STA} = \mathrm{PF} \times \mathrm{FE}
\end{array}
$$

Statistics results from projections from (abstractions of) orders ($\omega : \Omega$).

E.13.5 Strategics

Strategic management must regularly assess needs for acquiring additional resources, disposing of existing such, or converting present resources to other kinds. Given current resource allocations and schedules (*ras:RAS*) and order statistics (*sta:STA*) a set of alternative resource allocations and schedules can be ascertained (i.e. strategically planned (sp)): namely such which will enable the enterprise to meet customer orders.

value
sp: $\mathrm{RAS} \times \mathrm{STA} \stackrel{\sim}{\to} \mathrm{RAS}\text{-}\mathbf{set}$

Strategic management now has to go to the market — which is composed from like and other *E*enterprises — to negotiate changes to its resource requirements (as implied rather explicitly by some of the *ras* output results (wrt. the current *ras* input).[77] Out of such negotiations the enterprise selects one (the future) resource plan (an *ras* in *RAS*).

value
neg: $(\mathrm{B} \times \mathrm{RAS}\text{-}\mathbf{set}) \times (\mathrm{B} \underset{m}{\to} \mathrm{RAS}\text{-}\mathbf{set}) \stackrel{\sim}{\to} \mathrm{RAS}$

Then statistics and resource plans may have to be revised in the light of the negotiated resource plan which may be different from any of the ones first selected through strategic planning.

value
ree: $\mathrm{RAS} \times \mathrm{STA} \stackrel{\sim}{\to} (\mathrm{STA} \times \mathrm{RAS})$

E.13.6 Tactics

Whereas strategic resource plans [location] allocated resources to physical locations, tactical planning (tp) is concerned with [task] allocating such physically located resources to specific production tasks.

[77]Example changes may be: (i) more working capital, in which case either negotiations for bank loans, or for new stock issues, etc. may have to be made; (ii) landing & departure rights at certain airports may have to be negotiated, etc.

type
  GRAS = (TI $\overrightarrow{m}$ (On $\overrightarrow{m}$ (P $\overrightarrow{m}$ ((N|E) $\overrightarrow{m}$ RAS))))
value
  tp: STA $\times$ RAS $\stackrel{\sim}{\rightarrow}$ GRAS

E.13.7 Operations

Finally operations apply production actions to resources and yield traces:

value
  op: Ω $\times$ GRAS $\stackrel{\sim}{\rightarrow}$ Traces

E.13.8 Discussion

Algorithmically Computable vs. Heuristics "Functions"

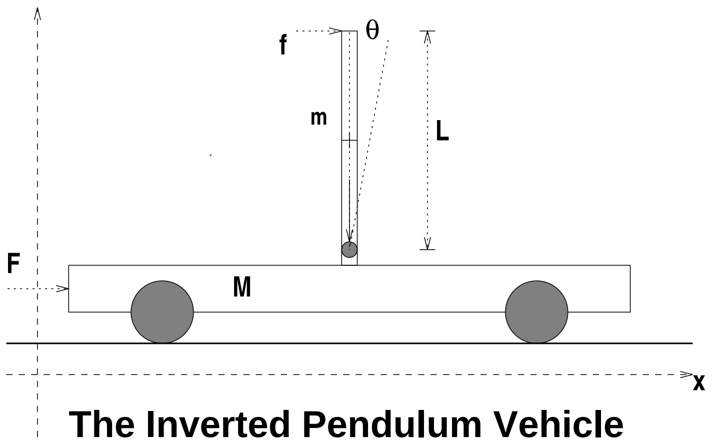
The functions sp, neg, ree, tp, and op may not all be computable! The strategic planning (sp) function, for example, can most likely not be automated as strategic planning contains much heuristics. So what have we achieved, if only to give the signature (abstract data and function types)? Well, already that is something! Now, at least, we seem to have a strong "inkling" of what goes into also the heuristics activities of especially top level management, and we can start supporting its work by providing computerized support for gathering, vetting and maintaining the data on resources, allocations, schedules, production descriptions, past performance, future estimates, and the various, alternative and resulting resource plans. And we have a much better grasp now on how it all hangs together! Furthermore, the latter functions are indeed computable, certainly op. And it is important that it receives well "nurtured" (up-to-date, reliable) data input. There are also strong attempts, cf. Papodopolous[78] at University of Toronto, at providing *knowledge-engineered* support for the more heuristic functions mentioned above.

Requirements Capture and Software Design

From the Domain Theory sketched above one is now in a much better situation, we claim, to capture requirements for overall management support software, and hence for its trustworthy design.

[78] Dines: Check name etc.

Figure 28: The Inverted Pendulum Vehicle



F A Development Analogy: The Inverted Pendulum

F.1 The Analogy

The following example shows, by analogy, what we mean by the "tripartite" division of concern into domain theory, requirements and software design.

It is seemingly a control theory & engineering case story!

- Synopsis:

  The example is that of an inverted pendulum mounted on a flatbed car that can move along a straight rail track. Figure 28 show the relevant components:

  Assume an initial state in which the vehicle is stationary and the pendulum upright and no forces are applied either to the vehicle or the pendulum. The vehicle has mass M, the pendulum mass m, then height of the pendulum is L.

  Different behaviours of the inverted pendulum vehicle can be described:

  - By moving the flat bed truck to the left (that is: by applying a force F as shown) [or to the right] the pendulum will fall to the right [respectively left, and vice versa].
  - Or: by applying a force f to the pendulum, in direction towards left [or right] in the initial state the pendulum will fall to the left [respectively left].
  - By a combined application of forces F and f as shown the pendulum movement can be "controlled" with respect to the situation when only one of the forces is applied.
  - *&c.*

At any time the pendulum deviates from the vertical (upright) position by an angle θ.

No other physical constraints or conditions are present. That is: We assume no track, wheel or revolute joint friction or air resistance, etc. That is: We assume otherwise idealised conditions — without this in any way diminishing the relevance of what we wish to illustrate!

- Domain Theory & Analysis:

  The domain theory for the 'inverted pendulum' example can be presented in two steps:

  1. Newton's Laws of Gravity and Dynamics — which are normally assumed.
  2. A set of sets of differential equations describing movement of vehicle and pendulum in response to each of a variety of forces being applied: momentaneously or discretely or continuously over time.

     That is: These sets of equations describe all possible (however idealised) behaviours of the inverted pendulum and its vehicle.

     What makes such a set of sets of equations correct is its adherence to Newton's Laws.

