Banking
A Domain Description

Dines Bjørner

Technical University of Denmark
bjorner@gmail.com – www.dtu.dk/˜db

August 18, 2025: 12:03

Work in Progress:

– is updated, corrected, revised, extended daily !

An Incomplete Draft

- Some basic structure of this documents was set up in February 2025.
- I really began this document, in earnest, i.e., every day since early June 2025.
- **Thursday, June 26, 2025:**
  - I started with some chapter structure.
  - In the last week I shuffled various chapters.
  - Yesterday and today I insert chapters on Currencies, Exchange Rates, etc.
- **Sunday/Monday, June 29/30, 2025:**
  - I added Appendices on RSL and Domain Modeling, A–B.
- **Friday August 1, 2025:** In sections on customer, branch office, bank, etc., behaviours
  - some subsections are labeled 'Command', but should probably be labeled 'Message'
  - and some communicated messages which presently only lists a command should also be prefixed with a triplet: $(from_{ui}, \tau, to_{ui})$ –
  - or should all such triplets be "abandoned" ?
- **Saturday August 2, 2025:** "Completed" `open bank account` and `deposit` behaviours (for customers, branch offices and banks).
- **Saturday August 9, 2025:** Check, later, that account and bank history attributes are properly augmented.

Prelude

- We analyze and describe a conceptual domain of *banking*. Our “ambition” is to capture some essential aspects of banking: from Your “mortar-&-nrick” [neigbourhood] *local bank* with its *branch offices* and *head-quarter* via the *national bank* of Your country and the *central bank* of Your region (or continent) to *The World Bank* and *The International Monetary Fund:* which are their discernable structures, their individual operations and interaction.

  This motivation for work on this document is to test “my method”.

  To try it out an “near-real-life / real-scale” domains.

  Alas, the domain is being “researched”, analyzed and described by only one person, 87 years old !

  In a proper, professional, commercial banking domain modeling effort, see Sect. 18.4 on page 157, there would be 7-8 persons, well-trained in Domain Modeling, doing, in 3-4 months, what is here being done by 1 person in 3-4 months (June-Sept. 2025).
- This report assumes some familiarity with RSL, cf. Appendix A on page 169, and Domain Modeling, cf. Appendix B on page 197.

Dines Bjørner

Contents

I A General Setting 1

1 Introduction 3
1.1 On A Notion of 'Infrastructure' 3
1.2 A Dichotomy 4
1.3 The Dichotomy Resolved 5
1.4 A Series of Infrastructure Domain Models 5

II Introductory Remarks 7

2 Currencies, Exchange Rates and Interests 9
2.1 Introduction 9
2.2 Currency 10
2.3 Exchange Rates 10
2.4 Interests 11
2.5 Discussion 12

3 A First Take 13
3.1 A Survey 13
3.2 Examples 14
3.3 A First Rigorous Description 16
3.4 Summary 24

III Ordinary Customer Banking 25

4 Customers 27
4.1 External Qualities 29
4.2 Internal Qualities 30
4.3 Customer Commands 35
4.4 Customer Behaviours 37
4.5 Discussion 68

5 Banks 69
5.1 Introduction 70
5.2 Attributes 72
5.3 Banking System Communication 74
5.4 Bank Commands 75
5.5 Bank Behaviours 76
5.6 Discussion 97

6 **Branch Offices** **99**
6.1 External Qualities . . . 100
6.2 Internal Qualities . . . 100
6.3 Branch Office Intentional Pull . . . 102
6.4 Branch Office Commands . . . 103
6.5 Branch Office Behaviours . . . 104

7 **Bank Head Quarter** **129**
7.1 External Qualities . . . 129
7.2 Internal Qualities . . . 129
7.3 Bank Head Quarte Intentional Pull . . . 130
7.4 Bank Head Quarte Commands . . . 130
7.5 Bank Head Quarter Behaviours . . . 130

8 **Tellers** **131**
8.1 Tellers and Automatic Teller Machines[ATM] . . . 131
8.2 ATMs . . . 131
8.3 Discussion . . . 131

9 **Credit/Debit Company** **133**
9.1 External Qualities . . . 134
9.2 Internal Qualities . . . 134
9.3 Credit/Debit CompanyIntentional Pull . . . 134
9.4 Credit/Debit CompanyCommands . . . 134

10 **Mortgage, Savings and Loan Companies** **135**
10.1 External Qualities . . . 136
10.2 Internal Qualities . . . 136
10.3 Credit/Debit CompanyIntentional Pull . . . 136
10.4 Credit/Debit CompanyCommands . . . 136

IV **Stocks: Brokers & Exchange** 137

11 **Stock Brokers** **139**

12 **Stock Exchanges** **141**

V **National, Regional & Global Banks** 143

13 **National Banks: Endurants and Commands** **145**
13.1 Endurants . . . 145
13.2 National Bank Intentional Pull . . . 146
13.3 National Bank Commands . . . 146

14 **Central Banks: Endurants and Commands** **147**
14.1 Endurants . . . 147
14.2 Regional Bank Intentional Pull . . . 147
14.3 Regional Bank Commands . . . 147

15 **IMF: Endurants and Commands** **149**
15.1 Endurants . . . 149
15.2 IMF Intentional Pull . . . 149
15.3 IMF Office Commands . . . 149

16 **The World Bank: Endurants and Commands** **151**
16.1 Endurants . . . 151
16.2 The World Bank Intentional Pull . . . 151
16.3 The World Bank Commands . . . 151

VI **Closing** 153

17 **Discussion** **155**

18 **Conclusion** **157**
18.1 What Have We Achieved ? . . . 157
18.2 What Have We Not Achieved ? . . . 157
18.3 How Was This Domain Modelling Approached ? . . . 157
18.4 A Prelude to a Professional Banking Domain R & D . . . 157
18.5 What Next ? . . . 157
18.6 Acknowledgements . . . 157
18.7 References . . . 158

VII **Appendix** 165

A **A Raise Specification Language Primer** **167**
A.1 Types and Values . . . 169
A.2 The Propositional and Predicate Calculi . . . 174
A.3 Arithmetics . . . 176
A.4 Comprehensive Expressions . . . 176
A.5 Operations . . . 179
A.6 λ-Calculus + Functions . . . 186
A.7 Other Applicative Expressions . . . 188
A.8 Imperative Constructs . . . 191
A.9 Process Constructs . . . 191
A.10 RSL Module Specifications . . . 192
A.11 Simple RSL Specifications . . . 192
A.12 RSL$^+$: Extended RSL . . . 193
A.13 Distributive Clauses . . . 194
A.14 Space and Time . . . 194

B **The Domain Modeling Theory** **197**
B.1 Domains . . . 200
B.2 Six Languages . . . 202
B.3 Endurants and Perdurants, I . . . 204
B.4 A Domain Analysis & Description Ontology . . . 204
B.5 The Name, Type and Value Concepts . . . 207
B.6 Phenomena and Entities . . . 208
B.7 Endurants and Perdurants, II . . . 208
B.8 External and Internal Endurant Qualities . . . 209
B.9 Perdurant Concepts . . . 224
B.10 Facets . . . 235
B.11 Conclusion . . . 237

C **The Tokyo Stock Exchange** **241**
C.1 Introduction . . . 241
C.2 The Problem . . . 241
C.3 A Domain Description . . . 242

D Indexes 257
D.1 RSL Index 257
D.2 A Domain Modeling Index 259
D.3 Banking Domain Concepts 261
D.4 Formal Entities 262

Part I

A General Setting

Chapter 1

Introduction

Contents

1.1 **On A Notion of 'Infrastructure'** **3**

1.1.1 Domain Models 4

1.1.2 Some Characterizations 4

1.1.3 Purpose of Domain Models 4

1.1.4 Domain Science & Engineering 4

1.2 **A Dichotomy** **4**

1.2.1 An Outline 4

1.2.2 The Dichotomy 5

1.3 **The Dichotomy Resolved** **5**

1.4 **A Series of Infrastructure Domain Models** **5**

The Triptych Dogma

In order to *specify software*,
we must understand its requirements.

In order to *prescribe requirements*,
we must understand the *domain*.

So we must *study, analyze* and *describe* domains.

This is one of a series, [26, 31, 30, 29, 23], of domain studies of such infrastructure components as government, public utilities, banking, transport, insurance, health care, etc. The current, this 'Introduction' chapter is common to these study reports.

1.1 On A Notion of 'Infrastructure'

Central to our effort of studying "man-made" domains is the notion of *infrastructure*[1]. The *infrastructure* can be characterized as follows: *the basic physical and organizational structures and facilities (e.g. buildings, roads, power supplies) needed for the operation of a society or enterprise*, "`the social and economic infrastructure of a country`". We interpret the "for example, e.g.," to include, some already mentioned above: government structure: legislative, executive & judicial units, transport: roads, navigable rivers and lakes, the open sea, banking, educational system, health care, utilities: water, electricity, telecommunications (e.g. the

[1] https://en.wikipedia.org/wiki/Infrastructure

`Internet) gas, ,` etc.,[2] Also: Winston Churchill is quoted to have said in the House of Commons: *"The young Labour speaker we have just listened to wants clearly impressing his constituency with the fact that he went to Eton and Oxford since he now uses such modern terms as 'infrastructure' ".*

1.1.1 Domain Models

We rely on [27, 24, 19, 18, 15]. They provide a scientific foundation for modelling domains in the style of this report.

1.1.2 Some Characterizations

- **Domain:** By a *domain* we shall understand a *rationally describable* segment of a *manifest*[3], *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants*.
- **Endurants:** By *endurants* we mean those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster].
- **Perdurants:** By *perdurants* we mean those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster].
- **Domain Description:** By a *domain description* we shall here mean a syntactic entity, both narrative and formal, describing the domain. That is, a domain description is a structured text, such as shown in Chapters 2–16 (pages 9–151).
- **Domain Model:** By a *domain model* we shall here mean the mathematical meaning, the semantics as denoted the domain description.

1.1.3 Purpose of Domain Models

The *Triptych* dogma (above) expresses a relation of domain models to software. But domain models serve a wider role. Mathematical models of, say, physics, are primarily constructed to record our understanding of some aspects of the world – only secondarily to serve as a basis for engineering work. So it is with manifest models of infra structure components such banking, insurance, health care, transport, etc. In this, and a series of papers, [30, 29], we shall therefore present the result of infra structure studies. We have, over the years, developed many domain models: [5].

1.1.4 Domain Science & Engineering

A series of publications [15, 18, 19, 24, 28] reflects scientific insight into and an engineering methodology for analyzing and describing manifest domains.

1.2 A Dichotomy

1.2.1 An Outline

As citizens we navigate, daily, in a *God-given* and a *Man-made world.* The God-given world can be characterized, i.e., "domain described", as having natural science properties. The laws that these natural science properties obey are the same – all over the universe ! The Man-made world can be characterized, i.e., "domain described", as

[2] 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.

[3] The term 'manifest' is used in order to distinguish between these kinds of domains and those of computing and data communication: compilers, operating systems, database systems, the Internet, etc.

having infrastructure components[4]. The "laws" that these properties obey are not necessarily quite the same around our planet !

1.2.2 The Dichotomy

For our society to work, we are being educated (in primary, secondary, tertiary schools, colleges and at universities). We are taught to to read, write and [verbally] express ourselves, recon and do mathematics, languages, history and the sciences: physics (mechanics, electricity, chemistry, biology, botany's, zoology, geology, geography, ...), but we are not taught about most of the infrastructure structures[5]. That is the dichotomy.

1.3 The Dichotomy Resolved

So there it is:

- first *study* a or several domains;
- then *analyze*, *describe* and *publish* infrastructure domains;
- subsequently *prepare educational texts* "over" these;
- finally introduce *'an infrastructures'* school course.

1.4 A Series of Infrastructure Domain Models

So this *domain science & engineering* paper – on banking – is one such infrastructure domain description. In all we are and would like to work on these infrastructure domains:

- **Transport** https://www.imm.dtu.dk/ dibj/2025/infra/main.pdf [31]
- **Banking** https://www.imm.dtu.dk/ dibj/2025/infra/banking.pdf [26]
- etc.

A report on *double-entry bookkeeping* [23] relates strongly to most of these infra-structure component domains[6].

[4] state, regional and local government: executive, legislative and judicial, banking, insurance, health care (hospitals, clinics, rehabilitation, family physicians, pharmacies, ...), passenger and goods transport (road, rail, sea and air), manufacturing and sales, publishing (newspapers, radio, TV, books, journals, ...), shops (stores, ...),

[5] See footnote 4.

[6] `http://www.imm.dtu.dk/ dibj/2023/doubleentry/dblentrybook.pdf`

Part II

Introductory Remarks

Chapter 2

Currencies, Exchange Rates and Interests

Contents

2.1 **Introduction** **9**

2.1.1 The Main Players 9

2.1.2 The Main Concepts 10

2.2 **Currency** **10**

2.2.1 Currencies 10

2.2.2 Formal Model 10

2.3 **Exchange Rates** **10**

2.3.1 Informal 10

2.3.2 Formal 11

2.3.2.1 Buy/Sell Rate 11

2.3.2.2 Today's Exchange Rates: Thu. 26 June, 2025 11

2.4 **Interests** **11**

2.4.1 Deposit & Borrowing Rates 11

2.5 **Discussion** **12**

2.1 Introduction

2.1.1 The Main Players

- There are two main players:
  - customers and banks.
- We consider, in this document, only one kind of customer:
  - "lay" people, like You and me, ordinary people,
  - in contrast to enterprises and local and national governments

  seeking some form of association with banks:
  - depositing,
  - lending, and
  - borrowing monies.

2.1.2 The Main Concepts

- We need **money** to be able to sustain our daily life.
- And we, generally, **earn** or **accrue** money to do so.
- Money come in different **currencies**, Sect. 2.2.
- Cash in one currency can be **exchange**d for cash in another currency, Sect. 2.3.
- To **protect** our cash we can "put the money in the bank" – where it might accrue **interest**, Sect. 2.4 – and through whom [the bank] it can be currency exchanged.

2.2 Currency

2.2.1 Currencies

By 'currencies'[7] we shall here mean the variety of money currencies:

- Danish Kroner,
- Swedish Kroner,
- Norwegian Kroner,
- British Pound, £,
- Euro, €,
- Swiss Francs,
- Turkish Lira,
- Israeli Shekel,
- Thai Baht, THB,
- Singapore Dollar, S$,
- Hong-Kong Dollar, HK$,
- Macao Petacas,
- Chinese Yuan, CNY,
- Japanese Yen, ¥,
- US Dollar, $,

Etcetera !

2.2.2 Formal Model

1. There is a literal for each currency:

type
1. Currency = "**DKK**"|...|"**Pound**"|...|"**HKDollar**"|...|"**Yan**"|...|"**USD**"

2.3 Exchange Rates

2.3.1 Informal

Cash in "foreign currency", i.e., in a currency different from the one with which you wish to buy or sell an amount of that "other" currency, is *exchanged* "at a rate". There is usually one rate for buying and a another for selling. A rate is expressed as a pair, $((sell, curr_1), (cost, curr_2))$, of natural numbers: and currency names: I offer *sell* monies in currency $curr_1$, and get *cost* monies in currency $curr_2$. Likewise for buying: I offer *buy* monies in currency $curr_1$, and get *amount* monies in currency $curr_2$.

The *buy rate* is $\frac{sell\ curr_1}{cost\ curr_2}$, and the *sell rate* is $\frac{buy\ curr_1}{amount\ curr_2}$.

[7] https://en.wikipedia.org/wiki/List_of_currencies

2.3.2 Formal

2.3.2.1 Buy/Sell Rate

2. Four pieces of information are stated: the amount, `buy` [`sell`], of monies, in currency, `curr`$_1$, that will **buy** [`sell`] an amount, `bought` [`sold`], in currency, `curr`$_2$. We set `buy, sell` to 1 currency unit.

```
type
2. ExchangeRates = Buy × Sell
2. Buy, Sell = Curr_1 × Curr_2 →m Bought [resp. Sold]
value
2. buy: Buy → ExchangeRates → Bought
2. buy(a)(b,s) = a∗b(a)
2. sell: Sell × ExchangeRates → Sold
2. see(s)(b,s) = a∗s(a)
```

This `ExchangeRates` relation is determined, from *"day-to-day"*, by a national or central bank.

2.3.2.2 Today's Exchange Rates: Thu. 26 June, 2025

We refer to **https://danskebank.dk/privat/vaerktoejer/beregn/omregn-valuta**[8] and set the amounts to be bought and sold to 100 !

- 100 DKK to 13.40 €
- 100 € to 116.95 US$
- 100 £ to 874.35 DKK
- 100 € to 746.04 DKK
- 100 US$ to 85.5 €
- 100 DKK to 11.44 £

2.4 Interests

[9]

By 'interests' we shall here mean

MORE TO COME

2.4.1 Deposit & Borrowing Rates

- Ordinary, "You and me", customers, get one rate for our monies deposited with a bank,
  - in fact, there are, usually, several rates:
  - short- vs. long-term deposits, and
  - short- vs. long-term loans,
  - with some loans requiring installments, some not;
- and other rate for borrowing monies,
  - again subject to short- and long-term conditions.

MORE TO COME

[8] Check & double check !

[9] Danish: CIBOR-renten, eller *Copenhagen Interbank Offered Rate*, er en dagligt fastsat referencerente, der bruges som grundlag for rentesatser p mange finansielle produkter i Danmark, især lån og kreditter. Den beregnes som et gennemsnit af de renter, som et antal store banker er villige til at tilbyde hinanden ved udlån i danske kroner over en given periode. `https://da.wikipedia.org/wiki/CIBOR`

2.5 Discussion

TO BE WRITTEN

Chapter 3

A First Take

Contents

3.1 **A Survey** **13**
3.2 **Examples** **14**
3.3 **A First Rigorous Description** **16**
3.3.1 Main Endurants 16
3.3.2 External Qualities 16
3.3.2.1 The Sorts 16
3.3.2.2 A Global Parts State 18
3.3.3 Internal Qualities 19
3.3.3.1 Unique Identification 19
3.3.3.1.1 **Types & Observers** 19
3.3.3.1.2 **A Global Unique Identifier State** 20
3.3.3.1.3 **An Axiom on Part States** 21
3.3.3.1.4 **On Unique Identifiers** 22
3.3.3.1.4.1 **Retrieve functions** 22
3.3.3.1.4.2 **Constraints:** 23
3.3.3.2 Mereology 24
3.3.3.2.1 **Types and Observers** 24
3.3.3.2.2 **Wellformedness** 24
3.4 **Summary** **24**

3.1 A Survey

By ‘banking’[10] we shall here mean a [loose] structure of, “from bottom-up” of:

- **currency** – exchange rate and interests; [Chapter 2]
- a “first take” at a banking structure; [Chapter 3]
- ordinary bank **customers**; [Chapter 4]
- the ordinary customers, i.e., citizens, **bank**[11]; [Chapter 5]

[10] https://www.academicbooks.dk/content/introduction-banking

[11] – with a head quarter and one or more branch offices

- your neighbourhood bank's **branch office**; [Chapter 6]
- that branch office's **head-quarter** – *"the" bank*; [Chapter 7]
- **Tellers**; [Chapter 8]
- **credit/debit companies**; [Chapter 9]
- **mortgage, savings & loan banks**; [Chapter 10]
- **stock brokers & exchanges**; [Chapters 11–12]
- **national banks**,[12]; [Chapter 13]
- the **central [or regional] banks**,[13]; and [Chapter 14]
- the **continental** and **world-wide bank** authorities[14]. [Chapters 16–15]

The concept of *currency* is included in this study – see Chapter 2.

In this study we shall analyze and describe the endurants of this structure and their [perdurant] operations, endurants as well as perdurants: "things You can point at" and 'banking' actions You and they perform.

Your "mortar-&-brick" [neighbourhood] *local bank* is abstracted in the form of a pair: the set of branch offices as one entity closely "connected" (in its daily operations) with the bank head-quarters.

3.2 Examples

Examples of *banking structures* "as seen" from a country perspective are[15]:

- **Denmark:**
  - *Local Banks:* Den Danske Bank, Jyske Bank, ...
  - *National Bank:* National Banken
  - *Central Bank:* The European Central Bank
  - *The World Bank*
  - *IMF*
- **France:**
  - *Local Banks:* BNP Paris Bas, Societe Generale, Credit Agricole Group, ...
  - *National Bank:* Banque de France
  - *Central Bank:* The European Central Bank
  - *The World Bank*
  - *IMF*
- **England:**
  - *Local Banks:* HongKong & Shanghai Bank, Lloyds, Barclays, Standard Chartered, ...
  - *National Bank:* The Bank of England

[12] like the *Bank of England*, the US *Federal Reserve*, etc.

[13] like the *Bank of England*, the *European Bank*, the US *Federal Reserve*, etc.

[14] such as the *Asian Development Bank*, the *World Bank* and the *International Monetary Fund*, *IMF*

[15] TO BE CHECKED !

  - *The World Bank*
  - *IMF*

- **China:**
  - *Local Banks:* Industrial and Commercial Bank (ICBC), Bank of China, China Construction Bank, Agricultural Bank, Bank of Communications, Postal Savings Bank, ...
  - *National Bank:* Bank of China
  - *The World Bank*
  - *IMF*

- **United States of America:**
  - *Local Banks:* JPMorgan Chase, Bank of America, Citibank, Wells Fargo Bank, U.S. Bank, Goldman Sachs Bank, ...
  - *National Bank:* Federal Reserve
  - *The World Bank*
  - *IMF*

3.3 A First Rigorous Description

3.3.1 Main Endurants

3.3.2 External Qualities

3.3.2.1 The Sorts

3. There is “the entire”, world-wide banking system !
4. From the banking system we can observe a bank aggregate.
5. A bank aggregate consists of a set of one or more banks.
6. From a bank we can observe its head quarter.
7. And from a bank we can observe an aggregate of branch offices.
8. An aggregate of branch offices is a set of branch offices.
9. From the banking system we can observe a customer aggregate.
10. From a customer aggregate we can observe a set of customers.
11. Customers are atomic endurants.

type
3. WBS
4. BA
5. BS = B-**set**
5. B
6. HQ
7. BOA
8. BOS = BO-**set**
8. BO
9. CA
10. CS = C-**set**
11. C

value
4. **obs_**BA: WBS → BA
5. **obs_**BS: BA → BS
6. **obs_**HQ: B → HQ
7. **obs_**BOA: B → BOA
8. **obs_**BOS: BA → BOS
9. **obs_**CA: WBS → CA
10. **obs_**CS: CA → CS

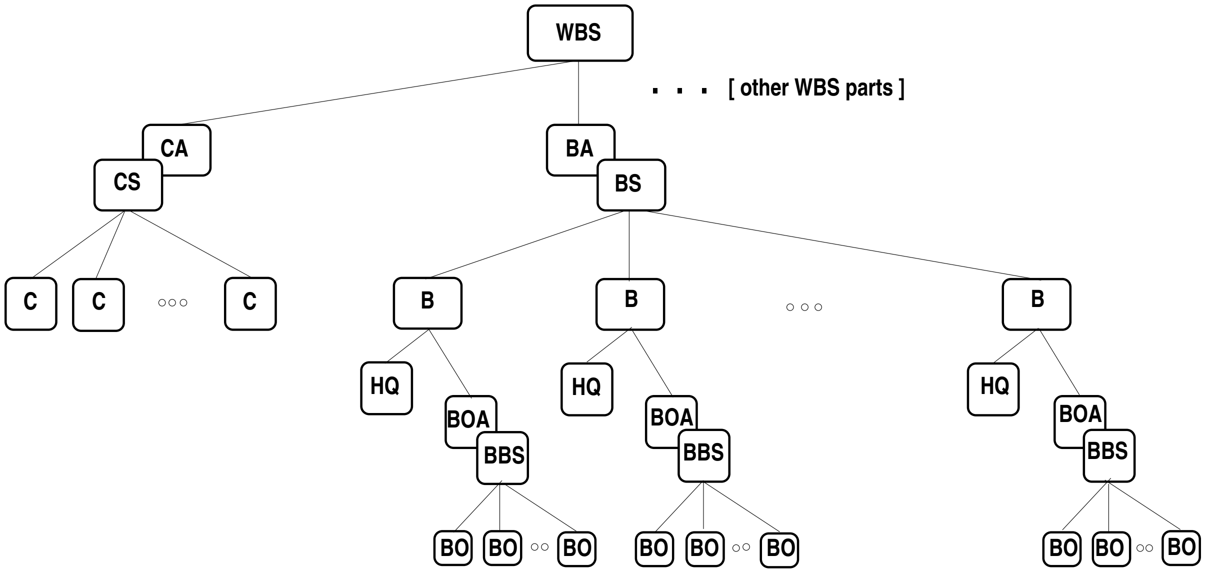


Figure 3.1: Bank + Customer Taxonomy

We show only a fragment of the world-wide banking system.

3.3.2.2 A Global Parts State

12. A given world-wide banking system forms a state.
13. From that we can observe “the banking” aggregate;
14. from which we can observe the set of all banks “in the world” !
15. And from these we can observe the sets of
    (a) all bank headquarters
    (b) all branch offices.
16. From the world-wide banking system we can observe a customer aggregate.
17. From the customer aggregate we can observe the set of customers.
18. Customers are atomic !
19. Together they all form an endurant parts state.

value

```
12.   wbs:WBS
13.   ba:BA = obs_BA(wbs)
14.   bs:BS = obs_BS(ba)
15a.  hqs:HQ-set = { obs_HQ(b) | b:B • b∈bs }
15b.  bos:BO-set = { obs_BA(b) | b:B • b∈bs }
16.   ca:CA = {obs_CA(wbs)}
17.   cs:CS = obs_CS(ca)
19.   σ_wbs = {wbs }∪{ ba }∪{ bs }∪ hqs ∪ bos∪{ ca}∪ cs
```

3.3.3 Internal Qualities

3.3.3.1 Unique Identification

3.3.3.1.1 Types & Observers

20. Parts have unique identifiers:

    (a) the world-wide banking system,

    (b) the banking aggregate,

    (c) the set of all banks,

    (d) each bank,

    (e) each bank head quarter,

    (f) each aggregate of branch offices,

    (g) each set of bank branch offices,

    (h) each branch office,

    (i) the customer aggregate.

    (j) the set of all customers, and

    (k) each customer.

type	**value**
20a. WBSI	20a. **uid**_WBS: WBS → WBSI
20b. BAI	20b. **uid**_BA: BA → BAI
20c. BSI	20c. **uid**_BS: BS → BSI
20d. BI	20d. **uid**_B: B → BI
20e. HQI	20e. **uid**_HQ: HQ → HQI
20f. BOA	20f. **uid**_BOA: BOA → BOAI
20g. BOSI	20g. **uid**_BOS: BOS → BOSI
20h. BOI	20h. **uid**_BO: BO → BOI
20i. CAI	20i. **uid**_CA: CA → CAI
20j. CSI	20j. **uid**_CS: CS → CSI
20k. CI	20k. **uid**_C: C → BOI

3.3.3.1.2 **A Global Unique Identifier State** We formulate the global unique identifier state in the style of that of the global parts state: $\iota 12\,\pi 18'$– $\iota 19\,\pi 18'$.

$\iota 12\,\pi 18'$ A given world-wide banking system forms a state.

$\iota 13\,\pi 18'$ From that we can observe the unique identifier of “the banking” aggregate;

$\iota 14\,\pi 18'$ from which we can observe the unique identifier of the set of all banks “in the world” !

$\iota 15\,\pi 18'$ And from these we can observe the sets

- $\iota 15\mathrm{a}\,\pi 18'$ the unique identifiers of all bank headquarters
- $\iota 15\mathrm{b}\,\pi 18'$ the unique identifiers all branch offices.

$\iota 16\,\pi 18'$ the unique identifier of the customer aggregate,

$\iota 17\,\pi 18'$ the unique identifier of the customer set, and

$\iota 18\,\pi 18'$ the unique identifiers of all the customers in that set.

$\iota 19\,\pi 18'$ Together they all form a unique identifier state.

value

$\iota 12\,\pi 18'$. wbs_{uid}:$\mathsf{WBSI} = \textbf{uid\_}(wbs)$

$\iota 13\,\pi 18'$. ba_{uid}:$\mathsf{BAI} = \textbf{uid\_}(\textbf{obs\_}\mathsf{BA}(wbs))$

$\iota 14\,\pi 18'$. bs_{uid}:$\mathsf{BSS} = \textbf{uid\_}(\textbf{obs\_}\mathsf{BS}(ba))$

$\iota 15\mathrm{a}\,\pi 18'$. hqs_{uid}:$\mathsf{HQI}\textbf{-set} = \{\ \textbf{uid\_}(\textbf{obs\_}\mathsf{HQ}(b)) \mid b{:}\mathsf{B} \bullet b \in bs\ \}$

$\iota 15\mathrm{b}\,\pi 18'$. bos_{uid}:$\mathsf{BOI}\textbf{-set} = \{\ \textbf{uid\_}(\textbf{obs\_}\mathsf{BA}(b)) \mid b{:}\mathsf{B} \bullet b \in bs\ \}$

$\iota 16\,\pi 18'$. ca_{uid}:$\mathsf{CAI} = \textbf{uid\_}(\textbf{obs\_}\mathsf{CA}(wbs))$

$\iota 17\,\pi 18'$. cs_{uid}:$\mathsf{CSI} = \textbf{uid\_}(\textbf{obs\_}\mathsf{CS}(ca))$

$\iota 18\,\pi 18'$. css_{uid}:$\mathsf{CI}\textbf{-set} = \{\ \textbf{uid\_}(\mathsf{c}) \mid \mathsf{c}{:}\mathsf{C} \bullet \mathsf{c} \in cs\ \}$

$\iota 19\,\pi 18'$. $\sigma_{wbs_{uid}} = \{wbs_{uid}\} \cup \{\ ba_{uid}\ \} \cup \{\ bs_{uid}\ \} \cup hqs_{uid} \cup bos_{uid} \cup \{\ ca_{uid}\ \} \cup css_{uid}$

3.3.3.1.3 An Axiom on Part States

21. Part identifiers are unique.

axiom

21. $\textbf{card}\,\sigma_{wbs} = \textbf{card}\,\sigma_{wbs_{uid}}$

• • •

Note on Manifest Bank Parts: I have “endowed” all parts so far introduced with unique identifiers. That is not to say that I presently, Sunday June 15, 2025, consider all these parts being manifest. For this reason I shall presently, 15.6.2025, for this section, only consider as *manifest* bank parts: **B**anks, their **H**ead **Q**uarters and **B**ranch **O**ffices.

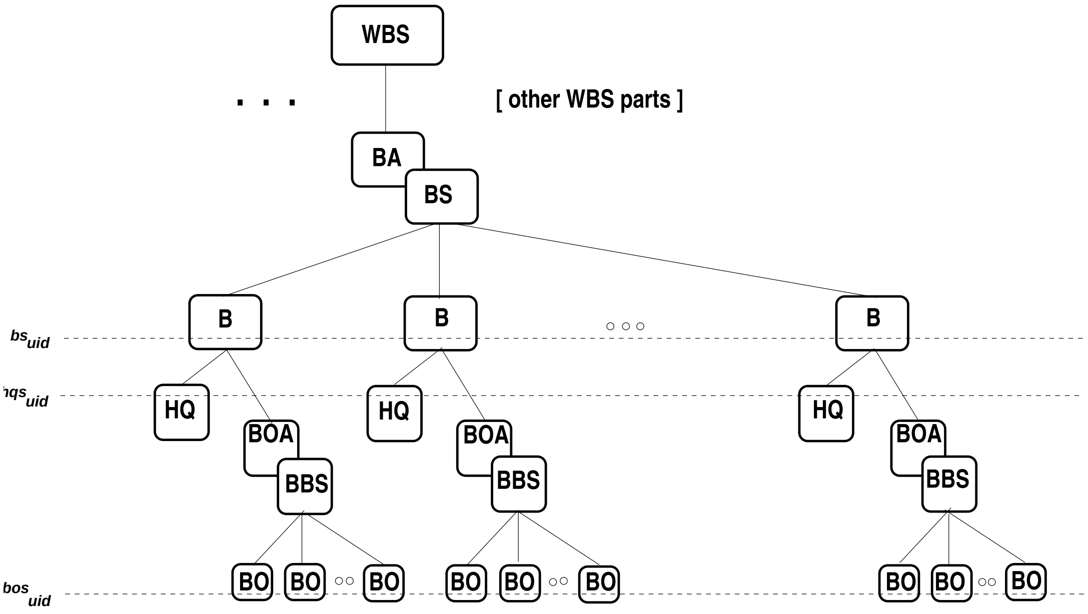


Figure 3.2: A Manifest Bank State

3.3.3.1.4 **On Unique Identifiers** Unique identifiers “embody” much information !

3.3.3.1.4.1 Retrieve functions

22. From the unique identifier of a bank we can observe
    (a) the identifier of “its” national bank,
    (b) the identifier of its head quarter and
    (c) the identifiers of all its branch offices (!).
23. From the unique identifier of a bank head quarter we can observe
    (a) the identifier of “its” national bank,
    (b) the identifier of “its” bank,
    (c) and the identifiers of all its branch offices (!).
24. From the unique identifier of a branch office we can observe
    (a) the identifier of “its” bank
    (b) and head quarter.

value
22a. xtr_NBI: B → NBI
22b. xtr_HQI: B → HQI
22c. xtr_BOIS: B → BOI-**set**
23a. xtr_NBI: HQ → NBI
23b. xtr_BI: HQ → BI
23c. xtr_BOIS: HQ → BOI-**set**
24a. xtr_BI: BO → BI
24b. xtr_HQI: BO → HQI

3.3.3.1.4.2 **Constraints:** The Wellformedness criteria related to banks amount to axioms and can be thought of as implying *"data vetting"*[16] – something to be "taken care of" by any software implementation.

25. For every bank
    - (a) the identifier of its head quarter must be in the set of all bank head quarter identifiers,
    - (b) and the identifiers of its branch offices must be in the set of all branch office identifiers,.
26. For every bank head quarter the identifiers of
    - (a) its branch offices
    - (b) and its bank

    must identify that head quarter.
27. For every branch office the identifiers of
    - (a) its head quarter
    - (b) and its bank

    must identify that branch office.

```
axiom [Consistent Bank Identification]
25.  ∀ b:B • b ∈ bs ⇒
25a.      let hqi = xtr_HQI(b) in hqi ∈ hqs_uid end
25b.   ∧ let bois = xtr_BOIS(b) in bois ∈ bos_uid
26.   ∧ ∀ hq:HQ • hq=obs_HQ(b) ⇒ let hqi=uid_HQ in
26a.        let bois = xtr_BOIS(hq) in ∀ boi∈bois⇒xtr_HQ(boi)=hqi end
26b.     ∧ let bi = xtr_BI(hq) in xtr_HQI(bi)=hqi end end end
27.   ∧ ∀ bo:BO • bo ∈ obs_BOS(b) ⇒ let boi=uid_BO in
27a.     ∧ let hqi′ = xtr_HQI(boi) in hqi = hqi′ end
27b.     ∧ let bi′ = xtr_BI(boi) in bi = bi′ end end
```

[16] Data vetting is a crucial process of checking the quality, accuracy, and reliability of data to ensure it meets specific requirements and is suitable for its intended use. Wikipedia

3.3.3.2 Mereology

3.3.3.2.1 Types and Observers

28. **Banks:** Banks relates, i.e., communicates with customers, their branch offices and their head quarter[17].
29. **Bank Headquarters:** communicates with the national bank, with other banks, with their branch offices and with their customers.
30. **Bank Branch Offices:** communicates with their head quarter, customers and with other banks.

type
28. BM = CI-**set** × BOI-**set** × HQI
29. HQM = NBI × HQI-**set** × BOI-**set** × CI-**set**
30. BOM = HQI × CI-**set** × BI-**set**
value
28. **mereo**_B: B → BM
29. **mereo**_HQ: HQ → HQM
30. **mereo**_BO: BO → BOM

3.3.3.2.2 Wellformedness

31. **Banks:** The various unique identifiers must be identifiers of the respective categories.
32. **Bank Headquarters:** The various unique identifiers must be identifiers of the respective categories.
33. **Bank Branch Offices:** The various unique identifiers must be identifiers of the respective categories.

value

31. wf_BM: BM → **Unit** > **Bool**

31. wf_BM(cis,bois,hqi)() ≡ bcs ⊆ cs_{uid} ∧ bois ⊆ bos_{uid}∧ hqi ∈ hqs_{uid}

32. wf_HQM: HQM → **Unit** → **Bool**

32. wf_HQM(nbi,hqis,bois,cis)() ≡ nbi ∈ nbs_{uid} ∧ hqis ⊆ hqs_{uid} ∧ bois ⊆ bos_{uid}∧ cis ⊆ css_{uid}

33. wf_BOM: BOM → **Unit** → **Bool**

32. wf_BOM(hqi,cis,bis)() ≡ hqis ∈ hqs_{uid} ∧ bis ⊆ bs_{uid}∧ bis ⊆ bs_{uid}

3.4 Summary

TO BE WRITTEN

[17] **Note:** This is a tentative model of bank mereologies. It, presently, omits, treatment of Credit/Debit companies, Mortgage, Savings & Loan Companies, Stock Brokers.

Part III

Ordinary Customer Banking

Chapter 4

Customers

Contents

4.1 **External Qualities** **29**

4.1.1 The Endurant Sorts 29

4.1.1.1 Core Local Bank Endurants 29

4.1.1.2 Customers 30

4.1.2 An Endurant State 30

4.2 **Internal Qualities** **30**

4.2.1 Unique Identification 30

4.2.1.1 Unique Identifier Sorts & Observers 30

4.2.1.2 A Unique Identifier State 31

4.2.2 Mereology 32

4.2.3 Attributes 33

4.2.3.1 Types and Observers 33

4.2.3.2 Attribute Wellformedness 35

4.3 **Customer Commands** **35**

4.4 **Customer Behaviours** **37**

4.4.1 Main Customer Behaviour 37

4.4.2 Pro-active Behaviours 38

4.4.2.1 Open Account 39

4.4.2.1.1 Command 39

4.4.2.1.2 Behaviour 39

4.4.2.2 Place Deposit 40

4.4.2.2.1 Command 40

4.4.2.2.2 Behaviour 40

4.4.2.3 Place Withdrawal 41

4.4.2.3.1 Command 41

4.4.2.3.2 Behaviour 41

4.4.2.4 Transfer 43

4.4.2.4.1 Command: 43

4.4.2.4.2 Behaviour 44
4.4.2.5 Exchange 45
4.4.2.5.1 Command 45
4.4.2.5.2 Behaviour 45
4.4.2.6 Open Display 46
4.4.2.6.1 Command 46
4.4.2.6.2 Behaviour 46
4.4.2.7 Close Display 47
4.4.2.7.1 Command 47
4.4.2.7.2 Behaviour 47
4.4.2.8 Obtain Credit/Debit Card 48
4.4.2.8.1 Command 48
4.4.2.8.2 Behaviour 48
4.4.2.9 Credit/Debit 49
4.4.2.9.1 Command 49
4.4.2.9.2 Behaviour 49
4.4.2.10 Close Credit/Debit Account 50
4.4.2.10.1 Command 50
4.4.2.10.2 Behaviour 50
4.4.2.11 Open Payment Service 51
4.4.2.11.1 Command 51
4.4.2.11.2 Behaviour 51
4.4.2.12 Close Payment Service 52
4.4.2.12.1 Command 52
4.4.2.12.2 Behaviour 52
4.4.2.13 Open Deposit Service 53
4.4.2.13.1 Command 53
4.4.2.13.2 Behaviour 53
4.4.2.14 Close Deposit Service 54
4.4.2.14.1 Command 54
4.4.2.14.2 Behaviour 54
4.4.2.15 Open Loan 55
4.4.2.15.1 Command 55
4.4.2.15.2 Command 55
4.4.2.16 Amortize Loan 56
4.4.2.16.1 Command 56
4.4.2.16.2 Behaviour 56
4.4.2.17 Increase Loan 57
4.4.2.17.1 Command 57
4.4.2.17.2 Behaviour 57
4.4.2.18 Close Loan 58
4.4.2.18.1 Command 58

4.4.2.18.2 Behaviour 58
4.4.2.19 Buy Stocks or Bonds 59
4.4.2.19.1 Command 59
4.4.2.19.2 Behaviour 59
4.4.2.20 Sell Stocks or Bonds 60
4.4.2.20.1 Command 60
4.4.2.20.2 Behaviour 60
4.4.2.21 Include/Exclude Debitor/Creditor 61
4.4.2.21.1 Command 61
4.4.2.21.2 Behaviour 61
4.4.2.22 Change Account Status 62
4.4.2.22.1 Command 62
4.4.2.22.2 Behaviour 62
4.4.2.23 Close Account 63
4.4.2.23.1 Command 63
4.4.2.23.2 Behaviour 63
4.4.3 Customer Re-active Behaviours 64
4.4.3.1 The `handle` Behaviour 64
4.4.3.2 ur2 64
4.4.3.2.1 Command 64
4.4.3.2.2 Behaviour 64
4.4.3.3 ur3 65
4.4.3.3.1 Command 65
4.4.3.3.2 Behaviour 65
4.4.3.4 ur4 66
4.4.3.4.1 Command 66
4.4.3.4.2 Behaviour 66
4.4.3.5 ur5 67
4.4.3.5.1 Command 67
4.4.3.5.2 Behaviour 67
4.5 **Discussion** **68**

In this chapter we treat the concept of “mortar & brick” *bank customers.* By customers we shall here mean non-bank individuals, businesses, organizations, communities, etc., who make use of local bank services.

4.1 External Qualities

4.1.1 The Endurant Sorts

4.1.1.1 Core Local Bank Endurants

34. There is the wold banking system – from which we observe

35. a, or the, banking aggregate, and

36. its set of banks.
37. There are banks.
38. he world banking system has an aggregate of customers.
39. Aggregates of customers are sets of these.
40. Etc.

```
type
34.  WBS
34.  BA, BS
37.  B
38.  CA
39.  CS = C-set
40.  ...
value
38.  obs_CA: WBS → CA
39.  obs_CS: CA → CS
```

4.1.1.2 Customers

For now we shall just analyze and describe a basic concepts of banks: namely that they have customers. Local bank customers have accounts in the bank, deposit and withdraw cash into these accounts, take loans against securities and pay off these loans, etc.

41. There are [local] bank customers: depositing and withdrawing, lending and borrowing.
42. Presently we shall not speculate as to how these customers otherwise “appear” !

```
type
41.  C
```

4.1.2 An Endurant State

TO BE WRITTEN

4.2 Internal Qualities

4.2.1 Unique Identification

4.2.1.1 Unique Identifier Sorts & Observers

43. Customers have unique identification.

```
type
43.  CI
value
43.  uid_C: C → CI
```

4.2.1.2 A Unique Identifier State

4.2.2 Mereology

44. Customers communicate with Branch Offices, banks, ... [to be filled in !]

type
44. $\mathsf{CM} = \mathsf{BOI}\textbf{-set} \times \mathsf{BI}\textbf{-set} \times \ldots$
value
44. $\textbf{mereo}\_\mathsf{C}\colon \mathsf{C} \to \mathsf{CM}$

4.2.3 Attributes

4.2.3.1 Types and Observers

45. Customers possess administrative information – such as name, birth date, address, marital status, and place of work, etc.

We need not be concerned with the representation of the above administrative items.

46. Customers have income: salaries, yield of bonds, stocks, etc.

47. Customers posses `cash_on_hand` `Cash_on_hand` could, e.g., be cash in any number of currencies.

48. Customers posses additional assets such as stocks, bonds, trade-able commodities (jewels, land, house, car, ...), etc. Stocks (stock portfolio) could be a set of company stocks, one or more per company. Bonds similarly.

49. Customers have **debitors**[18]: **"parties:"** enterprises, institutions, other customers, etc., to whom a customer owes monies. As an attribute we "list" them as a set of debitor identifiers.

50. Customers have creditors[19,20]: **"parties:"** enterprises, institutions, other customers, etc., to whom the customer owes monies. As an attribute we "list" them as a set of creditor identifiers.

51. Customers have debitors[21]

52. Customers [thus] have liabilities: Loans, ...

53. Customers pay taxes: ...

54. Customers "possess" awareness of bank offices with whom it already have accounts, or with whom it might wish to have accounts. identified by bank registration and account numbers.[22]

55. Customers have one or more bank accounts, in one or more banks, in possibly different [countries and currencies], ...

Banks have bank identifiers and accounts are "numbered".[23] We do not model the balance on these accounts. That information is with the bank and can be inquired.

[18] Debitor: a person who owes a debt [Cambridge Dict.]. A now obsolete term !

[19] A **creditor** or **lender** is a party (e.g., person, organization, company, or government) that has a claim on the services of a second party. It is a person or institution to whom money is owed. The first party, in general, has provided some property or service to the second party under the assumption (usually enforced by contract) that the second party will return an equivalent property and service. The second party is frequently called a debtor or borrower. The first party is called the creditor, which is the lender of property, service, or money.

Creditors can be broadly divided into two categories: secured and unsecured.

A secured creditor has a security or charge over some or all of the debtor's assets, to provide reassurance (thus to secure him) of ultimate repayment of the debt owed to him. This could be by way of, for example, a mortgage, where the property represents the security.

An unsecured creditor does not have a charge over the debtor's assets.

[20] O'Sullivan, Arthur; Sheffrin, Steven M. (2003). Economics: Principles in Action. Upper Saddle River, NJ: Pearson Prentice Hall. p. 264. ISBN 0-13-063085-3.

Insolvency for creditors. Australian Securities and Investments Commission. Retrieved March 22, 2022. King, Lawrence P.; Cook, Michael L. (February 1, 1989). Creditors' Rights, Debtors' Protection, and Bankruptcy. M. Bender. ISBN 9780256148237. Retrieved February 1, 2019 via Google Books.

[21] Debitor: A debitor, also known as a debtor, is an individual, company, or organization that owes money to another entity, called a creditor. Essentially, a debitor is someone who has a debt obligation. This debt could arise from various situations, such as loans, credit purchases, or contractual agreements.

[22] The identified accounts are "registered in" the identified bank[s].

[23] From bank identifiers and from account numbers ne can observe, i.e., extract their nationality and currencies

56. Customers have one or more credit and or debit cards, also [just] registered by customer name and card identifier.[24]
57. A customer may view the contents of any of her accounts.
58. We do not further describe that `Account Information`.
59. ...

type

45. AdminInfo = Name×Nationality×BirthDate×Addresses×MarStat×Work×...

45. Name = ...

45. Nationality = ... [25]

45. Birthdate = ...

45. Addresses = ...

45. MarStat = ...

45. Work = ...

46. Income = Source $\xrightarrow[m]{}$ **Nat**

47. Cash = Currency $\xrightarrow[m]{}$ **Nat** [26]

48. Assets = Stocks×Bonds×Commodities××RealEstate×...

48. Stocks = ...

48. Bonds = ...

48. Commodities = ...

48. RealEstate = ...

51. Debitors = DI-**set**

51. DI = ...

50. Creditors = KI-**set** ...

50. KI = ...

53. Liabilities = Loans×....

53. Taxes = Kind $\xrightarrow[m]{}$ **Nat**.

54. Banks = BOI-**set**

55. Accounts = BankID $\xrightarrow[m]{}$ AcctNu

55. BankID

55. AcctNu

56. Cards = Name×CDId-**set**

56. CDId

57. Displays = BankID $\xrightarrow[m]{}$ (AcctNu $\xrightarrow[m]{}$ AcctInfo)

58. AcctInfo = ...

59. ...

value

45. **attr**_AdminInfo: C→AdminInfo

46. **attr**_Income: C→Income

47. **attr**_Assets: C→Assets

52. **attr**_Liabilities: C→

53. **attr**_Taxes: C→Taxes

54. **attr**_Banks: C→Banks

55. **attr**_Accounts: C→Accounts

56. **attr**_Cards: C→Cards

57. **attr**_Displays: C → Displays

59. ...

55. xtr_UIs: BankID → BI × BOI-**set**

55. xtr_UIs: AcctNu → BI

[24] Again, see the previous footnote: credit cards are "registered in" the identified credit/debit card companies.

4.2.3.2 Attribute Wellformedness

TO BE WRITTEN

4.3 Customer Commands

Commands are syntactic entities. They are not endurants. They are an aspect of *domain facets*. In this chapter, i.e., Chapter 4, they are formulated by customers and issued "towards" banks. Semantically they denote state changes. The state referred to is the state of the bank. We shall treat the semantics of commands in respective chapters. Domain facets, as a wider concept than just commands, were first treated in [11, *2008*]. [19, *Chapter 8, 2021*] places facets in the wider context of domain modeling. Commands are but just one of the many kinds of facets, the *script* facet treated in [11]. Others are *support technology, rules & regulations, license languages, management & organization* and *human behaviour*.

These are presently 23 user commands of interest. Users direct commands at:

- **Banks:** `Display, Deposit, Withdraw, Transfer, CreditDebit, IncludeExcludeDebitor, IncludeExcludeCreditor, OpenPaymentService, ClosePaymentService, AmortizeLoan`
- **Branch Offices:** `OpenAcct, CloseAcct, ApplyLoan, OpenLoan, CloseLoan, IncreaseLoan`
- **Branch Offices or Credit/Debit Companies:** `ObtainCreditDebitCard, CreditDebit, CloseCreditDebitCard`
- **Branch Offices or Mortgage Companies:** `OpenLoan, CloseLoan, IncreaseLoan`
- **Branch Offices or Stock Brokers:** `BuyStockBond, SellStockBond`
- Etcetera !

The ordering in which I treat these commands is pragmatic.

60. There are [presently] 24 kinds of user to bank and branch office commands – cf. Items ι60a π35– ι60w π36:
    (a) The **Open Account** command requests the bank to open an account [of some currency] [DIRECTED AT BRANCH OFFICE].
    (b) The **Deposit** command requests the bank to accept cash to be added to an account [DIRECTED AT BANK].
    (c) The **Withdraw** command requests the bank to deliver cash, being subtracted from an account [DIRECTED AT BANK].
    (d) The **Transfer** command requests the bank to withdraw monies from one account and deposit in another account, either one of the requestor's other accounts, or some other account holder's account, nationally or internationally [DIRECTED AT BANK].
    (e) The **Exchange** command requests the buying or selling, i.e., the exchange, of one currecny for another – with the cost to be carried by one of the customers accounts.
    (f) The **Open Display** command requests the bank to let the customer have display some form of presentation of the customers account with a specific bank [DIRECTED AT BANK].

[25] Customers may have dual (or more ?) nationality !

[26] Cash may be in several currencies !

(g) The **Close Display** command requests the bank to cease updating the display of some form of presentation of the customers account with a specific bank [DIRECTED AT BANK].

(h) **Obtain Credit/Debit Card:** [DIRECTED AT BANK OR CREDIT/DEBIT COMPANY].

(i) **Pay with Credit/Debit Card:** [DIRECTED AT BANK OR CREDIT/DEBIT COMPANY].