     Let us give one such set of equations,[79] namely for the force F being applied instantaneously — as shown — to the vehicle. x denotes the position of the revolute joint (and hence of the ["centre"] of the vehicle), and θ the momentary angle of the pendulum. Single and double dotted x's and θ's denote the speed, respectively the acceleration of the vehicle, respectively the angular movement of the pendulum.

$$\begin{aligned}
(M+m)*\ddot{x} - \frac{m*L}{2}*\cos\theta*\ddot{\theta} - \frac{m*L}{2}*\sin\theta*\dot{\theta}^2 &= F+f && (1)\\
\frac{2*m*L}{3}*\ddot{\theta} + m*L*\cos\theta*\ddot{x} &= m*g*L*\sin\theta + 2*f*L*\cos\theta && (2)
\end{aligned}$$

- Requirements Capture & Definition:

  Now there may be different sets of requirements expressed with respect to desired behaviours of the inverted pendulum vehicle:

  1. Assuming relative small momentary forces F secure that θ stays within a largest allowable small deviation ϵ.
  2. Assuming relative small momentary forces f secure that θ stays within a largest allowable small deviation ϵ.
  3. Assuming relative small momentary forces F secure that θ does not change faster than a certain maximum angular speed.

[79] All formulas shown in this section were kindly developed by Mr. Gao JianPing, UNU/IIST Fellow, seconded from Beijing University of Aero & Astronautics, China

4. Assuming relative small momentary forces f secure that θ does not change faster than a certain maximum angular speed.
5. Assuming relative small continuous forces F assumed to change direction over time secure that θ stays within a largest allowable small deviation ϵ and that the vehicle stays within maximum deviations to the left and right from $x = 0$.
6. *&c.*

We will focus on the following requirement:

– Assume a relative small, continuous, positive and negative (i.e. both push and pull direction) control force F — such as shown in figure 29 on the facing page to the left. Secure that θ stays within a largest allowable small deviation ϵ after some initial time period t_r.

Thus one and the same problem may have many different requirements — and more will arise as shown below!

But let us first show the formal expressions for the above informally stated requirement:

$$\text{Given } \epsilon \text{ and } t_r\text{:} \quad \forall t \geq t_r \bullet \quad \theta \leq \epsilon \tag{3}$$

- Control Design:

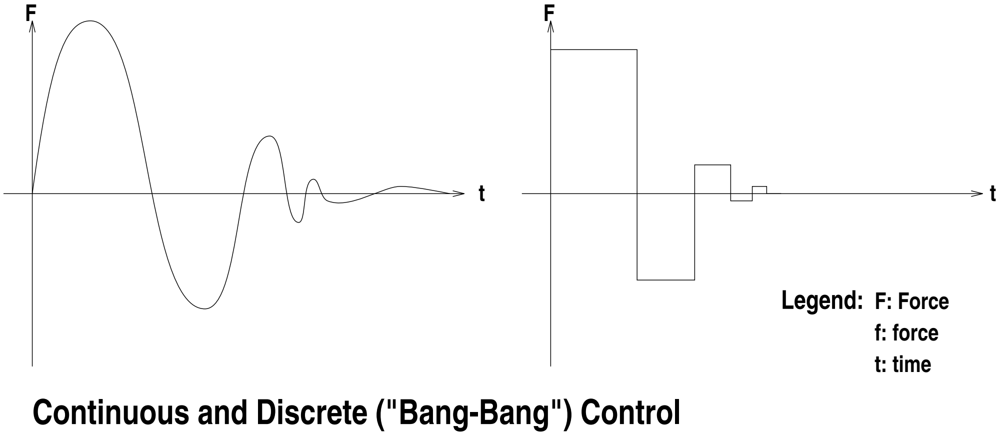
  Finally we can specify different ways of solving the problem of activating the controllable forces:

  1. Continuous Control:
     Under various assumptions controlling the vehicle may be done by a continuously acting force F of the kind illustrated in figure 29:
  2. Bang-Bang Control:
     Under various assumptions controlling the vehicle may alternatively be done by a discretely acting force F of the kind illustrated in figure 29.
  3. Adaptive Control
  4. Stochastic Control

  Below we shown just one solution to the design problem, that of the continuous control of managing the vehicle subject to a continuous application of a force F:

$$F = k_1 * \theta + k_2 * \dot{\theta} + k_3 * x; k_1 > 0, k_2 > 0, k_3 > 0 \tag{4}$$

Figure 29: Inverted Pendulum Control Architectures



where:

– $(k_1, k_2, k_3) = (g_1(m, M, L, \epsilon, t_r), g_2(m, M, L, \epsilon, t_r), g_3(m, M, L, \epsilon, t_r))$

That is: k_1, k_2, k_3 depend on the constants m, M, L, ϵ, t_r and can be selected by simulation to ensure that the requirement can be satisfied.

We have illustrated — from a regime seemingly other than software (and digital computing) — the distinct, non-overlapping, but strongly related notions of Domain Theory, Requirements Definition and Software Architecture, i.e. Software Design.

We apparently also say that the example application could be thought of as one being implemented by digital computing, in software. Indeed this may be the case — as illustrated in UNU/IIST Report [48]. So our example is, in fact, an example *"right smack in the middle"* of software development!

F.2 Some Domain Analysis Calculations

The inertia moment of inverted pendulum according to its mass center is $J = \frac{m*L^2}{12}$.

Refer to the coordinates of inverted pendulum mass center as (x_0, y_0). f_x and f_y are the projections of the (action/reaction) force working on the (friction free) revolute joint along the x, respectively y axes.

From Newton's law we have:

$$F - f_x = M\ddot{x} \tag{5}$$

We also have:

$$m\ddot{x}_0 = f_x + f \tag{6}$$

$$m\ddot{y}_0 = f_y - m * g \tag{7}$$

$$J * \ddot{\theta} = \frac{L}{2} * f_y * \sin\theta - \frac{L}{2} * (f_x - f) * \cos\theta \tag{8}$$

and following kinematic relationships:

$$\begin{cases} x_0 = x + \frac{L}{2} * \sin\theta \\ y_0 = \frac{L}{2} * \cos\theta \end{cases} \tag{9}$$

Therefore:

$$\begin{cases} \ddot{x}_0 = \ddot{x} + \frac{L}{2} * (\cos\theta * \ddot{\theta} - \sin\theta * \dot{\theta}^2) \\ \ddot{y}_0 = -\frac{L}{2} * (\sin\theta * \ddot{\theta} + \cos\theta * \ddot{\theta}^2) \end{cases} \tag{10}$$