(j) **Close Credit/Debit Card:** [DIRECTED AT BANK OR CREDIT/DEBIT COMPANY].

(k) **Open Payment Service:** Allow enterprises (employers, government institutions, pension funds, etc. to *withdraw* "cash" into named accounts. Not the same as debitor ! ? [DIRECTED AT BRANCH OFFICE].

(l) **Close Payment Service:** Terminate service [DIRECTED AT BRANCH OFFICE].

(m) **Open Deposit Service:** Allow enterprises (employers, government institutions, pension funds, etc. to *deposit* "cash" into named accounts. Not the same as debitor ! ? [DIRECTED AT BRANCH OFFICE].

(n) **Close Deposit Service:** [DIRECTED AT BRANCH OFFICE].

(o) The **Open Loan** command requests the bank to open that loan for withdrawals or transfers [DIRECTED AT BRANCH OFFICE OR MORTGAGE COMPANY].

(p) The **Amortize Loan** command repays on the loan [DIRECTED AT BRANCH OFFICE OR MORTGAGE COMPANY].

(q) The **Increase Loan** command increases the loan [DIRECTED AT BRANCH OFFICE OR MORTGAGE COMPANY].

(r) The **Close Loan** command requests the bank to terminate a loan, either delivering [or subtracting] the balance in cash or transferring the balance to an account [DIRECTED AT BRANCH OFFICE OR MORTGAGE COMPANY].

(s) The **Buy Stock or Bond or ...** command requests the bank to purchase a given number of stocks within a stock price range range, or "similarly" for bonds, or ... [DIRECTED AT BRANCH OFFICE OR STOCK BROKER].

(t) The **Sell Stock or Bond or ...** command requests the bank to sell a given number of stocks within a stock price range range, or "similarly" for bonds, or ... [DIRECTED AT BRANCH OFFICE OR STOCK BROKER].

(u) The **Include/Exclude Creditor Debitor** command informs the bank that the customer expects the debitor to regularly either withdraw monies from an account or transfer monies to the debitor from an account.

(v) The **Change Account Status** command requests the bank to edit/modify the account information [DIRECTED AT BRANCH OFFICE].

(w) The **Close Account** command requests the bank to terminate an account, either delivering [or subtracting] the balance in cash or transferring the balance to another account. Outstanding open displays, credits, payment sevices and loans are terminated ! [DIRECTED AT BRANCH OFFICE].

Whether these 23 commands are issued by the requestor "turning up in the bank, physically", or electronically, or otherwise – their abstraction amounts to the same !

```
type
60. Cmd =
60a.    OpenAcct
60b.  | Deposit
60c.  | Withdraw
60d.  | Transfer
60e.  | Exchange
60f.  | OpenDisplay
60g.  | CloseDisplay
60h.  | ObtainCreDebCard
60i.  | CreditDebit
60j.  | ClosCreDebCard
60k.  | OpenPaymentService
60l.  | ClosePaymentService
60m.  | OpenDepositService
60n.  | CloseDepositService
60o.  | OpenLoan
60p.  | AmortizeLoan
60q.  | IncreaseLoan
60r.  | CloseLoan
60s.  | BuyStockBond
60t.  | SellStockBond
60u.  | IncludeExcludeDebitorCreditor
60v.  | ChgeAcctStatus
60w.  | CloseAcct
```

4.4 Customer Behaviours

4.4.1 Main Customer Behaviour

The customer behaviour is what initially determines actions.

61. The customer behaviour internal non-deterministic (⊓) alternates between

    (a) pro-active actions: issuing commands to banks, branch offices, etc., and reactive actions: responding to messages from banks, branch office, etc., and

    (b) external non-deterministically responding handling responses from money institutions.

```
type
61. customer: CI → CM → CustAdminInfo
61.     → (Income×Assets×Liabilities×Taxes×Accounts×Cards×...×CHist) Unit
value
61. customer(ci)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch) ≡
61a.      pro_active_customer(ci)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
61b.  ⊓ re_active_customer(ci)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
```

62. The pro-active customer behaviour internal non-deterministically (⌈⌉) alternates behaviours specific to each of the 23 commands that users may issue.

```
62.  pro_active_customer(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
60a.      open_account(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60b.  ⌈⌉ deposit(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60c.  ⌈⌉ withdraw(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60d.  ⌈⌉ transfer(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60e.  ⌈⌉ exchange(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60f.  ⌈⌉ open_display(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60g.  ⌈⌉ close_display(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60h.  ⌈⌉ obtain_credit_debit_card(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60i.  ⌈⌉ credit_debit(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60j.  ⌈⌉ close_credit_debit_card(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60k.  ⌈⌉ open_paym_serv(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60l.  ⌈⌉ close_paym_serv(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60m.  ⌈⌉ open_deposit_serv(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60n.  ⌈⌉ close_deposit_serv(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60o.  ⌈⌉ open_loan(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60p.  ⌈⌉ amortize_loan(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60q.  ⌈⌉ increase_loan(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60r.  ⌈⌉ close_loan(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60s.  ⌈⌉ buy_stock_bond(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60t.  ⌈⌉ sell_stock_bond(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60u.  ⌈⌉ incl_excl_deb_cred(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60v.  ⌈⌉ change_acct_status(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
60w.  ⌈⌉ close_acct(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
```

63. The re-active customer behaviour external non-deterministically ([])

   (a) awaits (i.e., responds to) messages from banks, branch offices, etc.,

   (b) with these messages then being handled.

```
63.  re_active_customer(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
63a.     let msg = [] { comm[{ci,ui}]? | ui ∈ bois ∪ {bi} } in
63b.     handle(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)(msg) end
```

4.4.2 Pro-active Behaviours

4.4.2.1 **Open Account**

The case of the `open account` behaviour illustrates the following:

- the customer directs the request for an `Open_Account` to a branch office, item 65i, page 39.
- The branch office, Sect. 6.5.2.1 on page 106, examines the request, item 159a, page 106.
- If customer request is deemed unacceptable the customer is advised, item 159b, page 106, of the rejection.
- Otherwise the branch office deems the request acceptable, and, in turn, requests, item 159d, page 106, the bank, Sect. 5.5.2.1 on page 78, for a new account.
- The bank replies, item 127d, page 79, zero-setting a new account, item 127c, page 79, with such a number;
- and the customer duly informed, item 159f, page 106.

4.4.2.1.1 Command

64. In order to open an account with a bank the customer must provide the following a rough approximation of:

- admin. info.,
- income,
- assets,
- liabilities,
- debitors,
- creditors,
- and the currency of the new account.

type

64. OpenAcct :: $(\text{AdminInfo}'\times\text{Income}'\times\text{Assets}'\times\text{Liabilities}'\times\text{Debitors}'\times\text{Creditors}')\times\text{Currency}$

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.1 on page 106.

4.4.2.1.2 Behaviour

65. The `Open Account` behaviour assembles excerpts[27] of customers'

(a) income,

(b) assets,

(c) debitors,

(d) creditors,

(e) liabilities; and

(f) currency;

(g) chooses a branch office,

(h) and records the time;

(i) forms `Open Account` command,

(j) communicates it to the branch office.

(k) Then the customer awaits a reply from the branch office.

(l) The reply is used to update the customers' accounts information[28]

(m) whereupon the customer resumes being a customer.

(n) Preconditions are: The auxiliary functions `open_acct_income`, `open_acct_ass`, `open_acct_dis`, `open_acct_kis`, and `open_acct_liabilities` are well-defined.

[27] These excerpts depend on what the bank requires !

[28] We model only the case of a positive reply – leaving it to the reader to model the case of the bank not willing to open a new account for this user !

```
value
65.  open_account(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accts,cards,dis,...,ch) ≡
65a.,65b.  let inc′ = open_acct_income(inc), as′ = open_acct_assets(as),
65c.,65d.      dis′ = open_acct_dis(dis), kis′ = open_acct_kis(kis),
65e.,65f.      lia′ = open_acct_liabilites(lia), curr:Currency,
65g.           boi = select_bank(inc,cash,as,dis,kis,lia,tax,banks,accts,cards,dis),
65h.           τ = record TIME() in
65i.     let op_acct_cmd = ((ci,τ,boi),mk_OpenAccount((info,inc′,as′,dis′,kis′,lia′),curr)) in
65j.     comm[{ci,boi}] ! open_account_cmd ;
65k.     let mk_NewAcct(bi,anu) = comm[{ci,boi}] ? ;
65l.     let accts′ = accts † [ bi ↦ anu][29] in
65m.     customer(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs′,cards,dis,...,⟨op_acct_cmd⟩⁀ch)
65.      end end end end

65m.  pre: open_acct_income, open_acct_assets, ..., open_acct_liabilites are well−dined.
```

We leave it to the reader to sketch the auxiliary functions `open_···` and `select_bank`.

4.4.2.2 **Place Deposit**

4.4.2.2.1 Command

66. In order for a customer to deposit cash into an account of that customer, the following must be provided:

    - bank identification,
    - account number,
    - the amount of cash and the
    - identified currency

```
type
66.  Deposit :: BI × AcctNu × Cash × Currency
```

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.2 on page 80.

4.4.2.2.2 Behaviour

67. The `deposit` behaviour

    (a) selects an appropriate bank,

    (b) and an appropriate currency account of that customer with that bank,

    (c) decides upon the currency and a suitable amount of cash of that currency to be deposited into the account of that bank,

    (d) records the time,

    (e) and assembles this into a `Deposit` command

    (f) which it them communicates to the chosen bank –

    (g) from where it awaits a confirmation message

    (h) whereupon the bank conditionally[30] updates its `cash`,

[29] If `bi` is not in the domain of `accts` then `accts′ = accts ∪ [ bi ↦ anu]`

[30] we do not show the case where the bank declines the deposit – a simple **if .. then .. else .. end** clause

(i) and resumes being a customer [behaviour].

(j) Pre-conditions are: The auxiliary functions: `select_...` are well-defined, hence, for example, `curr` is in the **domain** of `cash`.

```
value
67.   deposit(ci)(bois,bi)(info)(prgr:(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)) ≡
67a.      let bi = select_bank(prgr),
67b.          anu = acct_number(prgr),
67c.          (curr,monies) = select_curr_cash(prgr),
67d.          τ = record 𝕋𝕀𝕄𝔼() in
67e.      let deposit_cmd = ((ci,τ,boi),mk_Deposit(bi,anu,curr,cash)) in
67f.      comm[{ci,boi}] ! deposit_cmd ;
67g.      let ok:mk_DepositOK(t,ci,anu.curr,old−balance,new−balance) = comm[{ci,boi}] ? in
67h.      let cash′ = cash † [ curr ↦ (cash(curr) − cash) ] in
67i.      customer(ci)(bois,bi)(info)(inc,cash′,as,dis,kis,lia,tax,banks,accs,cards,dis,...,⟨ok,deposit_cmd⟩⌢ch)
67.       end end end end
```

4.4.2.3 Place Withdrawal

4.4.2.3.1 Command

68. In order for a customer to withdraw cash from an account of that customer, the following information must be provided:

- bank identification,
- account number,
- currency,
- amount to be withdrawn.

```
type
68. Withdraw :: BI × AcctNu × Currency × Amount [= Nat]
```

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.3 on page 82.

4.4.2.3.2 Behaviour

69. The `withdraw` behaviour

(a) selects an appropriate bank,

(b) and an appropriate account of that customer with that bank,

(c) decides upon a suitable amount of cash to be witdrawn from the account of that bank,

(d) records the time,

(e) and assembles this into a `Wihdraw` command

(f) which it them communicates to the chosen bank.

(g) It then awaits a `reply` from the bank.

(h) There are two kinds of replies.

(i) Either the reply is "OK"

  i. in which case the customer's cash is incremented

  ii. and the customer resumes being so.

(j) Or the reply is "Not OK"

  i. whereupon it resumes being a customer [behaviour].

(k) Pre-conditions: Auxiliary functions `select_...` are welldefined.

```
value
69.      withdraw(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
69a.       let bi = select_bank(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis),
69b.           anu = select_acct_number(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis),
69c.           (curr,amount) = select_amount_curr(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis),
69d.           τ = record TIME() in
69e.       let withdraw_cmd = ((ci,τ,boi),mk_Withdraw(bi,anu,(curr,amount))) in
69f.       comm[{ci,boi}] ! withdraw_cmd ;
69g.        let reply = comm[{ci,boi}] ? in
69h.        case reply of
69i.            mk_OKWDR(anu,curr,amount,newbal) →
69(i)i.             let cash′ = cash † [curr ↦ cash(curr)+amount] in
69(i)ii.             customer(ci)(bois,bi)(info)
69(i)ii.                 (inc,cash′,as,dis,kis,lia,tax,banks,accs,cards,dis,...,⟨reply,withdraw_cmd⟩⌢ch) end
69j.            mk_NOKWDR(anu,curr,amount,bal) →
69(j)i.             customer(ci)(bois,bi)(info)
69(j)i.                 (inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,⟨reply,withdraw_cmd⟩⌢ch)
69.        end end end end

69.      pre: Auxiliary functionss select_... are welldefined.
```

4.4.2.4 Transfer

4.4.2.4.1 Command: The `transfer` command has two “versions”. The customer either transfers fund between “own” accounts, or from an own account to that of another customer. Syntactically the forms are indistinguishable.

70. The `Transfer` command has two elements:
    - the *from* element consists of
        - customer identifier,
        - accont number,
        - currency, and
        - amount
    - and the *to* element consists of
        - customer identifier, and
        - accont number
71. the account numbers differ.

type
70. Transfer :: (CI×AcctNu×Curr×Amount) × (CI×AcctNu)
70. Amount = **Nat**
axiom
71. $\forall$ ((ci1,anu1,curr1,am1),(ci2,anu2)):Transfer • anu1$\neq$anu2

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.4 on page 84.

4.4.2.4.2 Behaviour

72. The customer decides to transfer an amount of curency:

    (a) records the time;
    (b) selects currency and amount,
    (c) from account number, and
    (d) to customer account;
    (e) assembles the "from" and "to" elements
    (f) of the command
    (g) communicated to the bank.
    (h) Awaits `reply` from that bank.
    (i) Examines the reply[31], whereupon
    (j) it resumes being a customer.
    (k) Pre-conditions: Auxiliary functions `select_...` well defined.

value

```
72.   transfer(ci)(bois,bi)(info)(prgr:(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)) ≡
72a.      let $\tau$ = record $\mathbb{TIME}$() in
72b.      let (curr,amount) = select_Curr_Amount(prgr) ,
72c.          f_anu = select_AcctNu(prgr) ,
72d.          (t_ci,t_anu) = select_to_CI_AcctNu(prgr) ,
72e.          let from = (ci,f_anu,curr,amount), to = (t_ci,t_anu,curr,amount) in
72f.      let cmd = ((ci,$\tau$,bi),mk_Transfer(from,to)) in
72g.      comm[{ci,bi}] ! cmd ;
72h.      let reply = comm[{ci,bi}] ?  in
72i.      if $\mathscr{B}$(reply) = ... then ... else ...
72j.      customer(ci)(bois,bi)(info)(inc,as,dis,kis,lia,tax,banks,accs,cards,dis,...,$\langle$reply,cmd$\rangle^\frown$ch)
72.       end end end end end end

72k.  pre: select_... well defined
```

[31] We leave it to the reader to decide what such an examination might be ! We rfer to Sect. 5.5.2.4 on page 84.

4.4.2.5 **Exchange**

4.4.2.5.1 Command

73. The

    (a)

    (b)

    (c)

type
73.
73a.
73b.
73c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.5 on page 91.

4.4.2.5.2 Behaviour

74. (a)

    (b)

    (c)

value
```
74.  exchange(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
74a.
74b.
74c.
```

4.4.2.6 **Open Display**

By display we shall here understand some visualization of any specific account. By opening a display for a specific account, not already opened (and not [yet] closed) we shall "imagine" that som visualization is somehow displayed – a display that, at any time before it is closed, displays the current, "state" of the account – as it may be changed due to deposits, withdrawals, etc.

4.4.2.6.1 Command

75. In order to open a display of a customers' accounts with a specific bank the customer must provide the following information to a branch office: the customer's identity !

```
type
75. OpenDisplay :: CI
```

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.6 on page 92.

4.4.2.6.2 Behaviour

76. The `Open Display` behaviour
    (a) selects an appropriate bank,
    (b) and records the time –from which
    (c) it them assembles sender and receiver the time-stamped `Open Display` commands
    (d) which it then communicates to the selected branch office,
    (e) whereupon it resumes bieng a customer [behaviour].

```
value
76.  open_display(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
76a.     let boi = select_bank(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis),
76b.         τ = record TIME() in
76c.     let open_disp_cmd = ((ci,τ,boi),mk_OpenDisplay(boi)) in
76d.     comm[{ci,boi}] ! open_disp_cmd ;
76e.     customer(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,⟨open_disp_cmd ⟩⁀ch)
76.    end end
```

4.4.2.7 **Close Display**

4.4.2.7.1 Command

77. In order to close a display of a customers' accounts with a specific bank the customer must provide the following information to a branch office: the customer's identity !

```
type
77.  CloseDisplay :: CI
```

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.7 on page 93.

4.4.2.7.2 Behaviour

78. The `Close Display` behaviour
    (a) selects an appropriate bank,
    (b) and records the time –from which
    (c) it them assembles sender and receiver the time-stamped `Close Display` command –
    (d) which it then communicates to the selected branch office,
    (e) whereupon it resumes bieng a customer [behaviour].

```
type
78.   close_display(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
78a.     let boi = select_bank(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis),
78b.         $\tau$ = record TIME() in
78c.     let close_disp_cmd = ((ci,$\tau$,boi),mk_CloseDisplay(boi)) in
78d.     comm[{ci,boi}] ! close_disp_cmd ;
78e.     customer(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,⟨close_disp_cmd ⟩⁀ch)
78.      end end
```

4.4.2.8 **Obtain Credit/Debit Card**

4.4.2.8.1 Command

79. The
    (a)
    (b)
    (c)

type
79.
79a.
79b.
79c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.2 on page 108.

4.4.2.8.2 Behaviour

80. The
    (a)
    (b)
    (c)

type
80.
80a.
80b.
80c.

4.4.2.9 **Credit/Debit**

4.4.2.9.1 Command

81. The

    (a)

    (b)

    (c)

```
type
81.
81a.
81b.
81c.
```

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.8 on page 94.

4.4.2.9.2 Behaviour

82. The

    (a)

    (b)

    (c)

```
type
82. credit_debit(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch) ≡
82a.
82b.
82c.
```

4.4.2.10 **Close Credit/Debit Account**

4.4.2.10.1 Command

83. The

    (a)

    (b)

    (c)

type
83.
83a.
83b.
83c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.3 on page 109.

4.4.2.10.2 Behaviour

84. The

    (a)

    (b)

    (c)

type
84.
84a.
84b.
84c.

4.4.2.11 **Open Payment Service**

4.4.2.11.1 Command

85. The

    (a)

    (b)

    (c)

type
85.
85a.
85b.
85c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.4 on page 110.

4.4.2.11.2 Behaviour

86. The

    (a)

    (b)

    (c)

type
86.
86a.
86b.
86c.

4.4.2.12 **Close Payment Service**

4.4.2.12.1 Command

87. The

    (a)

    (b)

    (c)

type
87.
87a.
87b.
87c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.5 on page 111.

4.4.2.12.2 Behaviour

88. The

    (a)

    (b)

    (c)

value
88.
88a.
88b.
88c.

4.4.2.13 **Open Deposit Service**

4.4.2.13.1 Command

89. The

    (a)

    (b)

    (c)

type
89.
89a.
89b.
89c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.6 on page 112.

4.4.2.13.2 Behaviour

90. The

    (a)

    (b)

    (c)

value
90.
90a.
90b.
90c.

4.4.2.14 Close Deposit Service

4.4.2.14.1 Command

91. The

    (a)

    (b)

    (c)

type
91.
91a.
91b.
91c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.7 on page 113.

4.4.2.14.2 Behaviour

92. The

    (a)

    (b)

    (c)

type
92.
92a.
92b.
92c.

4.4.2.15 **Open Loan**

4.4.2.15.1 Command

93. The
    (a)
    (b)
    (c)

type
93.
93a.
93b.
93c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.9 on page 115.

4.4.2.15.2 Command

94. The
    (a)
    (b)
    (c)

value
94.
94a.
94b.
94c.

4.4.2.16 Amortize Loan

4.4.2.16.1 Command

95. The
    (a)
    (b)
    (c)

type
95.
95a.
95b.
95c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 5.5.2.9 on page 95.

4.4.2.16.2 Behaviour

96. The
    (a)
    (b)
    (c)

type
96.
96a.
96b.
96c.

4.4.2.17 **Increase Loan**

4.4.2.17.1 Command

97. The
    (a)
    (b)
    (c)

type
97.
97a.
97b.
97c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.10 on page 116.

4.4.2.17.2 Behaviour

98. The
    (a)
    (b)
    (c)

type
98.
98a.
98b.
98c.

4.4.2.18 **Close Loan**

4.4.2.18.1 Command

99. The
    (a)
    (b)
    (c)

type
99.
99a.
99b.
99c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.11 on page 117.

4.4.2.18.2 Behaviour

100. The
    (a)
    (b)
    (c)

type
100.
100a.
100b.
100c.

4.4.2.19 **Buy Stocks or Bonds**

4.4.2.19.1 Command

101. The

    (a)

    (b)

    (c)

type
101.
101a.
101b.
101c.

4.4.2.19.2 Behaviour

102. The

    (a)

    (b)

    (c)

type
102.
102a.
102b.
102c.

4.4.2.20 **Sell Stocks or Bonds**

4.4.2.20.1 Command

103. The

    (a)

    (b)

    (c)

type
103.
103a.
103b.
103c.

4.4.2.20.2 Behaviour

104. The

    (a)

    (b)

    (c)

type
104.
104a.
104b.
104c.

4.4.2.21 Include/Exclude Debitor/Creditor

4.4.2.21.1 Command

105. The
    (a)
    (b)
    (c)

type
105.
105a.
105b.
105c.

4.4.2.21.2 Behaviour

106. The
    (a)
    (b)
    (c)

type
106.
106a.
106b.
106c.

4.4.2.22 **Change Account Status**

4.4.2.22.1 Command

107. The

    (a)

    (b)

    (c)

type
107.
107a.
107b.
107c.

4.4.2.22.2 Behaviour

108. The

    (a)

    (b)

    (c)

type
108.
108a.
108b.
108c.

4.4.2.23 **Close Account**

4.4.2.23.1 Command

109. The

    (a)

    (b)

    (c)

type
109.
109a.
109b.
109c.

The pro-active command [above] and the behaviour [next] expects a response, see command/behaviour Sect. 6.5.2.16 on page 122.

4.4.2.23.2 Behaviour

110. The

    (a)

    (b)

    (c)

value
110.
110a.
110b.
110c.

4.4.3 Customer Re-active Behaviours

4.4.3.1 The `handle` Behaviour

The `handle` behaviour "takes care of", i.e., "handles", replies, i.e., messages (**msg**), from other banking behaviours.

111. It does so by inquiring as to the type of the reply messages:
    (a)
    (b)
    (c)
    (d)
    (e)
    (f)
    (g)
    (h)
    (i)
    (j)

value

```
111.   handle_cust_reacts(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)(msg)
111a.     is_...(msg) →
111b.     is_...(msg) →
111c.     is_...(msg) →
111d.     is_...(msg) →
111e.     is_...(msg) →
111f.     is_...(msg) →
111g.     is_...(msg) →
111h.     is_...(msg) →
111i.     is_...(msg) →
111j.     _ → customer(ci)(bois,bi)(info)(inc,cash,as,dis,kis,lia,tax,banks,accs,cards,dis,...,ch)
```

4.4.3.2 ur2

4.4.3.2.1 Command

4.4.3.2.2 Behaviour

4.4.3.3 **ur3**

4.4.3.3.1 Command

4.4.3.3.2 Behaviour

4.4.3.4 **ur4**

4.4.3.4.1 Command

4.4.3.4.2 Behaviour

4.4.3.5 **ur5**

4.4.3.5.1 Command

4.4.3.5.2 Behaviour

4.5 Discussion

Chapter 5

Banks

Contents

5.1 **Introduction** **70**

5.2 **Attributes** **72**

5.2.1 Types 72

5.2.2 Wellformedness 73

5.3 **Banking System Communication** **74**

5.4 **Bank Commands** **75**

5.5 **Bank Behaviours** **76**

5.5.1 The Bank Main Behaviour 76

5.5.2 The Bank Re-active Behaviours 77

5.5.2.1 **New Account Number** 78

5.5.2.1.1 **Message** 78

5.5.2.1.2 **Behaviour** 79

5.5.2.2 **Receive Deposit** 80

5.5.2.2.1 **Command:** 80

5.5.2.2.2 **Behaviour** 81

5.5.2.3 **Deliver Withdrawal** 82

5.5.2.3.1 **Command:** 82

5.5.2.3.2 **Behaviour** 83

5.5.2.4 **Effect Transfer** 84

5.5.2.4.1 **Command:** 84

5.5.2.4.2 **Behaviour** 85

5.5.2.4.2.1 **Nil Transfer:** 86

5.5.2.4.2.2 **Same Customer, Same Bank Transfer:** 87

5.5.2.4.2.3 **Different Customers, Same Bank Transfer:** 88

5.5.2.4.2.4 **Same Customer, Different Banks Transfer:** 89

5.5.2.4.2.5 **Different Customers, Different Banks Transfer:** 90
5.5.2.5 **Exchange** 91
5.5.2.6 **Open Display** 92
5.5.2.6.1 **Command** 92
5.5.2.6.2 **Behaviour** 92
5.5.2.7 **Close Display** 93
5.5.2.7.1 **Command** 93
5.5.2.7.2 **Behaviour** 93
5.5.2.8 **Effect Credit/Debit** 94
5.5.2.8.1 **Command** 94
5.5.2.8.2 **Behaviour** 94
5.5.2.9 **Effect Amortization** 95
5.5.2.9.1 **Command** 95
5.5.2.9.2 **Behaviour** 95
5.5.3 The Bank Pro-active Behaviours 96
5.5.3.1 **Command** 96
5.5.3.2 **Behaviour** 96
5.6 **Discussion** **97**

5.1 Introduction

By a bank we shall here, in a restricted sense mean an aggregate of: a set of one or more **local banks**, i.e., **branch offices**, and a (i.e., one) **headquarter**, Bank customers "deal" with the bank in two ways

- through a branch office for other operations than cash deposits and withdrawals, and
- with the bank for cash deposits and withdrawals, debits and credits, etc.

Customers do not deal with bank head quarters.

So branch offices are where You go to

- open and close accounts,
- obtain and close credit/debit cards,
- apply for, open, increase and close loans,
- buy and sell stocks and bonds, and
- change account states.

So the concept of a bank has several "meanings": the physical embodiment of the "mortar & brick" branch offices (and, for that matter, their head quarters), and the mental notion of a bank: the three concepts outlined in this chapter.

• • •

We have already, in Chapter 3, covered

- *endurants*, 3.3.2 on page 16, and
- *unique identification*, 3.3.3.1 on page 19, and
- *mereology*, 3.3.3.2 on page 24, of banks.

5.2 Attributes

5.2.1 Types

112. Banks have [static attribute] administrative information (name, address, ...).

113. Banks keep track of customer accounts [programmable attribute].

    (a) There is a customer account table. It relates customer identifiers to account information [programmable attribute].

    (b) Account numbers are further unspecified.

    (c) Account information consists of customer name, [approximate recent] salary, [approximate recent] assets, etc.

114. For every account there is a set of one or more currency balances [programmable attribute]

    (a) where a currency balance is a natural number.

115. An `account history` lists the time-stamped deposits and withdrawals.

116.

117. Banks keep cash of various categories,

    (a) incl. denominations, f.ex. currencies.

118. Banks also have a history [programmable attribute] .

type
112. AdmInfo = ...
113. CustAccts = CI $\xrightarrow[m]{}$ CustAcct
113a. CustAcct = AcctNu-**set**
113b. AcctNu = ...
113c. AcctInfo = Name×Salary×Assets×...
114. Accounts = AcctNu $\xrightarrow[m]{}$ (Currency $\xrightarrow[m]{}$ Balance)
114a. Balance = **Nat**
115. AcctHist = ($\mathbb{TIME}$×Cash×(Dep|With))*
116. Dep = **"deposit"**, With = **"withdrawal"**
117. Cashs = CashCat $\xrightarrow[m]{}$ **Nat**
117a. CashCat = ″`cashcat1`″|″`cashcat2`″|″`cashcat3`″|...
118. BHist = (...)*
value
112. **attr**_AdmInfo: B → AdmInfo
113. **attr**_CustAccts: B → Accounts
114. **attr**_Accounts: B → Account
115. **attr**_AcctHist: B → AcctHist
117. **attr**_Cashs: B → Cashs

118. **attr**_BankHist: B → BankHist

113b. xtr_CI: AcctNu → CI, xtr_BI: AcctNu → BI

5.2.2 Wellformedness

119.

119.

5.3 Banking System Communication

120. There is a global communications medium that allows behaviours of identities `ui` and `uj` to synchronize and communicate, offer (**!**), respectively accept (**?**) messages.

121. These messages are typed.

```
120. channel
120.     comm { [{ui,uj}] | ui,uj:UI • {ui,uj} ⊆ ... } : MSG
121. type
121.     MSG = ...
```

We have defined `MSG`, “bit-by-bit” in this and other chapters !

5.4 Bank Commands

Banks responds to commands received from account holders, branch offices or the bank headquarter. Banks do not, in our model, by themselves issue commands. [Compare this opening paragraph with those of Sects. 4.3 on page 35 and 6.5 on page 104.]

122. Banks issue, in response to branch office and customer commands, that is: re-actively, the following (presently 8) commands:

    (a) **New Account Number:** A branch office alerts the bank: to assign a new, hitherto unused, account number of some identified currency, to a customer and set its balance to the name, `ci:CI`, of that customer, to 0.

    (b) **Receive Deposit:** Bank recognizes the deposit and, hence, deposit of an amount of cash into a customer account, and informs of new balance. This command is in response to the customer `Deposit` command, cf. ι60b π35.

    (c) **Delivers Withdraw:** Bank recognizes the request and, hence, conditionally withdraws of an amount of cash from a customer account, subject to the new balance not exceeding the lower [contracted] limit, and informs of new balance. This command is in response to the customer `Withdraw` command, cf. ι60c π35.

    (d) **Effect Transfer:** Bank recognizes the request and, hence, conditionally withdraws an amount of cash from a customer account, subject to the new account balance not exceeding the lower [contracted] limit, and then, conditionally, transfers that amount to an[other] account [of possibly another customer, in possibly the same or another bank], and informs of new balance. This command is in response to the customer `Transfer` command, cf. ι60d π35.

    (e) **Present Display:** The bank replies with an image of the named account – and refreshes that image whenever a "movement" (i.e., a change of values on that account), the refreshed image being sent [also] to the customer. This command is in response to the customer `OpenDisplay` command, cf. ι60f π35.

    (f) **Abort Display:** The bank cease to refresh the account image – by simply "closing down" the "imagery" ! This command is in response to the customer `CloseDisplay` command, cf. ι60g π36.

    (g) **Effect Credit/Debit:** The bank informs the customer of each "movement" on the account due to successful credit/debit requests. This command is in response to the customer `CreditDebit` command, cf. ι60i π36.

    (h) **Effect Amortization of Loan[s]:** The bank withdraws interest and annuity on one or more loans and informs the customer.

    (i) **Account Closed:** The bank closes an account in response to a customer close account request. It also closes outstanding credit/debit cards, loans, payment and deposit services.

123. At this stage of the development of this banking domain model it seems that there is no need for banks to pro-actively issue commands.

```
type
122.  BCmd =                          122e.      PresDisp
122a.        NewAcctNu                122f.      AbortDisp
122b.        RcvDep                   122g.      EffDebCred
122c.        DlvWith                  122h.      EffAmort
122d.        EffectXfer               122i.      EffClose
```

5.5 Bank Behaviours

5.5.1 The Bank Main Behaviour

124. The bank behaviour internal non-deterministically (⌈⌉) alternates between

    (a) external non-deterministically responding handling responses from money institutions, and

    (b) pro-active actions: issuing commands to banks, branch offices, etc., and reactive actions: responding to messages from banks, branch office, etc.

```
type
124.  bank: BOI→BM→AdminInfo→CustAccts×Accounts×AcctHist×Cashs×BHist Unit
value
124.  bank(boi)(bm)(...)(custaccts,accts,acthist,cashs,bhist) ≡
124a.  ⌈⌉ re_active_bank(boi)(bm)(...)(custaccts,accts,acthist,cashs,bhist)
124b.     pro_active_bank(boi)(bm)(...)(custaccts,accts,acthist,cashs,bhist)
```

5.5.2 The Bank Re-active Behaviours

125. The `re_active_bank` behaviour

    (a) external non-deterministically ⌷ offers to accept **comm**unications from customers, *cis*, and branch offices, *bois*.

    These communications are either:

    (b) `new account numbers`,

    (c) `deposits`,

    (d) `withdrawals`,

    (e) `transfefs`,

    (f) `open displays`,

    (g) `close displays`,

    (h) `credit/debits`, or

    (i) `loan amortizations`,

    in which cases corresponding behaviours are invoked, or, if none of these,

    (j) the `re_active_bank` behaviour "reverts" to being a, i.e., the, `bank` behaviour.

```
value
125.  re_active_bank(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist) ≡
125a.     let ((ci,τ,bi),msg) = ⌷ { comm[{ui,bi}] ? | ui ∈ cis∪bois } in
125b.     is_NewAcctNu(msg) →
125b.         bank_new_acct(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)(bi)((ci,τ,bi),msg),
125c.     is_Deposit(msg) →
125c.         bank_rcv_dep(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)(bi)((ci,τ,bi),msg),
125d.     is_Withdraw(msg) →
125d.         bank_del_with(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)((ci,τ,bi),msg),
125e.     is_Transfer(msg) →
125e.         bank_effect_xfer(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)((ci,τ,bi),msg),
125f.     is_OpenDisplay(msg) →
125f.         bank_open_disp(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)((ci,τ,bi),msg),
125g.     is_CloseDisplay(msg) →
125g.         bank_coose_disp(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)((ci,τ,bi),msg),
125h.     is_CreditDebit(msg) →
125h.         eff_deb_cre_bank(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)((ci,τ,bi),msg),
125i.     is_AmortizeLoan(msg) →
125i.         bank_amortize_loan(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)(bi)((ci,τ,bi),msg),
125j.     _ → bank(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)(bi)
125.    end
```

5.5.2.1 New Account Number

The `new account number` bank action is in response to the branch office request, Sect. 6.5.2.1 on page 106, ι159d π106, for a new account.

5.5.2.1.1 Message

126. The bank response is in the form of a message which contains just a new account number.

type
126. NewAcctMsg :: CI × AcctNu

5.5.2.1.2 **Behaviour**

127. The `bank` behaviour has received a request fora new [currency] account (`NewAcctNu(ci,boi,curr)`).
    (a) The bank creates an hitherto unused account number.
    (b) Updates the `customer accounts` (record) for that customer with that new account number.
    (c) Updates the bank `accounts` with that new accounts number to show a (currency) balance of zero.
    (d) Records the time.
    (e) Assembles a reply message.
    (f) Communicates that reply message for that customer to the branch office.
    (g) And resumes being a bank.

value

```
127.   bank_new_acct(bi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)(bi)
127.                                (request:((ci,t,boi),mk_NewAcctNu(ci,boi,curr))) ≡
127a.      let an:AcctNu • an ∉ dom accts in
127b.      let custaccts′ = custaccts † [ci↦(custaccts(ci))∪{anu}] in
127c.      let accts′ = accts † [ anu ↦ [curr ↦ 0 ] ],
127d.          τ = record 𝕋𝕀𝕄𝔼()  in
127e.      let reply = ((bi,τ,boi),mk_NewAcctMsg(ci,anu)) in
127f.      comm[{bi,boi}] ! mk_NewAcctMsg(ci,anu) ;
127g.      bank(bi)(bm:(cis,bois,hqi))(...)(custaccts′,accts′,cashs,⟨reply,request⟩⌢bhist)(bi)
127.       end end end end
```

5.5.2.2 **Receive Deposit**

5.5.2.2.1 **Command:** The bank receives a request to deposit an amount of cash in some currency in some account (of that currency).

128. It replies with a message informing of the new balance of that account. That message contains:

    - deposit time,
    - customer identifier,
    - identified account,
    - amount of deposit,
    - currency,
    - old and
    - new balance.

type

128. DepositOK :: $\mathbb{TIME} \times$ CI $\times$ AcctNu $\times$ Cash $\times$ Curr $\times$ Balance $\times$ Balance

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.2 on page 40.

5.5.2.2.2 **Behaviour**

129. The bank receives a `deposit` request.
    (a) Records the time and notes the depositing customer[32].
    (b) The bank updates its account for that customer.
    (c) The bank updates its account history for that customer, and
    (d) its cashs.
    (e) Assembles a reply message.
    (f) Communicates a `DepositOK` to the customer.
    (g) Resumes being a bank.

    Prerequisites for the proper handling of the customer's deposit request are:

    (h) the customer is recorded as having an account of the indicated currency, and the bank's `cashs` is recorded as containing the indicated currency.

```
value
129.  bank_rcv_dep(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,bhist)
129.              ((ui,τ,bi),mk_Deposit(bi,anu,curr,amount)) ≡
129a.   let τ = record 𝕋𝕀𝕄𝔼(), ci = xtr_CI(anu) in
129b.   let accts′ = acctts † [ ci ↦ [curr ↦ (accts(ci))(curr)+amount ]],
129c.       accthist′ = ⟨(τ,amount,"deposit")⟩⌢accthist,
129d.       cashs′ = cashs † [ curr ↦ cashs(curr) + amount ],
129e.       reply = mk_DepositOK(τ,ci,anu.curr,(accts(ci))(curr),(accts(ci))(curr)+)amount in
129f.    comm[{bi,ui}] ! reply ;
129g.    bank(bi)(bois,bi)(info)(custaccts,accts′,accthist′,cashs′,⟨((bi,τ,ci),reply)⟩⌢bhist) end end
129h.    pre: ci ∈ dom accts ∧ curr ∈ dom accts(ci) ∧ curr ∈ dom cashs
```

[32] – as the depositors may be a bank, as in a `transfer`

5.5.2.3 Deliver Withdrawal

5.5.2.3.1 **Command:** The bank has received a withdraw request from a customer. That request, as it is for all such customer-banking requests, is in the form of a command, cf. ι69e π41. The bank responds with either of two messages.

130. These two messages, `WithdrawResponse`, are either an OK reply with the cash or a Not OK reply:

    (a) The `OK_WDR` reply contains

    - the account number,
    - the currency,
    - cash, and
    - new balance.

    (b) The `NOK_WDR` reply contains

    - the account number,
    - the currency,
    - the requested amount of cash and
    - the current balance.

```
type
130.    WithDrawResp = OK_WDR | NOK_WDR
130a.   OK_WDR   :: AccNu × Curr × Cash × NewBal
130b.   NOK_WDR :: Amount × Curr × Amount × CurrBal
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.3 on page 41.

5.5.2.3.2 Behaviour

131. The bank response to a `withdraw` request is to

    (a) record time;

    (b) question whether there ace sufficient funds to honour the withdrawal.

    (c) If not, then concoct a suitable reply;

    (d) communicate this to the customer and

    (e) resume being a bank.

    (f) If indeed, then update customer's account,

    (g) the bank's cashs, and a

    (h) reply –

    (i) which is communicated to the customer –

    (j) whereupon the bank resumes being a bank.

    Prerequisites for the proper handling of the customer withdrawal request are:

    (k) the customer is recorded as having an account of that currency and the bank's `cashs` is recorded as containing the indicated currency.

```
value
131.  bank_del_with(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,bhist)
131.            ((ci,τ,bi),mk_Withdraw(bi,anu,(curr,amount))) ≡
131a.    let τ = record 𝕋𝕀𝕄𝔼() in
131b.    if (accts(ci))(curr)<amount
131c.      then let reply = ((bi,τ,ci),mk_NOK_WDR(anu,curr,amount,(accts(ci))(curr))) in
131d.           comm[{bi,ci}] ! reply ;
131e.           bank(bi)(bois,bi)(info)(custaccts,accts,⟨reply⟩⌢accthist,cashs,⟨reply⟩⌢bhist) end
131f.      else let accts′ = acctts † [ ci ↦ [curr ↦ (accts(ci))(curr) − amount ]] ,
131g.               cashs′ = cashs † [ curr ↦ cashs(curr) − amount ] ,
131h.               reply =  ((bi,τ,ci),mk_OK_WDR(anu,curr,amount,(accts(ci))(curr) − amount )) in
131i.           comm[{bi,ci}] ! reply ;
131j.           bank(bi)(bois,bi)(info)(custaccts,accts′,⟨reply⟩⌢accthist,cashs′,⟨reply⟩⌢bhist) end
131.     end end
131k.  pre: ci ∈ dom accts ∧ curr ∈ accts(ci) ∧ curr ∈ dom cashs
```

5.5.2.4 **Effect Transfer**

One may consider a `transfer` *in either of two ways:*

- **either** *as a transfer of cash-o-hand, that is, as a pair of* `withdrawals` *and* `deposits`*, both initiated by the transferring customer,*
- **or** *as a transfer in two parts:*
  1. *first the transferring customer commands its bank to withdraw and amount from an identified account,*
  2. *whereupon the transferring [customers] bank commands the "transferred to" customers bank to receive a deposit.*

We shall opt for the latter form.

5.5.2.4.1 **Command:** The bank has received a transfer request from a customer. That request, as it is for all such customer-banking requests, is in the form of a command, cf. ι72 π44. The bank responds with either of two messages. There is either insufficient funds, or there are sufficient funds for the transfer. The transfer is from one customers account to another account of the same customer or another customer. The bank response to the transferring customer is either one of insufficient funds or an OK, i.e., sufficient funds.

132. The two kinds of bank replies are:

    (a) `TransferNOK` which consists of

    - `Transfer` and
    - **"not ok"**; and

    (b) `TransferOK` which consists of

    - `Transfer` and
    - **"ok"**;

type
132. TransferReply = TransferNOK | TransferOK
132a. TransferNOK :: Transfer × **"not ok"**
132b. TransferOK :: Transfer × **"ok"**

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.4 on page 43.

5.5.2.4.2 Behaviour

133. The bank response to a customer transfer request is to test the following cases:
    (a) Are there sufficient funds to transfer ?

    If so, then

    (b) the transfer is between the same customer's [different] accounts ?
    (c) the transfer is between the same customer's [different, and different bank] accounts ?
    (d) the transfer is between two different customers of the same bank ?
    (e) the transfer is between two different customers of different banks ?
    (f) otherwise the bank resumes being the bank.

In all of these cases the transferring bank performs the expected operations – as outlined in respective actions.

```
value
value
133.  bank_effect_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,bhist)
133.            (msg:((ci,t,bi),mk_Transfer((ci,anu,curr,amount),(ci′,anu′,curr,amount)))) ≡
133a.     (accts(anu))(curr) ≤ amount
133a.         → nil_xfer(bi)(bois,bi)(info)
133a.                 (custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)(msg),
133b.     ci=ci′ ∧ xtr_BI(anu)=xtr_BI(anu′)
133b.         → same_cust_same_bank_xfer(bi)(bois,bi)(info)
133b.                 (custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)(msg),
133c.     ci=ci′ ∧ xtr_BI(anu)≠xtr_BI(anu′)
133c.         → same_cust_diff_banks_xfer(bi)(bois,bi)(info)
133c.                 (custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)(msg),
133d.     ci≠ci′ ∧ xtr_BI(anu)=xtr_BI(anu′)
133d.         → diff_cust_same_bank_xfer(bi)(bois,bi)(info)
133d.                 (custaccts,accts,accthist,cashs,⟨msh⟩⌢bhist)(msg),
133e.     xtr_BI(anu)=xtr_BI(anu′)
133e.         → diff_custs_diff_banks_xfer(bi)(bois,bi)(info)
133e.                 (custaccts,accts,accthist,cashs,⟨msh⟩⌢bhist)(msg)
133f.     _ →  bank(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,bhist)
```

As it turns out: the two pairs:

- `same_cust_same_bank_xfer` and `same_cust_diff_banks_xfer`, and
- `diff_cust_same_bank_xfer` and `diff_custs_diff_banks_xfer`

are, in effect, of the same form.

The five `tranfer` sub-actions are:

- 5.5.2.4.2.1. nil_xfer Page 86
- 5.5.2.4.2.2. same_cust_same_bank_xfer Page 87
- 5.5.2.4.2.3. diff_custs_same_bank_xfer Page 88
- 5.5.2.4.2.4. same_cust_diff_banks_xfer Page 89
- 5.5.2.4.2.5. diff_custs_diff_banks_xfer Page 90

• • •

5.5.2.4.2.1 **Nil Transfer:** Customer cash is insufficient.

134. For the **nil_xfer**,
    (a) if that the customer balance for the identified currency account is not sufficient for the transfer,
    (b) then reply to that effect is assembled and communicated –
    (c) whereupon the bank resumes being a bank.

```
value
134.    nil_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)
134.                (msg:((ci,t,bi),mk_Transfer((ci,anu,curr,amount),(ci′,anu′)))) ≡
134a.       let reply = ((bi,τ,ci),mk_TransferNOK(msg,sort{″not ok″})) in
134b.       comm[{ci,bi}]reply ;
134c.       bank(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨reply,msg⟩⌢bhist)
134.        end

134a.       pre: anu ∈ dom accts ∧ curr ∈ dom (accts(anu)) ∧  (accts(anu))(curr) < amount
```

5.5.2.4.2.2 **Same Customer, Same Bank Transfer:** Transfer is between two different accounts, but same customer at same bank.

135. The **same_cust_same_bank_xfer** action
    (a) updates the two [same] customer accounts,
    (b) records time,
    (c) assembles a `reply` message,
    (d) and communicates this [back] to the customer,
    (e) whereupon it resumes being that [same] customer.
    (f) Preconditions are: the accounts are of the same customer, and are commensurate with the currency, and customer has enough currency to transfer.

```
value
135.  same_cust_same_bank_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)
135.          (msg:((ci,t,bi),mk_Transfer((f_ci,f_anu,curr,amount),(t_ci,t_anu)))) ≡
135a.     let accts′ = accts  † [ f_anu ↦ [ curr ↦ (accts(f_anu))(curr)  − amount ] ]
135a.                         † [ t_anu ↦ [ curr ↦ (accts(t_anu))(curr) + amount ] ] ,
135b.         τ = record 𝕋𝕀𝕄𝔼() in
135c.     let reply = ((bi,τ,f_ci),mk_TransferOK(msg,"ok")) in
135d.     comm[{f_ci,bi}] ! reply ;
135e.     bank(bi)(bois,bi)(info)(custaccts,accts′,accthist,cashs,⟨reply,msg⟩⌢bhist)
135.      end end

135f.     pre: f_ci=t_ci ∧ xtr_BI(f_anu)=xtr_BI(t_anu)
135f.         ∧ f_anu ∈ dom accts ∧ curr ∈ dom (accts(f_anu))
135f.         ∧ t_anu ∈ dom accts ∧ curr ∈ dom (accts(t_anu))
135f.         ∧ (accts(anu))(curr) ≥ amount
```

5.5.2.4.2.3 **Different Customers, Same Bank Transfer:** Transfer is between two different customer accounts in same bank.

As it turns out – some simple reasoning could have show that – the two actions are essentially of the same form, The two actions: **same_cust_same_bank_xfer** *and this, the* **diff_custs_same_bank_xfer** *action, could be formulated as one.*

136. The **diff_custs_same_bank_xfer** action:
    (a) updates the two [different] customers accounts,
    (b) records time,
    (c) assembles a `reply` message,
    (d) and communicates this [back] to the transferring customer,
    (e) whereupon it resumes being the transferring customer.
    (f) Preconditions are: the transferring customer's account balance is larger than or equal to the amount to be transferred, the two customers are different and their banks are the same.

```
value
136.   diff_custs_same_bank_xfer(bi)(bois,bi)(info)(cust_accts,accts,acct_hist,cashs,bhist)
136.                  (((ci,t,bi),mk_Transfer((f_ci,f_anu,curr,amount),(t_ci,t_anu)))) ≡
136a.      let accts″ = accts′  † [ anu ↦ [ curr ↦ (accts(anu))(curr)  − amount ] ]
136a.                           † [ anu′ ↦ [ curr ↦ (accts(anu′))(curr) + amount ] ] ,
136b.          τ = record 𝕋𝕀𝕄𝔼() in
136c.      let reply = ((bi,τ,ci),mk_TransferOK(msg,"ok")) in
136d.      comm[{ci,bi}] ! reply ;
136e.      bank(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨reply,msg⟩⌢bhist)
136.       end end

136f.     pre:  f_ci≠t_ci ∧ xtr_BI(f_anu)=xtr_BI(t_anu)
136f.         ∧ f_anu ∈ dom accts ∧ curr ∈ dom (accts(f_anu))
136f.         ∧ t_anu ∈ dom accts ∧ curr ∈ dom (accts(t_anu))
136f.         ∧ (accts(f_anu))(curr) ≥ amount
```

5.5.2.4.2.4 **Same Customer, Different Banks Transfer:** Transfer is between two different customers' accounts in different banks.

The description lets the transferring customer's bank handle the customer's account, as a "withdrawal", and, through a "deposit command", lets the "transferred to", i.e., receiving customer's bank, handle the "deposit".

137. The **same_cust_diff_banks_xfer** action:

    (a) The transferring customer's bank adjusts its accounts,

    (b) records time,

    (c) assembles a deposit command

    (d) which is then communicated to the "transferred to" bank account;

    (e) from which it then awaits a reply message;

    (f) if the reply is **"ok"** then a successful transfer is completed and the transferring bank reverts to being a bank with an updated customer account.

    (g) else an un-successful transfer is completed ("terminated") and the transferring bank reverts to being a bank with no updated customer account.