From equations (6) and (7), we get:

$$\begin{cases} f_x + f = m * (\ddot{x} + \frac{L}{2} * (\cos\theta * \ddot{\theta} - \sin\theta * \dot{\theta}^2)) \\ f_y = -\frac{m*L}{2} * (\sin\theta * \ddot{\theta} + \cos\theta * \ddot{\theta}^2) + m * g \end{cases} \tag{11}$$

Substituting equations (11) into (5 on the preceding page) and (8), we get:

$$\begin{cases} M\ddot{x} = F - m * (\ddot{x} + \frac{L}{2} * (\cos\theta * \ddot{\theta} - \sin\theta * \dot{\theta}^2)) + f \\ J\ddot{\theta} = -\frac{m*L}{2} * (\sin\theta * \ddot{\theta} + \\ \qquad \cos\theta * \dot{\theta}^2) * \sin\theta * \frac{L}{2} + \frac{L}{2} * m * g * \sin\theta - \frac{m*L}{2} * (\ddot{x} + \\ \qquad \frac{L}{2} * (\cos\theta * \ddot{\theta} - \sin\theta * \dot{\theta}^2)) * \cos\theta + f * L * \cos\theta \end{cases} \tag{12}$$

From this we derive the domain equations as shown earlier:

$$\begin{cases} (M + m) * \ddot{x} - \frac{m*L}{2} * \cos\theta * \ddot{\theta} - \frac{m*L}{2} * \sin\theta * \dot{\theta}^2 = F + f \\ \frac{2*m*L}{3} * \ddot{\theta} + m * L * \cos\theta * \ddot{x} = m * g * L * \sin\theta + 2 * f * L * \cos\theta \end{cases} \tag{13}$$

G UNU/IIST Reports & Notes 1992–1997

G.1 UNU/IIST Reports

References

[1] Dines Bjørner. UNU/IIST Programme. Administrative Report, 1, UNU/IIST, P.O.Box 3058, Macau, February 19 1993. Presents the co-ordinates along, and principles according to which UNU/IIST will select and pursue its Programmatic Activities the next couple of years.

[2] Dines Bjørner. 1992 Annual Report. Administrative Report 2, UNU/IIST, P.O.Box 3058, Macau, January 1993. Reports on the status of UNU/IIST and its activities after the first 6 months of operation.

[3] Zhou Chaochen (ed.). Duration Calculus. Compendium, 3, UNU/IIST, P.O.Box 3058, Macau, March 1993.

(a) Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. ProCoS Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, June 1991, Revised June, 1992. Published in: *Information Processing Letters,* 40, 5, pp. 269-276, 1991.

(b) Liu Zhiming, A.P. Ravn, E.V. Sørensen, Zhou Chaochen: A Probabilistic Duration Calculus, *presented in the 2nd Intl. Workshop on Responsive Computer Systems, Saitama, Japan,* Oct. 1-2, 1992, published in H. Kopetz and Y. Kakuda (eds), *Dependable Computing and Fault-Tolerant Systems Vol. 7 : Responsive Computer Systems*, pp 30-52. Springer-Verlag, 1993.

(c) M.R. Hansen and Zhou Chaochen. Semantics and Completeness of Duration Calculus,. ProCoS Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, 1992. Published in: J.W. de Bakker, C. Huizing, W.-P. de Roever, G. Rozenberg, (Eds) *Real-Time: Theory in Practice,* REX Workshop, LNCS 600, pp 209-225, 1992.

(d) J.U. Skakkebæk and A.P. Ravn, H. Rischel, and Zhou Chaochen. Specification of Embedded Real-time Systems. ProCoS Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, June 1992. Published in: *Proceedings 4th Euromicro Workshop on Real-Time Systems,* IEEE Press, pp 116-121, June 1992.

(e) Zhou Chaochen, M.R. Hansen, A.P. Ravn, and H. Rischel. Duration Specifications for Shared Processors. Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, October 1991. Published in: *Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Niijmegen, January 1992, LNCS 571, pp 21-32, 1992.

(f) M.R. Hansen, Zhou Chaochen, and J. Staunstrup. A Real-time Duration Semantics for Circuits. Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, March 1992. Published in: *Proceedings of the Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Princeton, March 1992.

(g) Zhou Chaochen, A.P. Ravn, and M.R. Hansen. An Extended Duration Calculus for Hybrid Real-time Systems. ProCoS Research Report, ID/DTH, Bldg. 344, DK-2800 Lyngby, Denmark, October 1992. (Revised version available as UNU/IIST Report 9.) Published in *Hybrid Systems*, LNCS 736, Springer Verlag, 1993.

(h) Liu Zhiming, A.P. Ravn, E.V. Sørensen, Zhou Chaochen. Towards a Calculus of Systems Dependability, Presented at *Workshop on Theory of Hybrid Systems*, Lyngby, Denmark, 19-20 October 1992. Published in the Journal of High Integrity Systems, Vol.1, No.1,1994, pp 49–65.

(i) Zhou Chaochen, M.R. Hansen, P. Sestoft: Decidability and Undecidability Results for Duration Calculus, *Proc. of STACS'93, 10th Symposium on Theoretical Aspects of Computer Science*, Würzburg, LNCS 665, pp 58-68, Feb. 1993.

Distributed only with publishers permission.

[4] Dines Bjørner (ed.). Issues and Techniques in Rigorous Development of Software, Technology Transfer and Education — Selected 1992 Papers. Compendium, 4, UNU/IIST, P.O.Box 3058, Macau, March 1993. This is a compendium of 4 papers:

(a) Dines Bjørner. Trustworthy Computing Systems: The ProCoS Experience, February, 1992. Published in: *14th International Conf. on Software Engineering,* Melbourne, Australia, May 11-15, 1992, pp. 15–34.

(b) Dines Bjørner and Jørgen Fischer Nilsson. Algorithmic & Knowledge Based Methods — Do they "Unify"? Published in: *International Conf. on Fifth Generation Computer Systems: FGCS'92,* Tokyo, June 1–5, 1992 (separate folder, pp. 191–198).

(c) Dines Bjørner. From Research to Practice: Self-reliance of the Developing World through Software Technology — Usage, Education & Training. Published in: IFIP Transaction, IFIP World Congress '92, Madrid, September 7–11, 1992, pp. 65–71.

(d) Dines Bjørner. A Rôle for UNU/IIST: Developing Countries' Access to New Information Technologies. Presented at: *Access to Science and Technology, The Rôle of Information Technology*, Kyoto University and UNU Symp., Kyoto, May 12–14, 1992; Expected publ. by UNU Press.

Distributed only with publishers permission.

[5] Zhou Chaochen and Li Xiaoshan. A Mean Value Duration Calculus. Research Report 5, UNU/IIST, P.O.Box 3058, Macau, March 1993. Published as Chapter 25 in *A Classical Mind*, Festschrift for C.A.R. Hoare, Prentice-Hall International, 1994, pp 432–451. .