```
value
137.  same_cust_diff_banks_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨msg⟩⌢bhist)
137.              (msg:((f_ci,t,bi),mk_Transfer((f_ci,f_anu,curr,amount),(t_ci,t_anu)))) ≡
137a.     let accts′ = accts † [f_anu ↦ (accts(f_anu))(curr) − amount ] ,
137b.         τ = record 𝕋𝕀𝕄𝔼() in
137c.     let fwd_cmd = ((bi,τ,xtr_BI(t_anu)),mk_Deposit(xtr_BI(t_anu),t_anu,curr,amount)) in
137d.     comm[{bi,xtr_BI(t_anu)}] ! fwd_cmd ;
137e.     let reply:(_,Transfer(ok_or_nok)) = comm[{bi,xtr_BI(t_anu)}] ? in
137e.     case Transfer(ok_or_nok) of
137f.         "ok" →
137f.             bank(bi)(bois,bi)(info)(custaccts,accts′,⟨reply,fwd_cmd⟩⌢accthist,cashs,⟨msg⟩⌢bhist)
137g.         "nok" →
137g.             bank(bi)(bois,bi)(info)(custaccts,accts,⟨reply,fwd_cmd⟩⌢accthist,cashs,⟨msg⟩⌢bhist)
137.      end end end end

137g.     pre: (accts(f_anu))(curr) ≥ amount ∧ f_ci=t_ci ∧ xtr_BI(f_anu)≠xtr_BI(t_anu)
```

5.5.2.4.2.5 **Different Customers, Different Banks Transfer:** Transfer is between two different customers accounts in different banks.

138. As for the `same_cust_same_bank_xfer` and `diff_custs_same_bank_xfer` action descriptions, the `diff_custs_diff_banks_xfer` action description,

139. is, in form, the same as the `same_cust_diff_banks_xfer` action description:

value

```
138.  diff_custs_diff_banks_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨msg⟩^bhist)
138.            (((ci,t,bi),mk_Transfer((f_ci,t_anu,curr,amount),(t_ci,t_anu)))) ≡
139.    same_cust_diff_banks_xfer(bi)(bois,bi)(info)(custaccts,accts,accthist,cashs,⟨msg⟩^bhist)
139.            (((f_ci,t,bi),mk_Transfer((f_ci,f_anu,curr,amount),(t_ci,t_anu))))
```

5.5.2.5 Exchange

TO BE WRITTEN

5.5.2.6 Open Display

5.5.2.6.1 Command

140. The

    (a)

    (b)

    (c)

    (d)

    (e)

```
type
140.
140a.
140b.
140c.
140d.
140e.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.6 on page 46.

5.5.2.6.2 Behaviour

141. The

    (a)

    (b)

    (c)

    (d)

    (e)

```
value
141.  bank_open_disp(bi)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
141.             ((ci,τ,bi),mk_OpenDisplay(_,_,_)) ≡
141a.
141b.
141c.
141d.
141e.
```

5.5.2.7 Close Display

5.5.2.7.1 Command

142. The
    (a)
    (b)
    (c)
    (d)
    (e)

```
type
142.
142a.
142b.
142c.
142d.
142e.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.7 on page 47.

5.5.2.7.2 Behaviour

143. The
    (a)
    (b)
    (c)
    (d)
    (e)

```
value
143.  bank_close_disp(bi)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
143.             ((ci,τ,bi),mk_CloseDisplay(_,_,_)) ≡
143a.
143b.
143c.
143d.
143e.
```

5.5.2.8 Effect Credit/Debit

5.5.2.8.1 Command

144. The

    (a)

    (b)

    (c)

    (d)

    (e)

type

```
144.
144a.
144b.
144c.
144d.
144e.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.9 on page 49.

5.5.2.8.2 Behaviour

145. The

    (a)

    (b)

    (c)

    (d)

    (e)

```
145a.   eff_deb_cre_bank(bi)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
145a.                   ((ci,τ,bi),mk_DebitCredit(_,_,_)) ≡
145b.
145c.
145d.
145e.
```

5.5.2.9 Effect Amortization

5.5.2.9.1 Command

146. The

(a)

(b)

(c)

(d)

(e)

146.
146a.
146b.
146c.
146d.
146e.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.16 on page 56.

5.5.2.9.2 Behaviour

147. The

(a)

(b)

(c)

(d)

(e)

```
value
147.   bank_amort(bi)(bois,bi)(info)(inc,as,lia,tax,banks,accs,cards,dis,...,ch)
147.             ((ci,τ,bi),mk_AmorizeLoan(_,_,_)) ≡
147a.
147b.
147c.
147d.
147e.
```

5.5.3 The Bank Pro-active Behaviours

5.5.3.1 Command

148. Since there is, at the moment of this being written, 24.6.2025, 10:40 am, no pro-active commands of the bank,

    (a) that bank resumes being a bank.

5.5.3.2 Behaviour

```
value
148.   pro_active_bank(boi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist) ≡
148a.      bank(boi)(bm:(cis,bois,hqi))(...)(custaccts,accts,acthist,cashs,bhist)
```

5.6 Discussion

Chapter 6

Branch Offices

Contents

6.1 **External Qualities** **100**

6.2 **Internal Qualities** **100**

6.2.1 Unique Identification 100

6.2.2 Mereology 100

6.2.3 Attributes 101

6.2.4 Wellformedness 101

6.3 **Branch Office Intentional Pull** **102**

6.4 **Branch Office Commands** **103**

6.5 **Branch Office Behaviours** **104**

6.5.1 Main Branch Office Behaviour 104

6.5.2 Branch Office Re-active Behaviours 105

6.5.2.1 Open Account 106

6.5.2.2 Open Debit/Credit Card 108

6.5.2.3 Close Debit/Credit Card 109

6.5.2.4 Open Payment 110

6.5.2.5 Close Payment 111

6.5.2.6 Open Deposit 112

6.5.2.7 Close Deposit 113

6.5.2.8 Close Account 114

6.5.2.9 Open Loan 115

6.5.2.10 Increase Loan 116

6.5.2.11 Close Loan 117

6.5.2.12 Buy Stocks or Bonds 118

6.5.2.13 Sell Stocks or Bonds 119

6.5.2.14 Incl./Excl. Deb,/Cre. 120

6.5.2.15 Change Account 121

6.5.2.16 Close Account 122

6.5.3 Branch Office Pro-active Behaviours 123

6.5.3.1 Information From Bank 124
6.5.3.2 XXX 125
6.5.3.3 YYY 126
6.5.3.4 ZZZ 127
6.5.3.5 WWW 128

We remind the reader that, for *local banks*, we distinguish between their head-quarter and their branch offices and that we abstract the perhaps visible set of more than one branch office into one. The thus abstracted branch office services non-banking customers: You and I, businesses, etc. The head-quarter services its branch office[s] and interacts with other local banks, the central banks, etc.

6.1 External Qualities

Was treated in Sect. 3.3.2 on page 16.

6.2 Internal Qualities

6.2.1 Unique Identification

Was treated in Sect. 3.3.3.1 on page 19.

6.2.2 Mereology

Was treated in Sect. 3.3.3.2 on page 24.

6.2.3 Attributes

149. Each branch office has its own *registration identification*[33]

150. which is further undefined,

151. but define an extraction function which from branch office registration (and, see next, account) identifiers extract the branch office identifier and the bank headquarter identifier.

152. One "central" attribute is that of the *accounts*! For every account identifier there is account information.

    (a) `Account identifiers` are further unspecified tokens.
    (b) `Account information` records such things as *Balance*, *Interest Rates*, *Time-stamped List of Transactions*, etc.
    (c) *Balance* is a rational number, zero, positive or negative fractions.
    (d) *Interest Rates* are either `Positive` or `Negative`,
    (e) *Transactions* lists a sequence of commands directed at the account – such that the time-stamped transactions are ordered: most recent transactions first.
    (f) A transaction is a time-stamped command.

153. Branch offices have a history of all transactions.

type
149. RegNu
150. CustCtlg = CI $\xrightarrow[m]{}$ AccId-**set**
150. AccId
152. Accts = (AccId $\xrightarrow[m]{}$ AcctInfo)
152a. AccId
152b. AcctInfo = Bal×IntR×TransI×...
152c. Bal = **Real**
152d. IntR,PosR,NegR =**Real**
152e. TransI = Trans*

153. BOHist =Trans*
152f. Trans = $\mathbb{TIME}$×Cmd
value
151. **attr**_RegNu: BO → RegNu
151. **attr**_CustCtlg: BO → CustCtlg
152. **attr**_Accts: BO → Accounts
153. **attr**_BOHist: BO → BOHist

151. xtr_BOI_HQI: (Regnu|AccId)
151. → (BOI×HQI)

6.2.4 Wellformedness

154.

154.

[33] – to be used, for example, when customers may wish to interact with the bank

6.3 Branch Office Intentional Pull

TO BE WRITTEN

6.4 Branch Office Commands

Branch offices either responds to commands received from account holders, banks or the bank headquarter. or issue, by their own initiative [most likely in response to a directive from its headquarter], commands to bank customers – say in case of interest rate changes, etc. [Compare this opening paragraph with those of Sects. 4.3 on page 35 and 5.4 on page 75.]

155. **Commands:** Branch offices issue the following (presently 9) commands:

   (a) **Your New Account is Granted:** This command is in response to the customer `OpenAcct` command, cf. ι60a π35.

   (b) **Your New Credit/Debit Card:** This command is in response to the customer `ObtainCreDebCard` command, cf. ι60h π36.

   (c) **Your Credit/Debit Card Closed:** This command is in response to the customer `ClosCreDebCard` command, cf. ι60j π36.

   (d) **Your Loan Application:** This command is in response to the customer `Open Loan` command, cf. ι60o π36.

   (e) **Your Loan Increase:** This command is in response to the customer `IncreaseLoan` command, cf. ι60q π36.

   (f) **Your Loan Closed:** This command is in response to the customer `CloseLoan` command, cf. ι60r π36.

   (g) **Change of Account Status Accepted:** This command is in response to the customer `ChgeAcctStatus` command, cf. ι60v π36.

   (h) **Information from Your Bank:** The bank, i.e., the branch office, occasionally informs its customers, f.ex.: of changes of interest rates on deposits, loans, etc., or other.

   (i) **Your Account has been Closed:** This command is in response to the customer `CloseAcct` command, cf. ι60w π36.

```
type
155.  BOCmd =
155a.    NewAcct
155i.  | ClosAcct
155b.  | OpnCDCard
155c.  | CloCDCard
155d.  | LoanAccDen
155e.  | LoanIncrOK
155f.  | LoanClos
155g.  | AccChgAcctSta
155h.  | InfoFromBank
```

6.5 Branch Office Behaviours

6.5.1 Main Branch Office Behaviour

156. The `branch office` internal non-deterministically alternates between

    (a) re-actively responding to customer, bank and bank headquarter commands and

    (b) pro-actively issuing commands to customers, [their] bank and [their bank] headquarter.

value

```
156.  branch_office(boi)(cis,bois,hqi)(static)(monit)(progr) ≡
156a.      re_active_branch_office(boi)(cis,bois,hqi)(static)(monit)(progr)
156b.    ⌈⌉ pro_active_branch_office(boi)(cis,bois,hqi)(static)(monit)(progr)
```

6.5.2 Branch Office Re-active Behaviours

157. The re-active-branch-office behaviour external non-deterministically (⌈⌉)

   (a) awaits (i.e., responds to) messages from customers, branch offices or its head quarter.

   (b) If the message received is of type "X" then a corresponding behavior is involved.

   (c) Else the behaviour resumes being a branch office behaviour.

```
value
157.  re_active_branch_office(boi)(bom:cis,bois,hqi)(static:regnu)(progr:(custctlg,access,bohist)) ≡
157a.     let ((ci,τ,bi),msg) = ⌈⌉ { commqui,boi | ui ∈ cis_uid∪bois_uid∪{hqi} } in
157b.     is_OpenAcct(msg) → open_account_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ObtainCredDebCard(msg) → open_CD_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ClosCredDebCard(msg) → clos_CD_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_OpenPaymentService(msg) → open_payment_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ClosPaymentService(msg) → clos_payment_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_OpenDepositService(msg) → open_deposit_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_CloseDepositService(msg) → close_deposit_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ApplyLoan(msg) → apply_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_IncreaseLoan(msg) → increase_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_CloseLoan(msg) → clos_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_BuyStock(msg) → buy_stock_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_SellStock(msg) → sell_stock_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_InclExclDebCred(msg) → incl_excl_DB_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ChangeAcctStatus(msg) → change_status_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157b.     is_ClosAcct(msg) → clos_account_bo(boi)(bom)(static)(progr)((ci,τ,bi),msg)
157c.     _ → branch_office(boi)(cis,bois,hqi)(static:regnu)(progr:(custctlg,access,bohist))
157b.     end
```

6.5.2.1 Open Account

6.5.2.1.1 Command. The Open Account command, **mk**`_OpenAcct(info,inc,as,dis,kis,lia)`, is a request from a customer, `ci`. That customer expects a reply. Either in the form of a **no**, that is, we, the bank, do not accept that You open an account, given her information, `info,inc,as,dis,kis,lia`, that You have given, sorry, ot it is a **yes**, welcome to the bank, here is Your new bank account number etc. These replies amount to commands, sent by the branch office to whom the `Open Account` request was addressed. This reply command has the following form:

158. The branch office reply to a customer `open account` request is either to decline the request[34] or to accept the request:
    (a) The `DeclineAcct` reply just lists a **"no"**.
    (b) The `NewAcct` reply "lists" the bank indentifier and a new account number.

```
type
158.  OpenAcctReply = DeclineAcct | NewAcct
158a.  DeclineAcct :: "no"
158b.  NewAcct :: BI × AcctNu
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.1 on page 39.

6.5.2.1.2 Behaviour

159. The branch office as received an `open account` request.
    (a) Based on the customer information the branch office examines whether to decline or grant the request.
    (b) If it is to decline the reques it replies so to the customer
    (c) and resumes being a `branch_office`.
    (d) If it is to accept the `open account` request then it alerts its bank as to a fresh, hitherto unused account number, and
    (e) awaits one such.
    (f) It then communicates this to the customer
    (g) and resumes being a `branch_office`.

```
value
159.  open_account_bo(boi)(bom)(static)(progr:((ci,τ,bi),cmd:mk_OpenAcct(info,inc,as,dis,kis,lia)),curr) ≡
159a.      let yes_no = examine_open_account_request(cmd,progr) in
159b.      if yes_no = "no"
```

[34] The bank [head quarter or branch office] judges that the customer is not "fit".

```
159b.      then comm[{boi,ci}] ! mk_DeclineAcct("no") ;
159c.           branch_office(boi)(bom)(static)(progr)
159d.      else  comm[{xtr_BI(boi),boi}] ! mk_NewAcctNu(ci,boi) ;
159e.           let mk_AcctNu(anu) = comm[{xtr_BI(boi),boi}] ?  in
159f.           comm[{boi,ci}] ! mk_NewAcct(bi,anu) ;
159g.           branch_office(boi)(bom)(static)(progr) end
159.    end end
```

6.5.2.2 **Open Debit/Credit Card**

6.5.2.2.1 Command

160. The

    (a)

    (b)

    (c)

    (d)

```
160.
160a.
160b.
160c.
160d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.8 on page 48.

6.5.2.2.2 Behaviour

161. The

    (a)

    (b)

    (c)

    (d)

value

```
161. open_CD_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_ObtainCreDebCard()) ≡
161a.
161b.
161c.
161d.
```

6.5.2.3 **Close Debit/Credit Card**

6.5.2.3.1 Command

162. The

   (a)

   (b)

   (c)

   (d)

type
162.
162a.
162b.
162c.
162d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.10 on page 50.

6.5.2.3.2 Behaviour

163. The

   (a)

   (b)

   (c)

   (d)

value
163. close_CD_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_ClosCreDebCard()) $\equiv$
163a.
163b.
163c.
163d.

6.5.2.4 **Open Payment**

6.5.2.4.1 Command

164. The

    (a)

    (b)

    (c)

    (d)

```
164.
164a.
164b.
164c.
164d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.11 on page 51.

6.5.2.4.2 Behaviour

165. The

    (a)

    (b)

    (c)

    (d)

value

```
165.  open_paym_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_OpenPaymentService()) ≡
165a.
165b.
165c.
165d.
```

6.5.2.5 Close Payment

6.5.2.5.1 Command

166. The
    (a)
    (b)
    (c)
    (d)

type
166.
166a.
166b.
166c.
166d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.12 on page 52.

6.5.2.5.2 Behaviour

167. The
    (a)
    (b)
    (c)
    (d)

value
167. close_payt_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_ClosePaymentService()) ≡
167a.
167b.
167c.
167d.

6.5.2.6 **Open Deposit**

6.5.2.6.1 Command

168. The
    (a)
    (b)
    (c)
    (d)

type
168.
168a.
168b.
168c.
168d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.13 on page 53.

6.5.2.6.2 Behaviour

169. The
    (a)
    (b)
    (c)
    (d)

type
169. open_deposit_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_OpenDepositService()) $\equiv$
169a.
169b.
169c.
169d.

6.5.2.7 **Close Deposit**

6.5.2.7.1 Command

170. The
    (a)
    (b)
    (c)
    (d)

type
170.
170a.
170b.
170c.
170d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.14 on page 54.

6.5.2.7.2 Behaviour

171. The
    (a)
    (b)
    (c)
    (d)

value
171. close_deposit_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_CloseDepositService()) $\equiv$
171a.
171b.
171c.
171d.

6.5.2.8 **Close Account**

6.5.2.8.1 Command

172. The
    (a)
    (b)
    (c)
    (d)

```
type
172.
172a.
172b.
172c.
172d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.23 on page 63.

6.5.2.8.2 Behaviour

173. The
    (a)
    (b)
    (c)
    (d)

```
value
173.  close_account_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_CloseAcct()) ≡
173a.
173b.
173c.
173d.
```

6.5.2.9 **Open Loan**

6.5.2.9.1 Command

174. The

   (a)

   (b)

   (c)

   (d)

type
174.
174a.
174b.
174c.
174d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.15 on page 55.

6.5.2.9.2 Behaviour

175. The

   (a)

   (b)

   (c)

   (d)

value
175. apply_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_ApplyLoan()) ≡
175a.
175b.
175c.
175d.

6.5.2.10 Increase Loan

6.5.2.10.1 Command

176. The
    (a)
    (b)
    (c)
    (d)

```
type
176.
176a.
176b.
176c.
176d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.17 on page 57.

6.5.2.10.2 Behaviour

177. The
    (a)
    (b)
    (c)
    (d)

```
value
177. increase_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_CloseLoan()) ≡
177a.
177b.
177c.
177d.
```

6.5.2.11 **Close Loan**

6.5.2.11.1 Command

178. The

    (a)

    (b)

    (c)

    (d)

```
type
178.
178a.
178b.
178c.
178d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.18 on page 58.

6.5.2.11.2 Behaviour

179. The

    (a)

    (b)

    (c)

    (d)

```
value
179. close_loan_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_CloseLoan()) ≡
179a.
179b.
179c.
179d.
```

6.5.2.12 Buy Stocks or Bonds

6.5.2.12.1 Command

180. The

    (a)

    (b)

    (c)

    (d)

```
type
180.
180a.
180b.
180c.
180d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.18 on page 58.

6.5.2.12.2 Behaviour

181. The

    (a)

    (b)

    (c)

    (d)

```
value
181. buy_stock_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_BuyStock()) ≡
181a.
181b.
181c.
181d.
```

6.5.2.13 **Sell Stocks or Bonds**

6.5.2.13.1 Command

182. The
    (a)
    (b)
    (c)
    (d)

```
type
182.
182a.
182b.
182c.
182d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.18 on page 58.

6.5.2.13.2 Behaviour

183. The
    (a)
    (b)
    (c)
    (d)

```
value
183. sell_stock_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_SellStock()) ≡
183a.
183b.
183c.
183d.
```

6.5.2.14 **Incl./Excl. Deb,/Cre.**

6.5.2.14.1 Command

184. The

    (a)

    (b)

    (c)

    (d)

```
type
184.
184a.
184b.
184c.
184d.
```

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.18 on page 58.

6.5.2.14.2 Behaviour

185. The

    (a)

    (b)

    (c)

    (d)

```
value
185. incl_excl_deb_cre_bo(boi)(bom)(static)(progr)((ci,τ,bi),mk_InclExclDebCre()) ≡
185a.
185b.
185c.
185d.
```

6.5.2.15 **Change Account**

6.5.2.15.1 Command

186. The

    (a)

    (b)

    (c)

    (d)

type
186.
186a.
186b.
186c.
186d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.18 on page 58.

6.5.2.15.2 Behaviour

187. The

    (a)

    (b)

    (c)

    (d)

value
187. change_account_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_ChangeAcct()) ≡
187a.
187b.
187c.
187d.

6.5.2.16 Close Account

6.5.2.16.1 Command

188. The
    (a)
    (b)
    (c)
    (d)

type
188.
188a.
188b.
188c.
188d.

The re-active command [above] and the behaviour [next] is in response command/behaviour Sect. 4.4.2.23 on page 63.

6.5.2.16.2 Behaviour

189. The
    (a)
    (b)
    (c)
    (d)

value
189. close_account_bo(boi)(bom)(static)(progr)((ci,τ,bi),**mk**_CloseAcct()) $\equiv$
189a.
189b.
189c.
189d.

6.5.3 Branch Office Pro-active Behaviours

190. The

    (a)

    (b)

    (c)

    (d)

value

```
190.   yyy(boi)(mereo:bom)(stat:regnu)(progr:(custctlg,access,bohist)) ≡
190a.
190b.
190c.
190d.
```

6.5.3.1 **Information From Bank**

6.5.3.1.1 Command

191. The

    (a)

    (b)

    (c)

    (d)

type
191.
191a.
191b.
191c.
191d.

6.5.3.1.2 Behaviour

192. The

    (a)

    (b)

    (c)

    (d)

value
192. info_from_bank(...) ≡
192a.
192b.
192c.
192d.

6.5.3.2 XXX

193. The

    (a)

    (b)

    (c)

    (d)

    (e)

value

```
193.  yyy(boi)(bom)(regnu)(progr:(custctlg,access,bohist)) ≡
193a.
193b.
193c.
193d.
193e.
```

6.5.3.3 **YYY**

194. The

    (a)

    (b)

    (c)

    (d)

    (e)

value

```
194.   yyy(boi)(bom)(regnu)(progr:(custctlg,access,bohist)) ≡
194a.
194b.
194b.
194b.
```

6.5.3.4 ZZZ

195. The

   (a)

   (b)

   (c)

   (d)

   (e)

value

```
195.   yyy(boi)(bom)(regnu)(progr:(custctlg,access,bohist)) ≡
195a.
195b.
195c.
195d.
195e.
```

6.5.3.5 WWW

196. The

    (a)

    (b)

    (c)

    (d)

    (e)

value

```
196.   yyy(boi)(bom)(regnu)(progr:(custctlg,access,bohist)) ≡
196a.
196b.
196c.
196d.
196e.
```

Chapter 7

Bank Head Quarter

Contents

7.1 **External Qualities 129**

7.1.1 The Endurants 129

7.2 **Internal Qualities 129**

7.2.1 Unique Identification 129

7.2.2 Mereology 130

7.2.3 Attributes 130

7.2.3.1 Sorts & Types 130

7.2.3.2 Wellformedness 130

7.3 **Bank Head Quarte Intentional Pull 130**

7.4 **Bank Head Quarte Commands 130**

7.5 **Bank Head Quarter Behaviours 130**

7.5.1 The Bank Head Quarter Behaviour 130

7.5.2 The Bank Head Quarter Re-active Behaviours 130

7.5.3 The Bank Head Quarter Pro-active Behaviours 130

Note: In this chapter we shall relate the bank head quarter to the bank, its branch offices, to national banks, regional banks, etc., as well as to other banks, f.ex. with respect to currency trading.

7.1 External Qualities

7.1.1 The Endurants

Was treated in Sect. 3.3.2 on page 16.

7.2 Internal Qualities

7.2.1 Unique Identification

Was treated in Sect. 3.3.3.1 on page 19.

7.2.2 Mereology

Was treated in Sect. 3.3.3.2 on page 24.

7.2.3 Attributes

7.2.3.1 Sorts & Types

7.2.3.2 Wellformedness

7.3 Bank Head Quarte Intentional Pull

7.4 Bank Head Quarte Commands

7.5 Bank Head Quarter Behaviours

7.5.1 The Bank Head Quarter Behaviour

7.5.2 The Bank Head Quarter Re-active Behaviours

7.5.3 The Bank Head Quarter Pro-active Behaviours

Chapter 8

Tellers

Contents

8.1 **Tellers and Automatic Teller Machines[ATM]** **131**

8.2 **ATMs** **131**

8.3 **Discussion** **131**

8.1 Tellers and Automatic Teller Machines[ATM]

8.2 ATMs

8.3 Discussion

Chapter 9

Credit/Debit Company

Contents

9.1 **External Qualities 134**

9.1.1 The Endurants 134

9.1.1.1 Endurant Sorts 134

9.1.1.2 An Endurant State 134

9.2 **Internal Qualities 134**

9.2.1 Unique Identification 134

9.2.1.1 Unique Identifier Sorts 134

9.2.1.2 A Unique Identifier State 134

9.2.2 Mereology 134

9.2.2.1 Sorts 134

9.2.2.2 Wellformedness 134

9.2.3 Attributes 134

9.2.3.1 Sorts & Types 134

9.2.3.2 Wellformedness 134

9.3 **Credit/Debit CompanyIntentional Pull 134**

9.4 **Credit/Debit CompanyCommands 134**

9.1 External Qualities

9.1.1 The Endurants

9.1.1.1 Endurant Sorts

9.1.1.2 An Endurant State

9.2 Internal Qualities

9.2.1 Unique Identification

9.2.1.1 Unique Identifier Sorts

9.2.1.2 A Unique Identifier State

9.2.2 Mereology

9.2.2.1 Sorts

9.2.2.2 Wellformedness

9.2.3 Attributes

9.2.3.1 Sorts & Types

9.2.3.2 Wellformedness

9.3 Credit/Debit CompanyIntentional Pull

9.4 Credit/Debit CompanyCommands

Chapter 10

Mortgage, Savings and Loan Companies

Contents

10.1 **External Qualities** **136**

10.1.1 The Endurants 136

10.1.1.1 Endurant Sorts 136

10.1.1.2 An Endurant State 136

10.2 **Internal Qualities** **136**

10.2.1 Unique Identification 136

10.2.1.1 Unique Identifier Sorts 136

10.2.1.2 A Unique Identifier State 136

10.2.2 Mereology 136

10.2.2.1 Sorts 136

10.2.2.2 Wellformedness 136

10.2.3 Attributes 136

10.2.3.1 Sorts & Types 136

10.2.3.2 Wellformedness 136

10.3 **Credit/Debit CompanyIntentional Pull** **136**

10.4 **Credit/Debit CompanyCommands** **136**

- https://corporatefinanceinstitute.com/resources/wealth-management/mortgage-bank/

  A mortgage bank is a bank specializing in mortgage loans. It can be involved in originating or servicing mortgage loans, or both. The banks loan their own capital to borrowers and either collect payments in installments along with a certain rate of interest or sell their loans in the secondary market.

  Mortgage Bankers vs. Mortgage Brokers

- In terms of loan origination, mortgage bankers risk their own capital to fund loans. Also, they are not required to disclose the price at which they sell mortgages.
- On the other hand, mortgage brokers originate loans in the name of financial institutions and organizations. Regarding full disclosure, they need to disclose the additional fee(s) charged to the consumer under federal and state laws.

-
- https://danskebank.dk/en/personal/products/loans/personal-loans

10.1 External Qualities

10.1.1 The Endurants

10.1.1.1 Endurant Sorts

10.1.1.2 An Endurant State

10.2 Internal Qualities

10.2.1 Unique Identification

10.2.1.1 Unique Identifier Sorts

10.2.1.2 A Unique Identifier State

10.2.2 Mereology

10.2.2.1 Sorts

10.2.2.2 Wellformedness

10.2.3 Attributes

10.2.3.1 Sorts & Types

10.2.3.2 Wellformedness

10.3 Credit/Debit CompanyIntentional Pull

10.4 Credit/Debit CompanyCommands

Part IV

Stocks: Brokers & Exchange

Chapter 11

Stock Brokers

TO BE WRITTEN

Chapter 12

Stock Exchanges

- We refer to
  - Appendix Chapter C, pages 241–256:
  - An RSL model of the *The Tokyo Stock Exchange*.
- Also on the internet:
  - **www.imm.dtu.dk/˜db/todai/tse-1.pdf**,
  - **www.imm.dtu.dk/˜db/todai/tse-2.pdf**

TO BE WRITTEN

Part V

National, Regional & Global Banks

Chapter 13

National Banks: Endurants and Commands

The Danish National Bank: We contribute to stable prices through the fixed exchange rate policy. We ensure that payments can be effected in a secure and efficient manner. And we are committed to ensuring stability in the financial sector.

Also: VP Securities and

13.1 Endurants

13.1.1 External Qualities

13.1.1.1 The Endurant Sorts

13.1.1.2 An Endurant State

13.1.2 Internal Qualities

13.1.2.1 Unique Identification

13.1.2.1.1 Unique Identifier Sorts

13.1.2.1.2 A Unique Identifier State

13.1.2.2 Mereology

13.1.2.3 Attributes

13.1.2.3.1 Attribute Sorts

13.1.2.3.2 Attribute Wellformedness

13.2 National Bank Intentional Pull

13.3 National Bank Commands

Chapter 14

Central Banks: Endurants and Commands

14.1 Endurants

14.1.1 External Qualities

14.1.1.1 The Endurant Sorts

14.1.1.2 An Endurant State

14.1.2 Internal Qualities

14.1.2.1 Unique Identification

14.1.2.1.1 Unique Identifier Sorts

14.1.2.1.2 A Unique Identifier State

14.1.2.2 Mereology

14.1.2.3 Attributes

14.1.2.3.1 Attribute Sorts

14.1.2.3.2 Attribute Wellformedness

14.2 Regional Bank Intentional Pull

14.3 Regional Bank Commands

Chapter 15

IMF: Endurants and Commands

15.1 Endurants

15.1.1 External Qualities

15.1.1.1 The Endurant Sorts

15.1.1.2 An Endurant State

15.1.2 Internal Qualities

15.1.2.1 Unique Identification

15.1.2.1.1 Unique Identifier Sorts

15.1.2.1.2 A Unique Identifier State

15.1.2.2 Mereology

15.1.2.3 Attributes

15.1.2.3.1 Attribute Sorts

15.1.2.3.2 Attribute Wellformedness

15.2 IMF Intentional Pull

15.3 IMF Office Commands

Chapter 16

The World Bank: Endurants and Commands

16.1 Endurants

16.1.1 External Qualities

16.1.1.1 The Endurant Sorts

16.1.1.2 An Endurant State

16.1.2 Internal Qualities

16.1.2.1 Unique Identification

16.1.2.1.1 Unique Identifier Sorts

16.1.2.1.2 A Unique Identifier State

16.1.2.2 Mereology

16.1.2.3 Attributes

16.1.2.3.1 Attribute Sorts

16.1.2.3.2 Attribute Wellformedness

16.2 The World Bank Intentional Pull

16.3 The World Bank Commands

Part VI

Closing

Chapter 17

Discussion

- **The Banking Decomposition:** I have chosen a "flat" structuring [i.e., decomposition] of the world banking system.

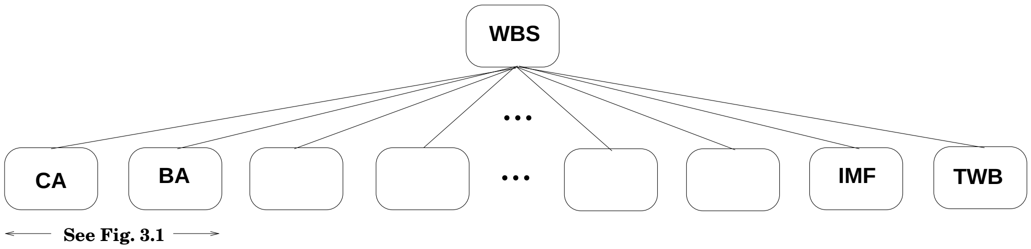


Figure 17.1: One Rendition of a World Banking System

This is in contrast to a decomposition that "favours" structuring by nationality, etc.:

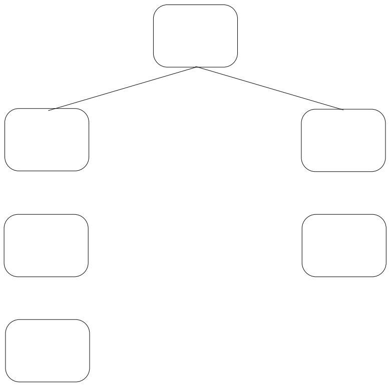


Figure 17.2: Another Rendition of a World Banking System

Figure 17.2 is under construction!

MORE TO COME

- **:**
- **:**
- **:**
- **:**

Chapter 18

Conclusion

Contents

18.1 **What Have We Achieved ?** **157**
18.2 **What Have We Not Achieved ?** **157**
18.3 **How Was This Domain Modelling Approached ?** **157**
18.4 **A Prelude to a Professional Banking Domain R & D** **157**
18.5 **What Next ?** **157**
18.6 **Acknowledgements** **157**
18.7 **References** **158**

18.1 What Have We Achieved ?

18.2 What Have We Not Achieved ?

18.3 How Was This Domain Modelling Approached ?

18.4 A Prelude to a Professional Banking Domain R & D

18.5 What Next ?

18.6 Acknowledgements

18.7 References

[1] J. L. Austin. How to Do Things with Words. Harvard University Press, Cambridge, Mass., 2 edition, 1975. (William James Lectures).

[2] H. Bekič, D. Bjørner, W. Henhapl, C.B. Jones, and P. Lucas. A Formal Definition of a PL/I Subset. Technical Report 25.139, Vienna, Austria, December 1974.

[3] Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version III. IBM Laboratory, Vienna, 1969.

[4] D. Bjørner and O. Oest. Towards a Formal Description of Ada, volume 98 of LNCS. Springer–Verlag, 1980.

[5] Dines Bjørner. Domain Case Studies:

- 2025: *Documents – a Domain Description*, Winter/Spring 2025, `www.imm.dtu.dk/ dibj/2025/documents/main.pdf`
- 2023: *Nuclear Power Plants, A Domain Sketch*, 21 July, 2023 `www.imm.dtu.dk/ dibj/2023/nupopl/nupopl.pdf`
- 2021: *Shipping*, April 2021. `www.imm.dtu.dk/ dibj/2021/ral/ral.pdf`
- 2021: *Rivers and Canals – Endurants*, March 2021. `www.imm.dtu.dk/ dibj/2021/Graphs/Rivers-and-Canals.pdf`
- 2021: *A Retailer Market*, January 2021. `www.imm.dtu.dk/ dibj/2021/Retailer/BjornerHeraklit27January2021.pdf`
- 2019: *Container Terminals*, ECNU, Shanghai, China `www.imm.dtu.dk/ dibj/2018/yangshan/maersk-pa.pdf`
- 2018: *Documents*, TongJi Univ., Shanghai, China `www.imm.dtu.dk/ dibj/2017/docs/docs.pdf`
- 2017: *Urban Planning*, TongJi Univ., Shanghai, China `www.imm.dtu.dk/ dibj/2017/urban-planning.pdf`
- 2017: *Swarms of Drones*, IS/CAS[35], Peking, China `www.imm.dtu.dk/ dibj/2017/swarms/swarm-paper.pdf`
- 2013: *Road Transport*, Techn. Univ. of Denmark `www.imm.dtu.dk/ dibj/road-p.pdf`
- 2012: *Credit Cards*, Uppsala, Sweden `www.imm.dtu.dk/ dibj/2016/credit/accs.pdf`
- 2012: *Weather Information*, Bergen, Norway `www.imm.dtu.dk/ dibj/2016/wis/wis-p.pdf`
- 2010: *Web-based Transaction Processing*, Techn. Univ. of Vienna, Austria, 186 pages `www.imm.dtu.dk/ dibj/wfdftp.pdf`
- 2010: *The Tokyo Stock Exchange*, Tokyo Univ., Japan `www.imm.dtu.dk/ db/todai/tse-2.pdf`

[35] Inst. of Softw., Chinese Acad. of Sci.

- 2009: *Pipelines*, Techn. Univ. of Graz, Austria `www.imm.dtu.dk/ dibj/pipe-p.pdf`
- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark `www.imm.dtu.dk/ dibj/container-paper.pdf`
- 2002: *The Market*, Techn. Univ. of Denmark `www.imm.dtu.dk/ dibj/themarket.pdf`
- 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium `www.imm.dtu.dk/ dibj/train-book.pdf`

Experimental research carried out to “discover”, try-out and refine method principles, techniques and tools, 1995–2025.

[6] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, 9th IFAC Symposium on Control in Transportation Systems, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.

[7] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In CTS2003: 10th IFAC Symposium on Control in Transportation Systems, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. `www2.imm.dtu.dk/ dibj/ifac-dynamics.pdf`.

[8] Dines Bjørner. New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems. In FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. `www2.imm.dtu.dk/ dibj/dines-amore.pdf`.

[9] Dines Bjørner. Software Engineering, Vol. 1: Abstraction and Modelling; Vol. 2: Specification of Systems and Languages; Vol. 3: Domains, Requirements and Software Design. Texts in Theoretical Computer Science, the EATCS Series. Springer, Heidelberg, Germany, 2006.

[10] Dines Bjørner. From Domains to Requirements `www.imm.dtu.dk/ dibj/2008/ugo/ugo65.pdf`. In Montanari Festschrift, volume 5065 of Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer), pages 1–30, Heidelberg, May 2008. Springer.

[11] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, Formal Methods: State of the Art and New Directions, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[12] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, Formal Methods: State of the Art and New Directions, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[13] Dines Bjørner. A Rôle for Mereology in Domain Science and Engineering. In Mereology and the Sciences, Synthese Library (eds. Claudio Calosi and Pierluigi Graziani), pages 323–357, Amsterdam, The Netherlands, October 2014. Springer. `https://www.imm.dtu.dk/ dibj/2011/urbino/urbino-colour.pdf`.

[14] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model `www.imm.dtu.dk/ dibj/2014/kanazawa/kanazawa-p.pdf`. In Shusaku Iida and José Meseguer and Kazuhiro Ogata, editor, Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi. Springer, Heidelberg, Garmany, May 2014.

[15] Dines Bjørner. Manifest Domains: Analysis & Description `www.imm.dtu.dk/ dibj/2015/faoc/faoc-bjorner.pdf`. Formal Aspects of Computing, 29(2):175–225, March 2017. Online: 26 July 2016.

[16] Dines Bjørner. Manifest Domains: Analysis & Description `www.imm.dtu.dk/ dibj/2015/faoc/faoc-bjorner.pdf`. Formal Aspects of Computing, 29(2):175–225, March 2017. Online: 26 July 2016.

[17] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. `www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf`. ACM Trans. on Software Engineering and Methodology, 28(2):66 pages, March 2019.

[18] Dines Bjørner. Domain Analysis & Description. `www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf`. ACM Trans. on Software Engineering and Methodology, 28(2):66 pages, March 2019.

[19] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [24].

[20] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [22].

[21] Dines Bjørner. Domain Modelling – A Primer. A short and significantly revised version of [20]. xii+202 pages[36], May 2023.

[22] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [20]. xii+346 pages[37], January 2023.

[23] Dines Bjørner. Double-entry Bookkeeping. Research, Institute of Mathematics and Computer Science. Technical University of Denmark, DK-2800 Kgs.Lyngby, Denmark, August 2023. `http://www.imm.dtu.dk/~dibj/2023/doubleentry/-dblentrybook.pdf`. One in a series of planned studies: [26, 31, 30, 29].

[36] This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing and into Russian by Dr. Mikhail Chupilko and his colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow

[37] Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [21]

[24] Dines Bjørner. Domain Modelling – A Primer. A significantly revised version of [19]. xii+202 pages[38], Summer 2024.

[25] Dines Bjørner. Domain Models – A Compendium. Internet: `http://www.imm.-dtu.dk/~dibj/2024/models/domain-models.pdf`, March 2024. This is a very early draft. 19 domain models are presented.

[26] Dines Bjørner. Banking – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [31, 30, 29, 23].

[27] Dines Bjørner. Domain Analysis & Description. To be submitted, page 33, March 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.

[28] Dines Bjørner. Domain Modelling. Submitted to ACM FAC, page 18, February 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.

[29] Dines Bjørner. Health Care – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [26, 31, 30, 23].

[30] Dines Bjørner. Insurance – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [26, 31, 29, 23].

[31] Dines Bjørner. Transport – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [26, 30, 29, 23].

[32] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. `www2.imm.dtu.dk/ dibj/pasadena-25.pdf`.

[33] Dines Bjørner and Cliff B. Jones, editors. The Vienna Development Method: The Meta-Language, volume 61 of LNCS. Springer, Heidelberg, Germany, 1978.

[34] Dines Bjørner and Cliff B. Jones, editors. Formal Specification and Software Development. Prentice-Hall, London, England, 1982.

[35] Dines Bjørner and Ole N. Oest, editors. Towards a Formal Description of Ada, volume 98 of LNCS. Springer, Heidelberg, Germany, 1980.

[38] This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing and into Russian by Dr. Mikhail Chupilko and colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow

[36] Geert Bagge Clemmensen and Ole N. Oest. Formal specification and development of an Ada compiler – a VDM case study. In Proc. 7th International Conf. on Software Engineering, 26.-29. March 1984, Orlando, Florida, pages 430–440, New York, USA, 1984. IEEE.

[37] Patrick Cousot. Principles of Abstract Interpretation. The MIT Press, 2021.

[38] Peter Fettke and Wolfgang Reisig. Understanding the Digital World – Modeling with HERAKLIT. Springer, 2024. To be published.

[39] John Fitzgerald and Peter Gorm Larsen. Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[40] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. CAFE: An Industrial–Strength Algebraic Formal Method, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

[41] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. The RAISE Specification Language. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[42] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. The RAISE Development Method. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[43] Chris W. George, Hung Dang Van, Tomasz Janowski, and Richard Moore. Case Studies using The RAISE Method. FACTS (Formal Aspects of Computing: Theory and Software) and FME (Formal Methods Europe). Springer–Verlag, London, 2002. This book reports on a number of case studies using RAISE (Rigorous Approach to Software Engineering). The case studies were done in the period 1994–2001 at UNU/IIST, the UN University's International Institute for Software Technology, Macau (till 20 Dec., 1997, Chinese Teritory under Portuguese administration, now a Special Administrative Region (SAR) of (the so–called People's Republic of) China).

[44] Michael Hammer and James A. Champy. Reengineering the Corporation: A Manifesto for Business Revolution. HarperCollins*Publishers*, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, May 1993. 5 June 2001, Paperback.

[45] Michael Hammer and Stephen A. Stanton. The Reengineering Revolutiuon: The Handbook. HarperCollins*Publishers*, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, 1996. Paperback.

[46] Michael Reichhardt Hansen and Hans Rischel. Functional Programming Using F#. Cambridge University Press, 2013.

[47] Charles Anthony Richard Hoare. Communicating Sequential Processes. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, London, England, 1985. Published electronically: `usingcsp.com/cspbook.pdf` (2004).

[48] Gerard J. Holzmann. The SPIN Model Checker, Primer and Reference Manual. Addison-Wesley, Reading, Massachusetts, 2003.

[49] Daniel Jackson. Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[50] Michael A. Jackson. Software Requirements & Specifications: a lexicon of practice, principles and prejudices. ACM Press. Addison-Wesley, Reading, England, 1995.

[51] Michael A. Jackson. Problem Frames — Analyzing and Structuring Software Development Problems. ACM Press, Pearson Education. Addison-Wesley, England, 2001.

[52] Michael A. Jackson. Program Verification and System Dependability. In Paul Boca, Jonathan Bowen, and Jawed Siddiqi, editors, Formal Methods: State of the Art and New Directions, pages 43–78, London, UK, December 2009. Springer.

[53] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. The Shorter Oxford English Dictionary on Historical Principles. Clarendon Press, Oxford, England, 1973, 1987. Two vols.

[54] R. Milne and C. Strachey. A Theory of Programming Language Semantics. Chapman and Hall, London, Halsted Press/John Wiley, New York, 1976.

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

[56] Charles W. Morris. Foundations of the theory of signs, volume I of International encyclopedia of unified science. The University of Chicago Press, 1938.

[57] Karl R. Popper. Conjectures and Refutations. The Growth of Scientific Knowledge. Routledge and Kegan Paul Ltd. (Basic Books, Inc.), 39 Store Street, WC1E 7DD, London, England (New York, NY, USA), 1963,. . . ,1981.

[58] F. Pulvermüller. Brain mechanisms linking language and action. Nature Reviews: Neuroscience, 6:576582, 2005. https://doi.org/10.1038/nrn1706.

[59] John R. Searle. Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, 1969.

[60] Kai Sørlander. Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]. Munksgaard · Rosinante, Copenhagen, Denmark, 1994. 168 pages.

[61] Kai Sørlander. Under Evighedens Synsvinkel [Under the viewpoint of eternity]. Munksgaard · Rosinante, Copenhagen, Denmark, 1997. 200 pages.

[62] Kai Sørlander. Den Endegyldige Sandhed [The Final Truth]. Rosinante, Copenhagen, Denmark, 2002. 187 pages.

[63] Kai Sørlander. Indføring i Filosofien [Introduction to The Philosophy]. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages.

[64] Kai Sørlander. Den rene fornufts struktur [The Structure of Pure Reason]. Ellekær, Slagelse, Denmark, 2022. See [65].

[65] Kai Sørlander. The Structure of Pure Reason. Springer, February 2025. This is an English translation of [64] – done by Dines Bjørner in collaboration with the author.

[66] Tetsuo Tamai. Social Impact of Information System Failures. Computer, IEEE Computer Society Journal, 42(6):58–65, June 2009.

[67] Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore, editors. Specification Case Studies in RAISE. Springer, 2002.

[68] Achille C. Varzi. On the Boundary between Mereology and Topology, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.

[69] James Charles Paul Woodcock and James Davies. Using Z: Specification, Proof and Refinement. Prentice Hall International Series in Computer Science, London, England, 1996.

Part VII

Appendix

Appendix A

A Raise Specification Language Primer

Contents

A.1 **Types and Values** **169**

A.1.1 Sort and Type Expressions 169

A.1.1.1 Atomic Types: Identifier Expressions and Type Values . 169

A.1.1.2 Composite Types: Expressions and Type Values 170

A.1.2 Type Definitions 171

A.1.2.1 Sorts — Abstract Types 171

A.1.2.2 Concrete Types 172

A.1.2.3 Subtypes 173

A.2 **The Propositional and Predicate Calculi** **174**

A.2.1 Propositions 174

A.2.1.1 Propositional Expressions 174

A.2.1.2 Propositional Calculus 174

A.2.2 Predicates 175

A.2.2.1 Predicate Expressions 175

A.2.2.2 Predicate Calculus 176

A.3 **Arithmetics** **176**

A.4 **Comprehensive Expressions** **176**

A.4.1 Set Enumeration and Comprehension 176

A.4.1.1 Set Enumeration 176

A.4.1.2 Set Comprehension 177

A.4.1.3 Cartesian Enumeration 177

A.4.2 List Enumeration and Comprehension 177

A.4.2.1 List Enumeration 177

A.4.2.2 List Comprehension 178

A.4.3 Map Enumeration and Comprehension 178

A.4.3.1 Map Enumeration 178

A.4.3.2 Map Comprehension 178

A.5 **Operations** **179**

A.5.1 Set Operations 179

A.5.1.1 Set Operator Signatures 179

A.5.1.2 Set Operation Examples 179

A.5.1.3 Informal Set Operator Explication 180

A.5.1.4 Set Operator Explications 180

A.5.2 Cartesian Operations 181

A.5.3 List Operations 181

A.5.3.1 List Operator Signatures 181

A.5.3.2 List Operation Examples 182

A.5.3.3 Informal List Operator Explication 182

A.5.3.4 List Operator Explications 183

A.5.4 Map Operations 184

A.5.4.1 Map Operator Signatures 184

A.5.4.2 Map Operation Examples 184

A.5.4.3 Informal Map Operation Explication 184

A.5.4.4 Map Operator Explication 185

A.6 **λ-Calculus + Functions** **186**

A.6.1 The λ-Calculus Syntax 186

A.6.2 Free and Bound Variables 186

A.6.3 Substitution 186

A.6.4 α-Renaming and β-Reduction 187

A.6.5 Function Signatures 187

A.6.6 Function Definitions 187

A.7 **Other Applicative Expressions** **188**

A.7.1 Simple **let** Expressions 188

A.7.2 Recursive **let** Expressions 189

A.7.3 Predicative **let** Expressions 189

A.7.4 Pattern and “Wild Card” **let** Expressions 189

A.7.4.1 Conditionals 190

A.7.5 Operator/Operand Expressions 190

A.8 **Imperative Constructs** **191**

A.9 **Process Constructs** **191**

A.9.1 Process Channels 191

A.9.2 Process Composition 191

A.9.3 Input/Output Events 192

A.9.4 Process Definitions 192

A.10 RSL **Module Specifications** **192**

A.11 **Simple** RSL **Specifications** **192**

A.12 RSL$^+$**: Extended** RSL **193**

A.12.1 Type Names and Type Name Values 193

A.12.1.1 Type Names 193
A.12.1.2 Type Name Operations 193
A.12.2 RSL-Text 193
A.12.2.1 The RSL-Text Type and Values 193
A.12.2.2 RSL-Text Operations 194
A.13 Distributive Clauses 194
A.13.1 Over Simple Values 194
A.13.2 Over Processes 194
A.14 Space and Time 194
A.14.1 Space 195
A.14.2 Time 195

We [39] present an `RSL Primer`. Indented text, in slanted font, such as this, presents informal material and examples. Non-indented text, in roman font, presents narrative and formal explanation of RSL constructs.

This `RSL Primer` omits treatment of a number of language constructs, notably the RSL module concepts of *schemes, classes* and *objects*. Although we do cover the imperative language construct of [declaration of] variables and, hence, assignment, we shall omit treatment of structured imperative constructs like **for** ..., **do** *s* **while** *b*, **while** *b* **do** *s* loops.

Section A.12 on page 193 introduces additional language constructs, thereby motivating the $^+$ in the RSL$^+$ name[40]

A.1 Types and Values

$\mathscr{I}$: Types are, in general, set-like structures[41] of things, i.e., values, having common characteristics.

A bunch of zero, one or more apples (type *apples*) may thus form a [sub]set of type *Belle de Boskoop* apples. A bunch of zero, one or more pears (type *pears*) may thus form a [sub]set of type *Concorde* pears. A union of zero, one or more of these apples and pears then form a [sub]set of entities of type *fruits*.

A.1.1 Sort and Type Expressions

Sort and type expressions are expressions whose values are types, that is, possibly infinite set-like structures of values (of "that" type).

A.1.1.1 Atomic Types: Identifier Expressions and Type Values

Atomic types have (atomic) values. That is, values which we consider to have no proper constituent (sub-)values, i.e., cannot, to us, be meaningfully "taken apart".

RSL has a number of [so-called] built-in atomic types. They are expressed in terms of literal identifiers. These are the **Bool**ea**n**s, **int**egers, **Nat**ural numbers, **Real**s, **Char**acters,

[39] The letter $\mathscr{I}$ shall designate begin of informal text.

[40] The ■ symbol shall designate end-of-informal text.

[41] We shall not, in this primer, go into details as to the mathematics of types.

and **Text**s. **Text**s are free-form texts and are more general than just texts of `RSL-like` formulas. `RSL`-Text's will be introduced in Sect. A.12 on page 193.

We shall not need the base types **Char**acters, nor the general type **Text**s for domain modelling in this primer. They will be listed below, but not mentioned further.