[6] Dines Bjørner. Formal Models of Robots: Geometry & Kinematics. Research Report 6, UNU/IIST, P.O.Box 3058, Macau, March 15 1993. For abbreviated version see Chapter 3 in *A Classical Mind*, Festschrift for C.A.R. Hoare, Prentice-Hall International, Publ., 1994, pp 37–58.

[7] Dines Bjørner. University Curricula in Software Technology. Technical Report 7, UNU/IIST, P.O.Box 3058, Macau, March 15 1993. Keynote address: IFIP TC3 WG3.4/SRIG-ET (SEARCC) International Working Conference 1993: *Software Engineering Education*, Hong Kong, September 28 – October 2, 1993.

[8] Oliver Corff. Mongolian Language Support — a Mongolian Language Environment for IBM-compatible PCs. Research Report 8, Institute of Language and Literature, Ulaanbaatar, Mongolia, Fall 1992. UNU/IIST has kindly been permitted to print and distribute this Research report.

[9] Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An Extended Duration Calculus for Real-time Systems. Research Report 9, UNU/IIST, P.O.Box 3058, Macau, January 1993. Published in: *Hybrid Systems*, LNCS 736, 1993. .

[10] Zhou Chaochen. Duration Calculi: An Overview. Research Report 10, UNU/IIST, P.O.Box 3058, Macau, June 1993. Published in: *Formal Methods in Programming and Their Applications,* Conference Proceedings, June 28 – July 2, 1993, Novosibirsk, Russia; (Eds.: D. Bjørner, M. Broy and I. Pottosin) LNCS 736, Springer-Verlag, 1993, pp 36–59. .

[11] Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. Linear Duration Invariants. Research Report 11, UNU/IIST, P.O.Box 3058, Macau, July 1993. Published in: Formal Techniques in Real-Time and Fault-Tolerant systems, LNCS 863, 1994. .

[12] Dines Bjørner. Prospects for a Viable Software Industry — Enterprise Models, Design Calculi, and Reusable Modules. Technical Report 12, UNU/IIST, P.O.Box 3058, Macau, 7 November 1993. Appendix by Søren Prehn and Dong Yulin,
Published in *Proceedings from first ACM Japan Chapter Conference*, March 7–9, 1994: World Scientific Publ., Singapore, 1994.

[13] Zhou Chaochen and Michael R. Hansen. Lecture Notes on Logical Foundations for the Duration Calculus. Lecture Notes, 13, UNU/IIST, P.O.Box 3058, Macau, August 1993.

[14] Dines Bjørner. Accreditation, Licensing and Certification; Curricula, Engineers and Software. Technical Report 14, UNU/IIST, P.O.Box 3058, Macau, 7 November 1993 1993, Revised 8 December 1993. Presented at the *UNIDO/COGIT Meeting*, Vienna, Austria, November 23, 1993.

[15] Michael R. Hansen, Pandya Paritosh, and Zhou Chaochen. Finite Divergence. Research Report 15, UNU/IIST, P.O.Box 3058, Macau, 10 November 1993. Published in *Theoretical Computer Science* Vol.138, pp 113-139, 1995. .

[16] Zheng Yuhua and Zhou Chaochen. A Formal Proof of a Deadline Driven Scheduler. Research Report 16, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994. Published in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 863, 1994, pp. 756–775. .

[17] Dines Bjørner, Jan Goossenaerts, and Søren Prehn. Enterprise Modelling. Technical Report 17, UNU/IIST, P.O.Box 3058, Macau, June 1994.

[18] Dines Bjørner. Annual Report: 1993. Administrative Report 18, UNU/IIST, P.O.Box 3058, Macau, 1. January 1994.

[19] Yu Xinyao, Wang Ji, Zhou Chaochen, and Paritosh K. Pandya. Specification of an Adaptive Control System. Research Report 19, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994. Published in: *Formal Techniques in Real-Time and Fault-Tolerant systems*, LNCS 863, 1994, pp. 738–755. .

[20] Wang Ji, Yu Xinyao, and Zhou Chaochen. Hybrid Refinement. Research Report 20, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994. .

[21] Yu Huiqun, Paritosh K. Pandya, and Sun Yongqiang. A Calculus of sampled data systems. Research Report 21, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994. Published in: *Formal Techniques in Real-Time and Fault-Tolerant systems*, LNCS 863, 1994, pp. 716–737.

[22] Dines Bjørner. Annual Report, 1994. Research Report 22, UNU/IIST, P.O.Box 3058, Macau, 1. January 1995. (Pre-version: 9 November 1994).

[23] Dines Bjørner, Dong Yu Lin, and Søren Prehn. Domain Analyses: A Case Study of Station Management. Research Report 23, UNU/IIST, P.O.Box 3058, Macau, 9 November 1994. Presented at the *1994 Kunming International CASE Symposium*: KICS'94, Yunnan Province, P.R.of China, 16–20 November 1994. .

[24] Zhou Chaochen and Yu Huiqun. A duration Model for Railway scheduling. Technical Report 24b, UNU/IIST, P.O.Box 3058, Macau, May 1994. .

[25] Dang Van Hung and Zhou Chaochen. Probabilistic Duration calculus for Continuous Time. Research Report 25, UNU/IIST, P.O.Box 3058, Macau, May 1994. Presented at *NSL'94 (Workshop on Non-standard Logics and Logical Aspects of Computer Science, Kanazawa, Japan, December 5–8, 1994)*, submitted to *Formal Aspects of Computing*. .

[26] Chen Zongji, Wang Ji, and Zhou Chaochen. An Abstraction of Hybrid Control Systems. Research Report 26, UNU/IIST, P.O.Box 3058, Macau, June 1994. .

[27] Chen Zongji, Wang Ji, and Zhou Chaochen. A Design Approach of Hybrid Control Systems. Research Report 27, UNU/IIST, P.O.Box 3058, Macau, June 1994. .

[28] Yu Xinyiao et al. Stability of Railway Systems. Technical Report 28, UNU/IIST, P.O.Box 3058, Macau, May 1994.

[29] Yu Xinyiao et al. On Stability of Hybrid Systems. Technical Report 29, UNU/IIST, P.O.Box 3058, Macau, May 1994.

[30] Jan Goossenaerts and Dines Bjørner. An Information Technology Framework for Lean/Agile Supply-based Industries in Developing Countries. Technical Report 30, UNU/IIST, P.O.Box 3058, Macau, 1994. Published in *Proceedings of the International Dedicated Conference on Lean/Agile Manufacturing in the Automotive Industries*, ISATA, London, UK.

[31] Jan Goossenaerts and Dines Bjørner. Interflow Systems for Manufacturing: Concepts and a Construction. Technical Report 31, UNU/IIST, P.O.Box 3058, Macau, 1994. Published in *Proceedings of the European Workshop on Integrated Manufacturing Systems Engineering*.

[32] Jan Goossenaerts. Generic Models for Manufacturing Industry. Technical Report 32, UNU/IIST, P.O.Box 3058, Macau, November 1994. Draft Version. In: Reference Handbook Ch. IV.1.

[33] Dang Van Hung. An Algorithm for Maintaining Consistent View of Processes in Distributed Systems. Research Report 33, UNU/IIST, P.O.Box 3058, Macau, January 1994. Published in the *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, December 14–16, 1994, Kanazawa, Japan, IEEE Computer Society Press, 1994, pp. 33–40. .

[34] He Weidong and Zhou Chaochen. A Case Study of Optimization. Research Report 34, UNU/IIST, P.O.Box 3058, Macau, December 1994. Published in *The Computer Journal*, Vol. 38, No. 9, pp. 734-746, 1995. .

[35] Dang Van Hung and Wang Ji. On The Design of Hybrid Control Systems Using Automata Model. Research Report 35, UNU/IIST, P.O.Box 3058, Macau, November 1994. Published in V. Chandru and V. Vinay (Eds.) *Foundations of Software Technology and Theoretical Computer Science* (FST&TCS16), LNCS 1180, Springer-Verlag, Dec 1996, pp. 156–167. .

[36] Belawati H. Widjaja, He Weidong, Chen Zongji, and Zhou Chaochen. A Cooperative Design for Hybrid Systems. Research Report 36, UNU/IIST, P.O.Box 3058, Macau, December 1994. Published in: Logic and Software Engineering International Workshop in Honor of Chih-Sung Tang, pp 127-150, Edited by A. Pnueli and H. Lin World Scientific, 1996. .

[37] Dines Bjøner, Chris George, and Søren Prehn. Domain Analysis — a Prerequisite for Requirements Capture. Technical Report 37, UNU/IIST, P.O.Box 3058, Macau, February 1995. .

[38] Wang Juan, Li Xiaoshan, and Zhou Chaochen. A Duration Calculus Approach to Specifying the Steam-boiler Problem. Technical Report 38, UNU/IIST, P.O.Box 3058, Macau, February 1995. .

[39] Wang Juan, Li Xiaoshan, and Zhou Chaochen. Specifying Optimal Design of the Steam-boiler System. Technical Report 39, UNU/IIST, P.O.Box 3058, Macau, February 1995. .

[40] Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A Duration Calculus with Infinite Intervals. Research Report 40, UNU/IIST, P.O.Box 3058, Macau, February 1995. Published in: *Fundamentals of Computation Theory*, Horst Reichel (ed.), pp 16-41, LNCS 965, Springer-Verlag, 1995. .

[41] Xu Qiwen and He Weidong. Hierarchical Design of a Chemical Concentration Control System. Research Report 41, UNU/IIST, P.O.Box 3058, Macau, April 1995. .

[42] Chris George. Distributed Train Rescheduling. Research Report 42, UNU/IIST, P.O.Box 3058, Macau, April 1995. .

[43] Fu Hongguang, Yang Lu, and Zhou Chaochen. A Geometric Approach to Solving Inverse Kinematics. Research Report 43, UNU/IIST, P.O.Box 3058, Macau, March 1995. .

[44] Philip Chan and Dang Van Hung. Duration Calculus Specification of Scheduling for Tasks with Shared Resources. Research Report 44, UNU/IIST, P.O.Box 3058, Macau, June 1995. Published in: Kanchana Kanchanasut and Jean-Jacques Levy (Eds.), *Algorithms, Concurrency and Knowledge*, LNCS 1023, Springer-Verlag 1995, pp. 365–380. .

[45] Mia Indrika. Formal development of A Toll Way Control System. Technical Report 45, UNU/IIST, P.O.Box 3058, Macau, August 1995. .

[46] Dines Bjørner. New Software Technology Development. Technical Report 46, UNU/IIST, P.O.Box 3058, Macau, November 1995. International Symposium: New IT for Governance and Publication Administration, Beijing, China; organized by UNDDSMS, June 1996.

[47] Dines Bjørner. Software Support for Infrastructure Systems. Technical Report 47, UNU/IIST, P.O.Box 3058, Macau, November 1995. Resource position statement for the *First Malaysia Information Technology Days: 1–3 November 1995.* .

[48] Dines Bjørner. Software Systems Engineering — From Domain Analysis to Requirements Capture [— an Air Traffic Control Example]. Technical Report 48, UNU/IIST, P.O.Box 3058, Macau, November 1995. Keynote paper for the *Asia Pacific Software Engineering Conference*, APSEC'95, Brisbane, Australia, 6–9 December 1995. .

[49] Xu Qi Wen. On Compositionality in Refining Concurrent Systems. Research Report 49, UNU/IIST, P.O.Box 3058, Jan 1996. .

[50] Dang Van Hung and Phan Hong Giang. A Sampling Semantics of Duration Calculus. Research Report 50, UNU/IIST, P.O.Box 3058, Macau, November 1995. Published in: *Formal Techniques for Real-Time and Fault Tolerant Systems*, Bengt Jonsson and Joachim Parrow (Eds), LNCS 1135, Spriger-Verlag, pp. 188–207, 1996. .

[51] Chris George. A Theory of Distributing Train Rescheduling. Research Report 51, UNU/IIST, P.O.Box 3058, Macau, December 1995. Published in: Marie-Claude Gaudel and James Woodcock (eds.), *FME'96: Industrial Benefit and Advances in Formal Methods*, LNCS 1051, Springer-Verlag, 1996, pp. 499–517. .

[52] Dines Bjørner, Chris George, and Søren Prehn. Scheduling and Rescheduling of Trains. Research Report 52, UNU/IIST, P.O.Box 3058, Macau, December 1995. .

[53] Dines Bjørner. 1995 Annual Report. Administrative Report 53, UNU/IIST, P.O.Box 3058, Macau, January 1996. Reports on the status of UNU/IIST and its activities during 1995 .

[54] Dang Van Hung and Ko Kwang Il. Verification via Digitized Model of Real-Time Systems. Research Report 54, UNU/IIST, P.O.Box 3058, Macau, February 1996. Published in the proceedings of *Asia-Pacific Software Engineering Conference 1996* (APSEC'96), IEEE Computer Society Press, 1996, pp 4–15. .