The base types are:

Basic Types

type
[1] **Bool**
[2] **Int**
[3] **Nat**
[4] **Real**
[5] **Char**
[6] **Text**

1. The Boolean type of truth values **false** and **true**.
2. The integer type on integers ..., –2, –1, 0, 1, 2, ...
3. The natural number type of positive integer values 0, 1, 2, ...
4. The real number type of real values, i.e., values whose numerals can be written as an integer, followed by a period ("."), followed by a natural number (the fraction).
5. The character type of character values ″a″, ″bbb″, ...
6. The text type of character string values ″aa″, ″aaa″, ..., ″abc″, ...

A.1.1.2 Composite Types: Expressions and Type Values

Composite types have composite values. That is, values which we consider to have proper constituent (sub-)values, i.e., can, to us, be meaningfully "taken apart".

From these one can form type expressions: finite sets, infinite sets, Cartesian products, lists, maps, etc.

Let A, B and C be any type names or type expressions, then these are the composite types, hence, type expressions:

Composite Type Expressions

[7] A-**set**
[8] A-**infset**
[9] A × B × ... × C
[10] A^*
[11] A^ω
[12] A $\underset{m}{\rightarrow}$ B
[13] A → B

[14] A $\overset{\sim}{\to}$ B
[15] A | B | ... | C
[16] mk_id(sel_a:A,...,sel_b:B)
[17] sel_a:A ... sel_b:B

The following are generic type expressions:

7. The set type of finite cardinality set values.

8. The set type of infinite and finite cardinality set values.

9. The Cartesian type of Cartesian values.

10. The list type of finite length list values.

11. The list type of infinite and finite length list values.

12. The map type of finite definition set map values.

13. The function type of total function values.

14. The function type of partial function values.

15. The postulated disjoint union of types A, B, . . . , and C.

16. The record type of mk_id-named record values mk_id(av,...,bv), where av, . . . , bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.

17. The record type of unnamed record values (av,...,bv), where av, . . . , bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.

Section A.12 on page 193 introduces the extended RSL concepts of type name values and the type, $\mathbb{T}$, of type names.

A.1.2 Type Definitions

A.1.2.1 Sorts — Abstract Types

Types can be (abstract) sorts in which case their structure is not specified:

Sorts

type
A, B, ..., C

A.1.2.2 Concrete Types

Types can be concrete in which case the structure of the type is specified by type expressions:

Type Definition

```
type
   A = Type_expr
```

RSL Example: Sets. Narrative: *H* stand for the domain type of street intersections – we shall call then hubs, and let *L* stand for the domain type of segments of streets between immediately neighboring hubs – we shall call then links. Then *Hs* and *Ls* are to designate the types of finite sets of zero, one or more hubs, respectively links. **Formalisation:**

```
type H, L, Hs=H-set, Ls=L-set •
```

RSL Example: Cartesians. Narrative: Let *RN* stand for the domain type of road nets consisting of hub aggregates, *HA*, and link aggregates, *LA*. Hub and link aggregates can be observed from road nets, and hub sets and link sets can be observed from hub, respectively link aggregates. **Formalisation:**

```
type RN = HA×LA, Hs, Ls
value obs_HA: RN→HA, obs_LA: RN− LA, obs_Hs: HA→Hs, obs_Ls: LA→Ls
```

Observer functions, obs_... are not further defined – beyond their signatures. They will (subsequently) be defined through axioms over their results •

Some schematic type definitions are:

Variety of Type Definitions

```
[18]  Type_name = Type_expr /* without |s or subtypes */
[19]  Type_name = Type_expr_1 | Type_expr_2 | ... | Type_expr_n
[20]  Type_name ==
           mk_id_1(s_a1:Type_name_a1,...,s_ai:Type_name_ai) |
           ... |
           mk_id_n(s_z1:Type_name_z1,...,s_zk:Type_name_zk)
[21]  Type_name :: sel_a:Type_name_a ... sel_z:Type_name_z
[22]  Type_name = {| v:Type_name′ • 𝒫(v) |}
```

where a form of [19–20] is provided by combining the types:

Record Types

```
[23]  Type_name = A | B | ... | Z
[24]  A == mk_id_1(s_a1:A_1,...,s_ai:A_i)
```

```
[25]  B == mk_id_2(s_b1:B_1,...,s_bj:B_j)
[26]  ...
[27]  Z == mk_id_n(s_z1:Z_1,...,s_zk:Z_k)
```

Of these we shall almost exclusively make use of [23–27].

Disjoint Types. Narrative: A pipeline consists of a finite set of zero, one or more [interconnected][42] pipe units. Pipe units are either wells, or are pumps, or are valves, or are joins, or are forks, or are sinks. **Formalisation:**

```
type PL = P-set, P == WU|PU|VA|JO|FO|SI, Wu,Pu,Vu,Ju,Fu,Su
     WU::mkWU(swu:Wu), PU::mkPU(spu:Pu), VA::mkVU(svu:Vu),
     JO::mkJu(sju:Ju), FO::mkFu(sfu:Fu), SI::mkSi(ssu:Su)
```

where we leave types Wu, Pu, Vu, Ju, Fu and Su further undefined •
Types A, B, ..., Z are disjoint, i.e., shares no values, provided all mk_id_k are distinct and due to the use of the disjoint record type constructor ==.

```
axiom
   ∀ a1:A_1, a2:A_2, ..., ai:Ai •
      s_a1(mk_id_1(a1,a2,...,ai))=a1 ∧ s_a2(mk_id_1(a1,a2,...,ai))=a2 ∧
      ... ∧ s_ai(mk_id_1(a1,a2,...,ai))=ai ∧
   ∀ a:A • let mk_id_1(a1′,a2′,...,ai′) = a in
      a1′ = s_a1(a) ∧ a2′ = s_a2(a) ∧ ... ∧ ai′ = s_ai(a) end
```

Note: Values of type A, where that type is defined by A::B×C×D, can be expressed A(b,c,d) for b:B, c:D, d:D.

A.1.2.3 Subtypes

In RSL, each type represents a set of values. Such a set can be delimited by means of predicates. The set of values b which have type B and which satisfy the predicate $\mathscr{P}$, constitute the subtype A:

Subtypes

```
type
   A = {| b:B • 𝒫(b) |}
```

Subtype. Narrative: The subtype of even natural numbers.

Formalisation: type ENat = {| en | en:**Nat** • is_even_natural_number(en) |} •

A.2 The Propositional and Predicate Calculi

A.2.1 Propositions

In logic, a proposition is the meaning of a declarative sentence. [A declarative sentence is a type of sentence that makes a statement]

A.2.1.1 Propositional Expressions

Propositional expressions, informally speaking, are quantifier-free expressions having truth (or **chaos**) values. $\forall$, $\exists$ and $\exists\,!$ are quantifiers, see below.

Below, we will first treat propositional expressions all of whose identifiers denote truth values. As we progress, in sections on arithmetic, sets, list, maps, etc., we shall extend the range of propositional expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values (**true** or **false** [or **chaos**]). Then:

Propositional Expressions

false, **true**
a, b, ..., c $\sim$a, a$\wedge$b, a$\vee$b, a$\Rightarrow$b, a=b, a$\neq$b

are propositional expressions having Boolean values. $\sim$, $\wedge$, $\vee$, $\Rightarrow$, $=$, $\neq$ and $\Box$ are Boolean connectives (i.e., operators). They can be read as: **not**, **and**, **or, if then** (or **implies**), **equal, not equal** and **always**.

A.2.1.2 Propositional Calculus

$\mathscr{I}$: Propositional calculus is a branch of logic. It is also called propositional logic, statement logic, sentential calculus, sentential logic, or sometimes zeroth-order logic. It deals with propositions (which can be true or false) and relations between propositions, including the construction of arguments based on them. Compound propositions are formed by connecting propositions by logical connectives. Propositions that contain no logical connectives are called atomic propositions Wikipedia
A simple two-value Boolean logic can be defined as follows:

```
type
    Bool
value
    true, false
    ~: Bool → Bool
    ∧, ∨, ⇒, =, ≠, ≡: Bool × Bool → Bool
axiom
    ∀ b,b′:Bool •
        ~b ≡ if b then false else true end
        b ∧ b′ ≡ if b then b′ else false end
```

$\mathsf{b} \vee \mathsf{b}' \equiv$ **if** b **then true else** b' **end**
$\mathsf{b} \Rightarrow \mathsf{b}' \equiv$ **if** b **then** b' **else true end**
$\mathsf{b} = \mathsf{b}' \equiv$ **if** $(\mathsf{b}\wedge\mathsf{b}')\vee(\sim\mathsf{b}\wedge\sim\mathsf{b}')$ **then true else false end**
$(\mathsf{b} \neq \mathsf{b}') \equiv \sim(\mathsf{b} = \mathsf{b}')$
$(\mathsf{b} \equiv \mathsf{b}') \equiv (\mathsf{b} = \mathsf{b}')$

We shall, however, make use of a three-value Boolean logic. The model-theory explanation of the meaning of propositional expressions is now given in terms of the *truth tables* for the logic connectives:

$\vee, \wedge,$ **and** $\Rightarrow$ **Syntactic Truth Tables**

$\vee$	**true**	**false**	**chaos**
true	**true**	**true**	**true**
false	**true**	**false**	**chaos**
chaos	**chaos**	**chaos**	**chaos**

$\wedge$	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**false**	**false**	**false**
chaos	**chaos**	**chaos**	**chaos**

$\Rightarrow$	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**true**	**true**	**true**
chaos	**chaos**	**chaos**	**chaos**

The two-value logic defined earlier 'transpires' from the **true,false** columns and rows of the above truth tables.

A.2.2 Predicates

$\mathscr{I}$: Predicates are mathematical assertions that contains variables, sometimes referred to as predicate variables, and may be true or false depending on those variables' value or values[43]

A.2.2.1 Predicate Expressions

Let x, y, ..., z (or term expressions) designate non-Boolean values, and let $\mathscr{P}(x)$, $\mathscr{Q}(y)$ and $\mathscr{R}(z)$ be propositional or predicate expressions, then:

Simple Predicate Expressions

[28] $\forall \mathsf{x}{:}\mathsf{X} \bullet \mathscr{P}(x)$
[29] $\exists \mathsf{y}{:}\mathsf{Y} \bullet \mathscr{Q}(y)$
[30] $\exists ! \mathsf{z}{:}\mathsf{Z} \bullet \mathscr{R}(z)$

are quantified, i.e., predicate expressions. $\forall$, $\exists$ and $\exists\,!$ are the quantifiers.

[43] https://calcworkshop.com/logic/predicate-logic/, and: predicate logic, first-order logic or quantified logic is a formal language in which propositions are expressed in terms of predicates, variables and quantifiers. It is different from propositional logic which lacks quantifiers https://brilliant.org/wiki/predicate-logic/.

A.2.2.2 Predicate Calculus

They are “read” as:

[28] For all x (values in type X) the predicate $\mathscr{P}(x)$ holds – if that is not the case the expression yields truth value **false**.

[29] There exists (at least) one y (value in type Y) such that the predicate $\mathscr{Q}(y)$ holds – if that is not the case the expression yields truth value **false**.

[30] There exists a unique z (value in type Z) such that the predicate $\mathscr{R}(z)$ holds – if that is not the case the expression yields truth value **false**.

[28–30] The predicates $\mathscr{P}(x)$, $\mathscr{Q}(y)$ or $\mathscr{R}(z)$ may yield **chaos** in which case the whole expression yields **chaos**.

A.3 Arithmetics

$\mathscr{I}$: RSL offers the usual set of arithmetic operators. From these the usual kind of arithmetic expressions can be formed.

Arithmetic

type
  Nat, **Int**, **Real**
value
  $+,-,*$: $\mathbf{Nat}\times\mathbf{Nat}\to\mathbf{Nat} \mid \mathbf{Int}\times\mathbf{Int}\to\mathbf{Int} \mid \mathbf{Real}\times\mathbf{Real}\to\mathbf{Real}$
  $/$: $\mathbf{Nat}\times\mathbf{Nat}\overset{\sim}{\to}\mathbf{Nat} \mid \mathbf{Int}\times\mathbf{Int}\overset{\sim}{\to}\mathbf{Int} \mid \mathbf{Real}\times\mathbf{Real}\overset{\sim}{\to}\mathbf{Real}$
  $<,\leq,=,\neq,\geq,>$ $(\mathbf{Nat}|\mathbf{Int}|\mathbf{Real}) \to (\mathbf{Nat}|\mathbf{Int}|\mathbf{Real})$

A.4 Comprehensive Expressions

$\mathscr{I}$: Comprehensive expressions are common in mathematics texts. They capture properties conveniently abstractly

A.4.1 Set Enumeration and Comprehension

A.4.1.1 Set Enumeration

Let the below a’s denote values of type A:

Set Enumerations

$\{\{\}, \{a\}, \{e_1,e_2,...,e_n\}, ...\} \in$ A-**set**
$\{\{\}, \{a\}, \{e_1,e_2,...,e_n\}, ..., \{e_1,e_2,...\}\} \in$ A-**infset**

A.4.1.2 Set Comprehension

The expression, last line below, to the right of the ≡, expresses set comprehension. The expression "builds" the set of values satisfying the given predicate. It is abstract in the sense that it does not do so by following a concrete algorithm.

Set Comprehension

```
type
    A, B
    P = A → Bool
    Q = A →̃ B
value
    comprehend: A-infset × P × Q → B-infset
    comprehend(s,P,Q) ≡ { Q(a) | a:A • a ∈ s ∧ P(a)}
```

A.4.1.3 Cartesian Enumeration

Let *e* range over values of Cartesian types involving *A*, *B*, ..., *C*, then the below expressions are simple Cartesian enumerations:

Cartesian Enumerations

```
type
    A, B, ..., C
    A × B × ... × C
value
    (e1,e2,...,en)
```

A.4.2 List Enumeration and Comprehension

A.4.2.1 List Enumeration

Let *a* range over values of type *A*, then the below expressions are simple list enumerations:

List Enumerations

```
    {⟨⟩, ⟨e⟩, ..., ⟨e1,e2,...,en⟩, ...} ∈ A*
    {⟨⟩, ⟨e⟩, ..., ⟨e1,e2,...,en⟩, ..., ⟨e1,e2,...,en,... ⟩, ...} ∈ Aω

    ⟨ a_i .. a_j ⟩
```

The last line above assumes a_i and a_j to be integer-valued expressions. It then expresses the set of integers from the value of e_i to and including the value of e_j. If the latter is smaller than the former, then the list is empty.

A.4.2.2 List Comprehension

The last line below expresses list comprehension.

List Comprehension

type
  A, B, P = A $\to$ **Bool**, Q = A $\overset{\sim}{\to}$ B
value
  comprehend: $A^\omega \times P \times Q \overset{\sim}{\to} B^\omega$
  comprehend(l,P,Q) $\equiv \langle$ Q(l(i)) | i **in** $\langle$1..**len** l$\rangle$ • P(l(i))$\rangle$

A.4.3 Map Enumeration and Comprehension

A.4.3.1 Map Enumeration

Let (possibly indexed) u and v range over values of type $T1$ and $T2$, respectively, then the below expressions are simple map enumerations:

Map Enumerations

type
  T1, T2
  M = T1 $\xrightarrow[m]{}$ T2
value
  u,u1,u2,...,un:T1, v,v1,v2,...,vn:T2
  [], [u$\mapsto$v], ..., [u1$\mapsto$v1,u2$\mapsto$v2,...,un$\mapsto$vn] $\forall \in$ M

A.4.3.2 Map Comprehension

The last line below expresses map comprehension:

Map Comprehension

type
  U, V, X, Y
  M = U $\xrightarrow[m]{}$ V
  F = U $\overset{\sim}{\to}$ X
  G = V $\overset{\sim}{\to}$ Y
  P = U $\to$ **Bool**
value
  comprehend: M$\times$F$\times$G$\times$P $\to$ (X $\xrightarrow[m]{}$ Y)

comprehend(m,F,G,P) ≡ [F(u) ↦ G(m(u)) | u:U • u ∈ **dom** m ∧ P(u)]

A.5 Operations

A.5.1 Set Operations

A.5.1.1 Set Operator Signatures

Set Operator Signatures

value
18 ∈: A × A-**infset** → **Bool**
19 ∉: A × A-**infset** → **Bool**
20 ∪: A-**infset** × A-**infset** → A-**infset**
21 ∪: (A-**infset**)-**infset** → A-**infset**
22 ∩: A-**infset** × A-**infset** → A-**infset**
23 ∩: (A-**infset**)-**infset** → A-**infset**
24 \\: A-**infset** × A-**infset** → A-**infset**
25 ⊂: A-**infset** × A-**infset** → **Bool**
26 ⊆: A-**infset** × A-**infset** → **Bool**
27 =: A-**infset** × A-**infset** → **Bool**
28 ≠: A-**infset** × A-**infset** → **Bool**
29 **card**: A-**infset** $\tilde{\to}$ **Nat**

A.5.1.2 Set Operation Examples

Set Operation Examples

examples
a ∈ {a,b,c}
a ∉ {}, a ∉ {b,c}
{a,b,c} ∪ {a,b,d,e} = {a,b,c,d,e}
∪{{a},{a,b},{a,d}} = {a,b,d}
{a,b,c} ∩ {c,d,e} = {c}
∩{{a},{a,b},{a,d}} = {a}
{a,b,c} \ {c,d} = {a,b}
{a,b} ⊂ {a,b,c}
{a,b,c} ⊆ {a,b,c}
{a,b,c} = {a,b,c}
{a,b,c} ≠ {a,b}

card {} = 0, **card** {a,b,c} = 3

A.5.1.3 Informal Set Operator Explication

The following is **not** a definition of RSL semantics. In RSL formulas we present an explication of RSL operators. Read, what appears as definitions, ≡, as [a kind of] identities.

18. ∈: The membership operator expresses that an element is a member of a set.
19. ∉: The nonmembership operator expresses that an element is not a member of a set.
20. ∪: The infix union operator. When applied to two sets, the operator gives the set whose members are in either or both of the two operand sets.
21. ∪: The distributed prefix union operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.
22. ∩: The infix intersection operator. When applied to two sets, the operator gives the set whose members are in both of the two operand sets.
23. ∩: The prefix distributed intersection operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.
24. \: The set complement (or set subtraction) operator. When applied to two sets, the operator gives the set whose members are those of the left operand set which are not in the right operand set.
25. ⊆: The proper subset operator expresses that all members of the left operand set are also in the right operand set.
26. ⊂: The proper subset operator expresses that all members of the left operand set are also in the right operand set, and that the two sets are not identical.
27. =: The equal operator expresses that the two operand sets are identical.
28. ≠: The nonequal operator expresses that the two operand sets are not identical.
29. **card**: The cardinality operator gives the number of elements in a finite set.

A.5.1.4 Set Operator Explications

The set operations can be "equated" as follows:

Set Operator Explications

value

$s' \cup s'' \equiv \{\, a \mid a{:}A \bullet a \in s' \vee a \in s'' \,\}$

$s' \cap s'' \equiv \{\, a \mid a{:}A \bullet a \in s' \wedge a \in s'' \,\}$

$s' \setminus s'' \equiv \{\, a \mid a{:}A \bullet a \in s' \wedge a \notin s'' \,\}$

```
    $s' \subseteq s'' \equiv \forall a{:}A \bullet a \in s' \Rightarrow a \in s''$
    $s' \subset s'' \equiv s' \subseteq s'' \wedge \exists a{:}A \bullet a \in s'' \wedge a \notin s'$
    $s' = s'' \equiv \forall a{:}A \bullet a \in s' \equiv a \in s'' \equiv s{\subseteq}s' \wedge s'{\subseteq}s$
    $s' \neq s'' \equiv s' \cap s'' \neq \{\}$
    card $s \equiv$
        if $s = \{\}$ then $0$ else
        let $a{:}A \bullet a \in s$ in $1 +$ card $(s \setminus \{a\})$ end end
        pre $s$ /* is a finite set */
    card $s \equiv$ chaos /* tests for infinity of s */
```

A.5.2 Cartesian Operations

Cartesian Operations

```
type
    A, B, C
    g0: $G0 = A \times B \times C$
    g1: $G1 = ( A \times B \times C )$
    g2: $G2 = ( A \times B ) \times C$
    g3: $G3 = A \times ( B \times C )$

value
    va:A, vb:B, vc:C, vd:D
    (va,vb,vc):G0,
    (va,vb,vc):G1
    ((va,vb),vc):G2
    (va3,(vb3,vc3)):G3

decomposition expressions
    let $(a1,b1,c1) = g0$,
          $(a1',b1',c1') = g1$ in .. end
    let $((a2,b2),c2) = g2$ in .. end
    let $(a3,(b3,c3)) = g3$ in .. end
```

A.5.3 List Operations

A.5.3.1 List Operator Signatures

List Operator Signatures

```
value
    hd: $A^\omega \xrightarrow{\sim} A$
    tl: $A^\omega \xrightarrow{\sim} A^\omega$
    len: $A^\omega \xrightarrow{\sim}$ Nat
    inds: $A^\omega \rightarrow$ Nat-infset
    elems: $A^\omega \rightarrow$ A-infset
    .(.): $A^\omega \times$ Nat $\xrightarrow{\sim} A$
    $\frown$: $A^* \times A^\omega \rightarrow A^\omega$
    $=$: $A^\omega \times A^\omega \rightarrow$ Bool
```

$\neq$: $\mathsf{A}^{\omega} \times \mathsf{A}^{\omega} \rightarrow$ **Bool**

A.5.3.2 List Operation Examples

List Operation Examples

examples
  hd⟨a1,a2,...,am⟩=a1
  tl⟨a1,a2,...,am⟩=⟨a2,...,am⟩
  len⟨a1,a2,...,am⟩=m
  inds⟨a1,a2,...,am⟩={1,2,...,m}
  elems⟨a1,a2,...,am⟩={a1,a2,...,am}
  ⟨a1,a2,...,am⟩(i)=ai
  ⟨a,b,c⟩$\widehat{\ }$⟨a,b,d⟩ = ⟨a,b,c,a,b,d⟩
  ⟨a,b,c⟩=⟨a,b,c⟩
  ⟨a,b,c⟩ ≠ ⟨a,b,d⟩

A.5.3.3 Informal List Operator Explication

The following is **not** a definition of RSL semantics. In RSL formulas we present an explication of RSL operators. Read, what appears as definitions, ≡, as [a kind of] identities.

- **hd**: Head gives the first element in a nonempty list.
- **tl**: Tail gives the remaining list of a nonempty list when Head is removed.
- **len**: Length gives the number of elements in a finite list.
- **inds**: Indices give the set of indices from 1 to the length of a nonempty list. For empty lists, this set is the empty set as well.
- **elems**: Elements gives the possibly infinite set of all distinct elements in a list.
- $\ell(i)$: Indexing with a natural number, i larger than 0, into a list ℓ having a number of elements larger than or equal to i, gives the ith element of the list.
- $\widehat{\ }$: Concatenates two operand lists into one. The elements of the left operand list are followed by the elements of the right. The order with respect to each list is maintained.
- $=$: The equal operator expresses that the two operand lists are identical.
- $\neq$: The nonequal operator expresses that the two operand lists are not identical.

The operations can also be defined as follows:

A.5.3.4 List Operator Explications

The following is **not** a definition of RSL semantics. In RSL formulas we present an explication of RSL operators. Read, what appears as definitions, ≡, as [a kind of] identities.

List Operator Explications

```
value
   is_finite_list: A^ω → Bool

   len q ≡
      case is_finite_list(q) of
         true → if q = ⟨⟩ then 0 else 1 + len tl q end,
         false → chaos end

   inds q ≡
      case is_finite_list(q) of
         true → { i | i:Nat • 1 ≤ i ≤ len q },
         false → { i | i:Nat • i≠0 } end

   elems q ≡ { q(i) | i:Nat • i ∈ inds q }

   q(i) ≡
      if i=1
         then
            if q≠⟨⟩
               then let a:A,q′:Q • q=⟨a⟩⌢q′ in a end
               else chaos end
         else q(i−1) end

   fq ⌢ iq ≡
         ⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i − len fq) end
          | i:Nat • if len iq≠chaos then i ≤ len fq+len end ⟩
      pre is_finite_list(fq)

   iq′ = iq″ ≡
      inds iq′ = inds iq″ ∧ ∀ i:Nat • i ∈ inds iq′ ⇒ iq′(i) = iq″(i)

   iq′ ≠ iq″ ≡ ∼(iq′ = iq″)
```

A.5.4 Map Operations

A.5.4.1 Map Operator Signatures

Map Operator Signatures

value
[30] $\cdot(\cdot)$: $\mathsf{M} \to \mathsf{A} \overset{\sim}{\to} \mathsf{B}$
[31] **dom**: $\mathsf{M} \to \mathsf{A}$-**infset** [domain of map]
[32] **rng**: $\mathsf{M} \to \mathsf{B}$-**infset** [range of map]
[33] †: $\mathsf{M} \times \mathsf{M} \to \mathsf{M}$ [override extension]
[34] $\cup$: $\mathsf{M} \times \mathsf{M} \to \mathsf{M}$ [merge $\cup$]
[35] $\backslash$: $\mathsf{M} \times \mathsf{A}$-**infset** $\to \mathsf{M}$ [restriction by]
[36] $/$: $\mathsf{M} \times \mathsf{A}$-**infset** $\to \mathsf{M}$ [restriction to]
[37] $=,\neq$: $\mathsf{M} \times \mathsf{M} \to$ **Bool**
[38] $^\circ$: $(\mathsf{A} \xrightarrow[m]{} \mathsf{B}) \times (\mathsf{B} \xrightarrow[m]{} \mathsf{C}) \to (\mathsf{A} \xrightarrow[m]{} \mathsf{C})$ [composition]

A.5.4.2 Map Operation Examples

Map Operation Examples

value
[30] $\mathsf{m(a) = b}$
[31] **dom** $[\mathsf{a1 \mapsto b1, a2 \mapsto b2, \ldots, an \mapsto bn}] = \{\mathsf{a1, a2, \ldots, an}\}$
[32] **rng** $[\mathsf{a1 \mapsto b1, a2 \mapsto b2, \ldots, an \mapsto bn}] = \{\mathsf{b1, b2, \ldots, bn}\}$
[33] $[\mathsf{a \mapsto b, a' \mapsto b', a'' \mapsto b''}] \dagger [\mathsf{a' \mapsto b'', a'' \mapsto b'}] = [\mathsf{a \mapsto b, a' \mapsto b'', a'' \mapsto b'}]$
[34] $[\mathsf{a \mapsto b, a' \mapsto b', a'' \mapsto b''}] \cup [\mathsf{a''' \mapsto b'''}] = [\mathsf{a \mapsto b, a' \mapsto b', a'' \mapsto b'', a''' \mapsto b'''}]$
[35] $[\mathsf{a \mapsto b, a' \mapsto b', a'' \mapsto b''}] \backslash \{\mathsf{a}\} = [\mathsf{a' \mapsto b', a'' \mapsto b''}]$
[37] $[\mathsf{a \mapsto b, a' \mapsto b', a'' \mapsto b''}] / \{\mathsf{a', a''}\} = [\mathsf{a' \mapsto b', a'' \mapsto b''}]$
[38] $[\mathsf{a \mapsto b, a' \mapsto b'}] \circ [\mathsf{b \mapsto c, b' \mapsto c', b'' \mapsto c''}] = [\mathsf{a \mapsto c, a' \mapsto c'}]$

A.5.4.3 Informal Map Operation Explication

- *m*(*a*): Application gives the element that *a* maps to in the map *m*.
- **dom**: Domain/Definition Set gives the set of values which maps to in a map.
- **rng**: Range/Image Set gives the set of values which are mapped to in a map.
- †: Override/Extend. When applied to two operand maps, it gives the map which is like an override of the left operand map by all or some "pairings" of the right operand map.
- $\cup$: Merge. When applied to two operand maps, it gives a merge of these maps.