[55] Tomasz Janowski. On Bisimulation, Fault-Monotonicity and Provable Fault-Tolerance. Research Report 55, UNU/IIST, P.O.Box 3058, Macau, Dec 1995. .

[56] Wang Ji and He Weidong. Formal Specification of Stability in Hybrid Control Systems. Research Report 56, UNU/IIST, P.O.Box 3058, Macau, Dec 1995.

[57] Zhou Chaochen, Wang Ji, and Anders P. Ravn. A Formal Description of Hybrid Systems. Research Report 57, UNU/IIST, P.O.Box 3058, Macau, Dec 1995. Published in *Hybrid Systems III,* R. Alur, T. Henzinger, and E. Sontag (Editors), LNCS 1066, pp. 511-530, Springer-Verlag, 1996. .

[58] Dines Bjørner. Infrastructure Software Systems. Technical Report 58, UNU/IIST, P.O.Box 3058, Macau, Dec 1995. Presentation solicited for the Academia Europae (AE/CWI/SMC) Symposium, Amsterdam 11–12 April, 1996. .

[59] Dines Bjørner. New Software Development. Administrative/Technical Report 59, UNU/IIST, P.O.Box 3058, Macau, January 1996.
Special *Theme* paper: *New Software Technology Development.* Paper was first prepared in September 1995 for an International Symposium: *New IT Applications for Governance and Public Administration*, convened by the UN's Department for Development Support and Management Service: UNDDSMS, Beijing, November 9–14, 1995. This symposium was subsequently moved (tentatively) to June 1996, same venue .

[60] Dines Bjørner. Models of Enterprise Management: Strategy, Tactics & Operations — Case Study Applied to Airlines and Manufacturing. Technical Report 60, UNU/IIST, P.O.Box 3058, Macau, January – April 1996. .

[61] Dines Bjørner and Mohamed Rais. Federated GIS+DIS–based Decision Support Systems for Sustainable Development — a Conceptual Architecture. Research Report 61, UNU/IIST, P.O.Box 3058, Macau, March 1996. Draft.

[62] Cleta Milagros Acebedo. An Informal Domain Analysis for Manufacturing Enterprises. Research Report 62, UNU/IIST, P.O.Box 3058, Macau, March 1996.

[63] Tomasz Janowski. Domain Analysis for Manufacturing: Formalization of the Market. Research Report 63, UNU/IIST, P.O.Box 3058, Macau, March 1996. .

[64] Cleta Milagros Acebedo and Erwin Paguio. Manufacturing Enterprise Simulation: A Business Game. Research Report 64, UNU/IIST, P.O.Box 3058, Macau, March 1996.

[65] Tomasz Janowski. Categories for System Integration - A Proposal. Administrative 65, UNU/IIST, P.O.Box 3058, Macau, Mar 1996. .

[66] Tomasz Janowski and Mathai Joseph. Dynamic Scheduling in the Presence of Faults: Specification and Verification. Research Report 66, UNU/IIST, P.O.Box 3058, Macau, Mar 1996. .

[67] Roderick Durmiendo and Chris George. Formal Development of a Digital Mutiplexed Radio-Telephone System. Research Report 67, UNU/IIST, P.O.Box 3058, Macau, Feb 1996.

[68] Jan A. Bergstra and Cornelis A. Middelburg. Process Algebra Semantics of φSDL. Research Report 68, UNU/IIST, P.O.Box 3058, Macau, April 1996. .

[69] T. Janowski and C.M. Acebedo. Virtual Enterprise: On Refinement Towards an ODP Architecture. Research Report 69, UNU/IIST, P.O.Box 3058, Macau, May 1996. .

[70] Li Xuan Dong and Dang Van Hung. Checking Linear Duration invariants by Linear Programming. Research Report 70, UNU/IIST, P.O.Box 3058, Macau, May 1996. Published in Joxan Jaffar and Roland H. C. yap (Eds.), *Concurrency and Palalellism, Programming, Networking, and Securiry* LNCS 1179, Springer-Verlag, Dec 1996, pp. 321–332. .

[71] Ngo Quoc Tao. A Formal Specification of Realm – Spatial Data Type. Research Report 71, UNU/IIST, P.O.Box 3058, Macau, April 1996.

[72] Xu Qiwen. Semantics and Verification of the Extended Phase Transition Systems in the Duration Calculus. Research Report 72, UNU/IIST, P.O.Box 3058, Macau, June 1996. .

[73] Xu Qiwen and Yang Zengyu. Derivation of Control Programs: a Heating System. Research Report 73, UNU/IIST, P.O.Box 3058, Macau, June 1996. .

[74] Dao Nam Anh and Richard Moore. Formal Modelling of Large Domains. Technical Report 74, UNU/IIST, P.O.Box 3058, Macau, June 1996. Revised: September 1996 .

[75] Myatav Erdenechimeg and Richard Moore. Multi-directional Multi-lingual Script Processing. Technical Report 75, UNU/IIST, P.O.Box 3058, Macau, June 1996. .

[76] Ngo Quoc Tao, Dang Van Hung, and Tomasz Janowski. Sources on Map Generalization. Administrative Report 76, UNU/IIST, P.O.Box 3058, Macau, 24 July 1996.

[77] Mao Xiaoguang, Xu Qiwen, and Wang Ji. Towards a Proof Assistant for Interval Logics. Research Report 77, UNU/IIST, P.O.Box 3058, Macau, July 1996. .

[78] Venkat P Reddy. A First Course in Software Engineering. Technical Report 78, UNU/IIST, P.O.Box 3058, Macau, July 1996. .

[79] Yu Huiqun. A Course of Formal Methods in Software Development. Technical Report 79, UNU/IIST, P.O.Box 3058, Macau, August 1996. .

[80] Expert Group. Decision Support Systems for Sustainable Develpoment: Experience and Potential — a Position Paper. Administrative Report 80, UNU/IIST, P.O.Box 3058, Macau, August 1996.

[81] Do Tien Dung, Le Linh Chi, Nguyen Le Thu, Phung Phuong Nam, Tran Mai Lien, and Chris George. Developing a Financial Information System. Technical Report 81, UNU/IIST, P.O.Box 3058, Macau, September 1996. .

[82] Cornelis A. Middelburg. Truth of Duration Calculus Formulae in Timed Frames. Research Report 82, UNU/IIST, P.O.Box 3058, Macau, September 1996. .

[83] Tan Xinming. Enquiring about Bus Transport–Formal Development using RAISE. Technical Report 83, UNU/IIST, P.O.Box 3058, Macau, September 1996. .

[84] Nguyen Quoc Toan. An Example for the RAISE Development method: University School Administration. Technical Report 84, UNU/IIST, P.O.Box 3058, Macau, September 1996. .

[85] Dines Bjørner. UNU/IIST Annual Report 1996. Administrative Report 85, UNU/IIST, P.O.Box 3058, Macau, October 1996.

[86] Manoranjan Satpathy, Dang Van Hung, and Paritosh K. Pandya. Some Results on The Decidability of Duration Calculus under Synchronous Interpretation. Research Report 86, UNU/IIST, P.O.Box 3058, Macau, December 1996. .

[87] Jan A. Bergstra and Cornelis A. Middelburg. Process Algebraic Underpinning of Communication and Timing in SDL. Research Report 87, UNU/IIST, P.O.Box 3058, Macau, December 1996. .

[88] Nikolaj Nikitchenko. Towards Foundations of the General Theory of Transport Domains. Research Report 88, UNU/IIST, P.O.Box 3058, Macau, December 1996. .

[89] Dines Bjørner et al. LSD: Logical Systems Development. Lecture Note Compendium 89, UNU/IIST, P.O.Box 3058, Macau, Summer 1996. The report has been replaced by UNU/IIST Report 95. .

[90] Dines Bjørner. UN's Software Technology R&D in Africa, Asia, Eastern Europe, and Latin America — A Personal View. Research Report 90, UNU/IIST, P.O.Box 3058, Macau, December 1996.

[91] Zhou Chaochen and Michael R. Hansen. An Adequate First Interval Order Logic. Research Report 91, UNU/IIST, P.O.Box 3058, Macau, December 1996. .

[92] Tomasz Janowski and Rumel V. Atienza. A Formal Model For Competing Enterprises, Applied to Marketing Decision-Making. Research Report 92, UNU/IIST, P.O.Box 3058, Macau, January 1997. .

[93] Dines Bjørner, Chris W. George, Bo Stig Hansen, Hans Laustrup, and Søren Prehn. A Railway System, Coordination'97, Case Study Workshop Example. Research Report 93, UNU/IIST, P.O.Box 3058, Macau, January 1997. .

[94] Dines Bjørner et al. (Eds.). Design Calculi in Action — Five Years of R&D at UNU/IIST. Research Report 94, UNU/IIST, P.O.Box 3058, Macau, Jannuary 1997. Vols. I–V. Approx. 2,000 pages. Draft Compendium. Controlled Circulation

[95] Dines Bjørner. LSD: Logical Systems Development. Research Report 95, UNU/IIST, P.O.Box 3058, Macau, Jannuary 1997. Draft Compendium. Incomplete. Not for circulation

[96] Dines Bjørner. Models of Financial Services & Industries. Research Report 96, UNU/IIST, P.O.Box 3058, Macau, Jannuary 1997. Incomplete Draft Report

[97] Suman Roy and Zhou Chaochen. Notes on Neiborhood Logic. Research Report 97, UNU/IIST, P.O.Box 3058, Macau, February 1997. .

[98] J.A. Bergstra, C.A. Middelburg, and R. Şoricuţ. Discrete Time Network Algebra for a Semantic Foundation of SDL. Research Report 98, UNU/IIST, P.O.Box 3058, Macau, May 1997.

[99] C.A. Middelburg and Y.S. Ussenko. Towards a Semantics of φSDL suitable for Model Checking. Research Report 99, UNU/IIST, P.O.Box 3058, Macau, May 1997.

[100] C.A. Middelburg and B. Warinschi. Timed Frame Model for Discrete Time Process Algebra. Research Report 100, UNU/IIST, P.O.Box 3058, Macau, May 1997.

[101] Yang Lu Fu Hongguang and Zhou Chaochen. A Computer-Aided Geoemtric Approach to Inverse Kinematics. Research Report 101, UNU/IIST, P.O.Box 3058, Macau, April 1997. .

[102] Dines Bjørner in collaboration with Souleymane Koussoube and Roger Noussi. Jackson's Problem Frames: Syntax, Semantics and Pragmatics; Domains, Requirements and Design. Research Report 102, UNU/IIST, P.O.Box 3058, Macau, April 1997.

[103] Dang Van Hung. Modelling and Verificaton of Biphase Mark Protocols in Duration Calculus Using PVS/DC$^-$. Research Report 103, UNU/IIST, P.O.Box 3058, Macau, April 1997.

[104] Jose Nuno Oliveira. SETS — A Data Structuring Calculus and Its Application to Program Development. Research Report 104, UNU/IIST, P.O.Box 3058, Macau, May 1997. Lecture Notes of SETS Calculus Course, UNU/IIST, 5–16 May 1997.

[105] Hoang Thi Tung Lam and Richard Moore. Specification of a Switching Communications System. Technical Report 106, UNU/IIST, P.O.Box 3058, Macau, May 1997.

[106] Roger Noussi. An Efficient Construction of a Domain Theory for Resources Management: A Case Study. Research Report 107, UNU/IIST, P.O.Box 3058, Macau, May 1997. .

[107] Souleymane Koussoubé. Knowledge-Based Systems: Formalisation and Applications to Insurance. Research Report 108, UNU/IIST, P.O.Box 3058, Macau, May 1997. .

G.2 UNU/IIST Notes

References

[1] Søren Prehn et al. RAISE OH Foils. Oh foil lecture notes, UNU/IIST, P.O.Box 3058, Macau, Oct. '92, Nov. '92, April '93 1993. Lecture notes used for UNU/IIST two-week seminars in Beijing, Oct. 5–16, 1992, Pune (India), Nov. 24 – Dec. 8, 1992, Bangkok, April 19–30, 1993, and Hanoi May 10–21, 1993.

[2] Zhou Chaochen et al. Duration Calculus. Oh foil lecture notes, UNU/IIST, P.O.Box 3058, Macau, Oct. '92, Nov. '92, April '93 1992–93. Lecture notes used for UNU/IIST two-week seminars in Beijing, Oct. 5–16, 1992, Pune (India), Nov. 24 – Dec. 8, 1992, and Bangkok, April 19–30, 1993.

[3] Dines Bjørner. Development of Trustworthy Systems. Oh foil lecture notes, UNU/IIST, P.O.Box 3058, Macau, Feb. '92, April '93 1992–93. Lecture notes used for UNU/IIST two-week seminars in Bombay, Feb. 1992 (IFIP TC2/WG2.2 at TIFR), Bangkok, April 19–30, 1993, and Hanoi May 10–21, 1993.

[4] Dines Bjørner and Karl P.H.Leung. Semantics of Systems — From Application Domain Analysis to Requirements Capture with some Notes on Software Architecture — a Methodology Study. Technical note, UNUIIST, P.O.Box 3058, Macau, November 1995.

[5] Jan Goossenaerts. O H Foils. Oh foil lecture notes, UNU/IIST, P.O.Box 3058, Macau, Sept. '94 1994. Lecture notes used for UNU/IIST Field Study Workshop in Jakarta, Sept. 14–16, 1994.

[6] Dang Van Hung and Wang Ji. From Duration Calculus to Automata Models of Hybrid Systems. Research notes, UNU/IIST, P.O.Box 3058, Macau, August. '94 1994.

[7] Myatav Erdenechimeg and Richard Moore. Multi-Script: Domain Analysis and Requirements Capture. Technical note, UNU/IIST, P.O. Box 3058, Macau, December 1995.

[8] Dao Nam Anh. Airline Business Computing '2000: Informal Domain Analysis. Technical note, UNU/IIST, P.O. Box 3058, Macau, 27 November 1995.

[9] Dao Nam Anh. Airline Business Computing '2000: Formal Domain Analysis. Technical note, UNU/IIST, P.O. Box 3058, Macau, 27 November 1995.

[10] Dao Nam Anh. Airline Business Computing '2000: Requirements Capture. Technical note, UNU/IIST, P.O. Box 3058, Macau, 27 November 1995.

[11] Cleta Milagros Acebedo and Erwin Paguio. MI^{2}CI: Informal Domain Analysis. Technical Note RH X, UNU/IIST, P.O. Box 3058, Macau, 23 November 1995.

[12] Tomasz Janowski. MI^{2}CI: Requirements Capture. Technical Note RH X, UNU/IIST, P.O. Box 3058, Macau, 24 November 1995.

[13] Cleta Milagros Acebedo, Erwin Paguio, and Tomasz Janowski. MI^{2}CI: Software Architecture. Technical Note RH X, UNU/IIST, P.O. Box 3058, Macau, 23 November 1995.

[14] Ngo Quoc Tao. Map Generalization. Technical Note RH XIV.13, UNU/IIST, P.O. Box 3058, Macau, 1995.

[15] Md Rais, Dines Bjørner, and Ngo Quoc Tao. Decision Support Systems for Sustainable Development: Numerous documents. Technical & Administrative Notes RH XIV, UNU/IIST, P.O. Box 3058, Macau, 1995.

[16] Hong Mei. A Generic Concurrent Distributed Architecture. Technical note, UNU/IIST, P.O.Box 3058, Macau, June 1995.

[17] Liu Lian Suo and Yang Dong. Train Rescheduling Rules. Technical note, UNU/IIST, P.O.Box 3058, Macau, December 1995.

[18] Yang Dong. Basic Specification of PRaCoSy. Technical note, UNU/IIST, P.O.Box 3058, Macau, December 1995.

[19] Liu Lian Suo. PRaCoSy: Data Structures for Phase II. Technical note, UNU/IIST, P.O.Box 3058, Macau, December 1995.

[20] Roderick Durmiendo. Formal Development of a Digital Multiplexed Radio-Telephone System. Technical note, UNU/IIST, P.O.Box 3058, Macau, December 1995.

[21] Lu Jian. A Duration Calculus Proof Assistant for PVS. Research note, UNU/IIST, P.O.Box 3058, 1995.

[22] Dong Yu Lin. The Formal Development of a Railway Station Route Management System. M.sc. thesis, University of Macau, Taipa. Macau, May 1995. Available from UNU/IIST.

[23] Tomasz Janowski and Dines Bjørner. Virtual Library – On Refinement Towards an ODP Architecture. Research note, UNU/IIST, P.O.Box 3058, Macau, December 1995.

[24] Chris W. George, Tomasz Janowski, and Richard Moore. Domain Analysis for a Budgetary System. Technical note, UNU/IIST, P.O.Box 3058, Macau, October 1995.

[25] Achim Schneider, Alex Lai Iat Long, and Dines Bjørner. MaGICS: A Macau General Information and Command System. Technical note, UNU/IIST, P.O.Box 3058, 10 January 1995.

[26] Yang Zhen Yu, Lu Jian, and Chen ZongJi. A State-Space-based Approach to Acquisition of I/O-Automata — A Case Study for the Design of Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[27] Yang Zhen Yu. Application of Stochastic Differential Models and Optimization in Analysis of Stock Investments. Research note (extended abstract), UNU/IIST, P.O.Box 3058, 1995.

[28] Yang Zhen Yu. Hierarchical Control of an Industrial Process — A Case Study of Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[29] He Weidong. Optimization in Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[30] He Weidong and Zhang Minglian. Qualitative Simulation by Reasoning. Research note, UNU/IIST, P.O.Box 3058, 1995.

[31] He Weidong. Design of Stability in Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[32] He Weidong and Xu Qiwen. Hierarchical decision Maker in Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[33] He Weidong. Optimal Control in Controllable Phase Transition Hybrid Control Systems. Research note, UNU/IIST, P.O.Box 3058, 1995.

[34] Dines Bjørner, Chris George, and Tomasz Janowski. A Bouquet of *Info*Dev Project Proposals. Administrative Document Collection, UNU/IIST, P.O.Box 3058, 29 September 1995. RH II.11.4.

[35] Chris George. External Debt Management. Technical Report MoFIT/1, UNU/IIST, P.O. Box 3058 Macau, March 1996.

[36] Do Tien Dung, Le Linh Chi, Nguyen Le Thu, Phung Phuong Nam, and Tran Mai Lien. A Financial Information System: Domain Analysis: Narrative and Specification. Technical Report MoFIT/2, UNU/IIST, P.O. Box 3058 Macau, August 1996.

[37] Le Linh Chi, Nguyen Le Thu, Phung Phuong Nam, and Tran Mai Lien. Testing the Taxpayer Specification. Technical Report MoFIT/3, UNU/IIST, P.O. Box 3058 Macau, July 1996.

[38] Le Linh Chi and Nguyen Le Thu. Tax System Security. Technical Report MoFIT/4, UNU/IIST, P.O. Box 3058 Macau, August 1996.

[39] Tran Mail Lien. Tax Policy. Technical Report MoFIT/5, UNU/IIST, P.O. Box 3058 Macau, August 1996.

[40] Phung Phuong Nam. Optimising the Tax System Prototype. Technical Report MoFIT/6, UNU/IIST, P.O. Box 3058 Macau, September 1996.

[41] Do Tien Dung. Developing the Tax System Specification. Technical Report MoFIT/7, UNU/IIST, P.O. Box 3058 Macau, October 1996.