- $\backslash$: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements that are not in the right operand set.

- /: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements of the right operand set.

- $=$: The equal operator expresses that the two operand maps are identical.

- $\neq$: The nonequal operator expresses that the two operand maps are not identical.

- $^\circ$: Composition. When applied to two operand maps, it gives the map from definition set elements of the left operand map, m_1, to the range elements of the right operand map, m_2, such that if a is in the definition set of m_1 and maps into b, and if b is in the definition set of m_2 and maps into c, then a, in the composition, maps into c.

A.5.4.4 Map Operator Explication

The following is **not** a definition of RSL semantics. In RSL formulas we present an explication of RSL operators. Read, what appears as definitions, $\equiv$, as [a kind of] identities.

The map operations can also be defined as follows:

Map Operator Explications

```
value
    rng m ≡ { m(a) | a:A • a ∈ dom m }

    m1 † m2 ≡
        [ a↦b | a:A,b:B •
            a ∈ dom m1 \ dom m2 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

    m1 ∪ m2 ≡ [ a↦b | a:A,b:B •
                    a ∈ dom m1 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

    m \ s ≡ [ a↦m(a) | a:A • a ∈ dom m \ s ]
    m / s ≡ [ a↦m(a) | a:A • a ∈ dom m ∩ s ]

    m1 = m2 ≡
        dom m1 = dom m2 ∧ ∀ a:A • a ∈ dom m1 ⇒ m1(a) = m2(a)
    m1 ≠ m2 ≡ ∼(m1 = m2)

    m°n ≡
        [ a↦c | a:A,c:C • a ∈ dom m ∧ c = n(m(a)) ]
        pre rng m ⊆ dom n
```

A.6 λ-Calculus + Functions

$\mathscr{I}$: The λ-Calculus is a foundation for the abstract specification language that RSL is

A.6.1 The λ-Calculus Syntax

λ-Calculus Syntax

```
type /∗ A BNF Syntax: ∗/
    ⟨L⟩ ::= ⟨V⟩ | ⟨F⟩ | ⟨A⟩ | ( ⟨A⟩ )
    ⟨V⟩ ::= /∗ variables, i.e. identifiers ∗/
    ⟨F⟩ ::= λ⟨V⟩ • ⟨L⟩
    ⟨A⟩ ::= ( ⟨L⟩⟨L⟩ )
value /∗ Examples ∗/
    ⟨L⟩: e, f, a, ...
    ⟨V⟩: x, ...
    ⟨F⟩: λ x • e, ...
    ⟨A⟩: f a, (f a), f(a), (f)(a), ...
```

A.6.2 Free and Bound Variables

Free and Bound Variables

Let x, y be variable names and e, f be λ-expressions.

- ⟨V⟩: Variable x is free in x.
- ⟨F⟩: x is free in $\lambda y \bullet e$ if $x \neq y$ and x is free in e.
- ⟨A⟩: x is free in $f(e)$ if it is free in either f or e (i.e., also in both).

A.6.3 Substitution

In RSL, the following rules for substitution apply:

Substitution

- **subst**([N/x]x) ≡ N;
- **subst**([N/x]a) ≡ a,

  for all variables a≠ x;
- **subst**([N/x](P Q)) ≡ (**subst**([N/x]P) **subst**([N/x]Q));
- **subst**([N/x]($\lambda x \bullet P$)) ≡ λ y•P;

- **subst**([N/x](λ y•P)) $\equiv$ λy• **subst**([N/x]P),

  if x$\neq$y and y is not free in N or x is not free in P;

- **subst**([N/x](λy•P)) $\equiv$ λz•**subst**([N/z]**subst**([z/y]P)),

  if y$\neq$x and y is free in N and x is free in P
  (where z is not free in (N P)).

A.6.4 α-Renaming and β-Reduction

α and β Conversions

- α-renaming: λx•M

  If x, y are distinct variables then replacing x by y in λx•M results in λy•**subst**([y/x]M). We can rename the formal parameter of a λ-function expression provided that no free variables of its body M thereby become bound.

- β-reduction: (λx•M)(N)

  All free occurrences of x in M are replaced by the expression N provided that no free variables of N thereby become bound in the result. (λx•M)(N) $\equiv$ **subst**([N/x]M)

A.6.5 Function Signatures

For sorts we may want to postulate some functions:

Sorts and Function Signatures

```
type
   A, B, C
value
   obs_B: A → B,
   obs_C: A → C,
   gen_A: B×C → A
```

A.6.6 Function Definitions

Functions can be defined explicitly:

Explicit Function Definitions

```
value
```

f: Arguments → Result
f(args) ≡ DValueExpr

g: Arguments $\overset{\sim}{\to}$ Result
g(args) ≡ ValueAndStateChangeClause
pre P(args)

Or functions can be defined implicitly:

Implicit Function Definitions

value
f: Arguments → Result
f(args) **as** result
post P1(args,result)

g: Arguments $\overset{\sim}{\to}$ Result
g(args) **as** result
pre P2(args)
post P3(args,result)

The symbol $\overset{\sim}{\to}$ indicates that the function is partial and thus not defined for all arguments. Partial functions should be assisted by preconditions stating the criteria for arguments to be meaningful to the function.

A.7 Other Applicative Expressions

$\mathscr{I}$: RSL offers the usual collection of applicative constructs that functional programming languages (Standard ML [55, 55] or F# [46]) offer

A.7.1 Simple let Expressions

Simple (i.e., nonrecursive) **let** expressions:

Let Expressions

let a = $\mathscr{E}_d$ **in** $\mathscr{E}_b$(a) **end**

is an “expanded” form of:

(λa.$\mathscr{E}_b$(a))($\mathscr{E}_d$)

A.7.2 Recursive let Expressions

Recursive **let** expressions are written as:

Recursive **let** Expressions

let $f = \lambda a{:}A \bullet E(f)$ **in** $B(f,a)$ **end**

is "the same" as:

let $f = \mathbf{Y}F$ **in** $B(f,a)$ **end**

where:

$F \equiv \lambda g{\bullet}\lambda a{\bullet}(E(g))$ and $YF = F(YF)$

A.7.3 Predicative let Expressions

Predicative **let** expressions:

Predicative **let** Expressions

let $a{:}A \bullet \mathscr{P}(a)$ **in** $\mathscr{B}(a)$ **end**

express the selection of a value a of type A which satisfies a predicate $\mathscr{P}(a)$ for evaluation in the body $\mathscr{B}(a)$.

A.7.4 Pattern and "Wild Card" let Expressions

Patterns and wild cards can be used:

Patterns

let $\{a\} \cup s = set$ **in** ... **end**
let $\{a,\_\} \cup s = set$ **in** ... **end**

let $(a,b,...,c) = cart$ **in** ... **end**
let $(a,\_,...,c) = cart$ **in** ... **end**

let $\langle a\rangle^\frown\ell = list$ **in** ... **end**
let $\langle a,\_,b\rangle^\frown\ell = list$ **in** ... **end**

let $[a\mapsto b] \cup m = map$ **in** ... **end**
let $[a\mapsto b,\_] \cup m = map$ **in** ... **end**

A.7.4.1 Conditionals

Various kinds of conditional expressions are offered by RSL:

Conditionals

```
if b_expr then c_expr else a_expr
end

if b_expr then c_expr end ≡ /∗ same as: ∗/
   if b_expr then c_expr else skip end

if b_expr_1 then c_expr_1
elsif b_expr_2 then c_expr_2
elsif b_expr_3 then c_expr_3
...
elsif b_expr_n then c_expr_n end

case expr of
   choice_pattern_1 → expr_1,
   choice_pattern_2 → expr_2,
   ...
   choice_pattern_n_or_wild_card → expr_n
end
```

A.7.5 Operator/Operand Expressions

Operator/Operand Expressions

```
⟨Expr⟩ ::=
          ⟨Prefix_Op⟩ ⟨Expr⟩
         | ⟨Expr⟩ ⟨Infix_Op⟩ ⟨Expr⟩
         | ⟨Expr⟩ ⟨Suffix_Op⟩
         | ...
⟨Prefix_Op⟩ ::=
         − | ∼ | ∪ | ∩ | card | len | inds | elems | hd | tl | dom | rng
⟨Infix_Op⟩ ::=
          = | ≠ | ≡ | + | − | ∗ | ↑ | / | < | ≤ | ≥ | > | ∧ | ∨ | ⇒
         | ∈ | ∉ | ∪ | ∩ | \ | ⊂ | ⊆ | ⊇ | ⊃ | ^ | † | °
⟨Suffix_Op⟩ ::= !
```

A.8 Imperative Constructs

We ski taerptment of the imperative constructs of RSL !

A.9 Process Constructs

𝒮: RSL offers several of the constructs that CS [47] offers

A.9.1 Process Channels

As for channels we deviate from common RSL [41] in that we directly *declare* channels – and not via common RSL *objects* etc.

Let A and B stand for two types of (channel) messages and i:KIdx for channel array indexes, then:

Process Channels

channel c:A
channel { k[i]:B • i:Idx }
channel { k[i,j,...,k]:B • i:Idx,j:Jdx,...,k:Kdx }

declare a channel, c, and a set (an array) of channels, k[i], capable of communicating values of the designated types (A and B).

A.9.2 Process Composition

Let P and Q stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, thereby communicating over declared channels. Let P() and Q stand for process expressions, then:

Process Composition

P ‖ Q	Parallel composition
P ⌷ Q	Nondeterministic external choice (either/or)
P ⌈⌉ Q	Nondeterministic internal choice (either/or)
P ╢ Q	Interlock parallel composition

express the parallel (‖) of two processes, or the nondeterministic choice between two processes: either external (⌷) or internal (⌈⌉). The interlock (╢) composition expresses that the two processes are forced to communicate only with one another, until one of them terminates.

A.9.3 Input/Output Events

Let c, k[i] and e designate channels of type A and B, then:

Input/Output Events

```
c ?, k[i] ?      Input
c ! e, k[i] ! e  Output
```

expresses the willingness of a process to engage in an event that "reads" an input, respectively "writes" an output.

A.9.4 Process Definitions

The below signatures are just examples. They emphasise that process functions must somehow express, in their signature, via which channels they wish to engage in input and output events.

Process Definitions

```
value
    P: Unit → in c out k[i]
    Unit
    Q: i:KIdx →  out c in k[i] Unit

    P() ≡ ... c ? ... k[i] ! e ...
    Q(i) ≡ ... k[i] ? ... c ! e ...
```

The process function definitions (i.e., their bodies) express possible events.

A.10 RSL Module Specifications

We shall not include coverage nor use of the RSL module concepts of *schemes, classes* and *objects*.

A.11 Simple RSL Specifications

Often, we do not want to encapsulate small specifications in schemas, classes, and objects, as is often done in RSL. An RSL specification is simply a sequence of one or more types, values (including functions), variables, channels and axioms:

Simple RSL Specifications

```
    type
        ...
```

```
variable
    ...
channel
    ...
value
    ...
axiom
    ...
```

A.12 RSL$^+$: Extended RSL

Section A.1 on page 169 covered standard RSL types. To them we now add two new types: Type names and RSL-Text.

A.12.1 Type Names and Type Name Values

A.12.1.1 Type Names

- Let T be a type name.
- Then ηT is a type name value.
- And $\eta\mathbb{T}$ is the type of type names.

A.12.1.2 Type Name Operations

- η can be considered an operator.
  - It (prefix) applies, then, to type (T) identifiers and yields the name of that type.
  - Two type names, nT_i, nT_j, can be compared for equality: $\mathsf{nT}_i = \mathsf{nT}_j$ iff $i = j$.
- It, vice-versa, suffix applies to type name (nT) identifiers and yields the name, T, of that type: $\mathsf{nT}\eta = \mathsf{T}$.

A.12.2 RSL-Text

A.12.2.1 The RSL-Text Type and Values

- RSL-Text is the type name for ordinary, non-extended RSL texts.

We shall not here give a syntax for ordinary, non-extended RSL texts – but refer to [41].

A.12.2.2 RSL-Text Operations

- RSL-Texts can be compared and concatenated:
  - $\text{rsl-text}_a = \text{rsl-text}_b$
  - $\text{rsl-text}_a \widehat{\ }\, \text{rsl-text}_b$

The $\widehat{\ }$ operator thus also applies, besides, lists (tuples), to RSL texts – treating RSL texts as (if they were) lists of characters.

A.13 Distributive Clauses

We clarify:

A.13.1 Over Simple Values

```
$\oplus$ { a | a:A • a $\in$ {a_1,a_2,...,a_n} } =
    if n>0 then a_1$\oplus$a_1$\oplus$...$\oplus$a_n else
        case $\oplus$ of
            + $\rightarrow$ 0, $-$ $\rightarrow$ 0, $*$ $\rightarrow$ 1, / $\rightarrow$ chaos, $\cup$ $\rightarrow$ {}, $\cap$ $\rightarrow$ {}, ...
        end end
```

```
(f_1,f_2,...,f_n)(a) $\equiv$ if n>0 then (f_1(a),f_2(a),...,f_n(a)) else chaos end
```

A.13.2 Over Processes

```
$\|$ { p(i) | i:I • i $\in$ {i_1,i_2,...,i_n} } $\equiv$ if n>0 then p(i_1)$\|$p(i_2)$\|$...$\|$p(i_n) else () end
$\sqcap$ { p(i) | i:I • i $\in$ {i_1,i_2,...,i_n} } $\equiv$ if n>0 then p(i_1)$\sqcap$p(i_2)$\sqcap$...$\sqcap$p(i_n) else () end
$\Box$ { p(i) | i:I • i $\in$ {i_1,i_2,...,i_n} } $\equiv$ if n>0 then p(i_1)$\Box$p(i_2)$\Box$...$\Box$p(i_n) else () end
```

A.14 Space and Time

The concepts of space and time can be *transcendentally deduced*, by rational reasoning, as has been shown in [60, 61, 62, 63, 64, *Kai Sørlander*], from the facts of *symmetry, asymmetry, transitivity* and *intransitivity* relations.

They are therefore facts of every possible universe.

In this section, i.e., Sect. A.14, we shall distinguish between mathematical "types" and RSL-Text types.

A.14.1 Space

There is one given space. As a mathematical type we name it $\mathbb{SPACE}$. We do not present models of $\mathbb{SPACE}$. But we do introduce such the mathematical type notion of $\mathbb{POINT}$ (with $\mathbb{SPACE}$ being some dense and infinite collection of points), and a corresponding `RSL`-Text type notion of $\mathbb{LOCATION}$: as the location in space of some endurant e:E[44]; We do not bother, here, about textual representation of spatial locations, but here is an example that would work in or near this globe we call our earth: `Latitude 55.805600, Longitude 12.448160, Altitude 35 m`[45].

value **record_**$\mathbb{LOCATION}$: E $\to$ $\mathbb{LOCATION}$

We leave it unspecified as to which $\mathbb{POINT}$ of the sub$\mathbb{SPACE}$ taken up by the endurant *e* that serves as a (or the) location reference point.

Further `RSL`-Text types are: $\mathbb{CURVE}$: as an infinite collection of locations forming a mathematical curve – having a (finite or infinite) *length*; $\mathbb{SURFACE}$: as an infinite collection of locations forming a mathematical surface – having a (finite or infinite) *area*; and $\mathbb{VOLUME}$: as an infinite collection of locations forming a mathematical volume – having a (finite or infinite) *volume*. The derived notios of LENGTH, AREA, VOLUME are `RSL`-Text types:

LENGTH **m**, AREA $\mathbf{m}^2$, and VOLUME $\mathbf{m}^3$

We suggest, as a domain science & engineering research topic, that somebody studies *a calculus or calculi of spatial modelling*.

A.14.2 Time

There is one given time. As a mathematical and also as an `RSL`-Text type we name it $\mathbb{TIME}$. We do not bother, here, about textual representation of time, but here is an example: `August 18, 2025: 12:03`[46]. But we do introduce such crucial notions as ("positive" or "negative" valued) *time intervals* $\mathbb{TI}$ and operations on $\mathbb{TIME}$ and $\mathbb{TI}$:

value

$-$: $\mathbb{TIME}\times\mathbb{TIME}\to\mathbb{TI}$

$+$: $\mathbb{TIME}\times\mathbb{TI}\to\mathbb{TIME}$

$*$: $\textbf{Real}\times\mathbb{TI}\to\mathbb{TI}$

A crucial time-related operation is that of **record_**$\mathbb{TIME}$. It applies to "nothing": **record_**$\mathbb{TIME}$() and yields $\mathbb{TIME}$.

value **record_**$\mathbb{TIME}$: **Unit** $\to$ $\mathbb{TIME}$

[44] Endurant is a notion defined in [20]

[45] An approximation of the author's house location !

[46] The time this text was last compiled !

Appendix B

The Domain Modeling Theory

Contents

B.1 **Domains** **200**

B.1.1 What are They ? 200

B.1.2 Some Introductory Remarks 201

B.1.2.1 A Discussion of Our Characterization of a Concept of Domain 201

B.1.2.2 Formal Methods and Description Language 201

B.1.2.3 Programming Languages versus Domain Semantics . . 201

B.1.2.4 A New Universe 202

B.2 **Six Languages** **202**

B.2.1 The 6 Languages 202

B.2.2 Semiotics 203

B.2.3 Speech Acts 204

B.3 **Endurants and Perdurants, I** **204**

B.4 **A Domain Analysis & Description Ontology** **204**

B.4.1 The Chosen Ontology 204

B.4.2 Discussion of The Chosen Ontology 206

B.5 **The Name, Type and Value Concepts** **207**

B.5.1 Names 207

B.5.2 Types 207

B.5.3 Values 207

B.6 **Phenomena and Entities** **208**

B.7 **Endurants and Perdurants, II** **208**

B.7.1 Endurants 209

B.7.2 Perdurants 209

B.7.3 Ontological Choice 209

B.8 **External and Internal Endurant Qualities** **209**

B.8.1 External Qualities – Tangibles 210

B.8.1.1 The Universe of Discourse 210

B.8.1.2 Solid and Fluid Endurants 211
B.8.1.3 Parts and Living Species Endurants 212
B.8.1.4 States 216
B.8.1.5 Validity of Endurant Observations 216
B.8.1.6 Summary of Endurant Analysis Predicates 216
B.8.1.7 "Trees are Not Recursive" 217
B.8.2 Internal Qualities – Intangibles 217
B.8.2.1 Unique Identity 218
B.8.2.2 Mereology 219
B.8.2.3 Attributes 220
B.8.3 Intentional Pull 223
B.8.4 Summary of Endurants 224
B.9 **Perdurant Concepts 224**
B.9.1 "Morphing" Parts into Behaviours 224
B.9.2 Transcendental Deduction 225
B.9.3 Actors – A Synopsis 225
B.9.3.1 Action 225
B.9.3.2 Event 226
B.9.3.3 Behaviour 226
B.9.4 Channel 226
B.9.5 Behaviours 227
B.9.5.1 Behaviour Signature 227
B.9.5.2 Inert Arguments: Some Examples 228
B.9.5.3 Behaviour Definitions 228
B.9.5.4 Action Definitions 230
B.9.5.5 Behaviour Invocation 231
B.9.5.6 Argument References 232
B.9.5.7 Behaviour Description – Examples 233
B.9.6 Behaviour Initialization. 235
B.10 **Facets 235**
B.10.1 Intrinsics 235
B.10.2 Support Technology 236
B.10.3 Rules & Regulations 236
B.10.4 Scripts 236
B.10.5 License Languages 236
B.10.6 Management & Organization 236
B.10.7 Human Behaviour 237
B.11 **Conclusion 237**
B.11.1 Previous Literature 237
B.11.2 The Method 238
B.11.3 Specification Units 238
B.11.4 Object Orientation 238

B.11.5 Other Domain Modeling Approaches 238
B.11.6 How Much ? How Little ? 239
B.11.7 Correctness 239
B.11.8 Domain Facets 239
B.11.9 Perspectives 239
B.11.10The Semantics of Domain Models 240
B.11.11Further on Domain Modeling 240
B.11.12Software Development 240
B.11.13Modeling 240
B.11.14Philosophy of Computing 240
B.11.15A Manifesto 240

The Triptych Dogma

In order to *specify* $\mathbb{S}$*oftware*, we must understand its $\mathbb{R}$*equirements*.
In order to *prescribe* $\mathbb{R}$*equirements* we must understand the $\mathbb{D}$*omain*.
So we must study, analyze and describe $\mathbb{D}$*omains*.
$\mathbb{D},\mathbb{S} \models \mathbb{R}$:
In proofs of $\mathbb{S}$*oftware* correctness,
with respect to $\mathbb{R}$*equirements*,
assumptions are made with respect to the $\mathbb{D}$*omain*.

We present a systematic *method*, its *principles*, *procedures*, *techniques* and *tools*, for efficiently *analyzing* & *describing* domains. This paper is based on [16, 17, 20]. It simplifies the methodology of these considerably – as well as introduces some novel presentation and description language concepts.

• • •

Alert: Before You start reading this paper, You are kindly informed of the following:

Highlight: 1 *What The Paper is All About*: The *Triptych Dogma*, above, says it all: this paper is about a new area of computing science – that of *domains*. It is about what domains are. How to model them. And their role in software development. There are many "domain things" it is not about: it is not about 'derived' properties of domains – beyond, for example, *intentional pull* [Sect. B.8.3 on page 223]. Such are left for studies of domains based on the kind of formal domain descriptions such as those advocated by this paper■

Highlight: 2 *A Radically New Approach to Software Development*: The *Triptych Approach to Software Development*, calls for *software* to be developed on the basis of *requirements prescriptions*, themselves developed on the basis of *domain descriptions*. We furthermore advocate these specifications and their development be formal. That is: there are formal methods for the development of either of these three kinds of specifications:

- Development of domain descriptions is outlined in this paper.

- Development of requirements, from domain descriptions, is outlined in [20, *Chapter 9*].
- Development of software, from requirements prescriptions, is treated, extensively, in [9].

The reader should understand that the current paper, with its insistence of strictly following a method, formally, is at odds with current 'software engineering' practices. ■

Highlight: 3 *Characterizations rather than Definitions*: The object of domain study, analysis and description, i.e., the domains, are, necessarily, informal. A resulting domain description is formal. So the domain items being studied and analyzed cannot be given a formal definition. Conventionally [so-called theoretical] computer scientists expect and can seemingly only operate in a world of clearly defined concepts. Not so here. It is not possible. Hence we use the term 'characterization' in lieu of 'definition' ■

Highlight: 4 *Seemingly Fragmented Texts*: The text of this paper is a sequence of enumerated sections, subsections, sub-subsections and paragraphs, with short HIGHLIGHTS, CHARACTERIZATIONS, EXAMPLES, ONTOLOGICAL CHOICES, PROMPTS, SCHEMAS and ordinary short texts. The brevity is intentional. Each and all of these units outline important concepts. Each contain a meaning and can be read "in isolation" ■

B.1 Domains

We start by delineating the informal concept of domain,[47]

B.1.1 What are They ?

What do we mean by 'domain' ?

Characterization: 1 *Domain*: By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants* ■

Example: 1 *Some Domain Examples*: A few, more-or-less self-explanatory examples:

- **Rivers** – with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. – and their conveyage of materials (ships etc.) [25, *Chapter B*].
- **Road nets** – with street segments and intersections, traffic lights and automobiles – and the flow of these [25, *Chapter E*].
- **Pipelines** – with their liquids (oil, or gas, or water), wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids [25, *Chapter I*].

[47] Our use of the term 'domain' should not be confused with that of Dana Scott's Domain Theory: `https://en.wikipedia.org/wiki/Scott_domain`.

- **Container terminals** – with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these[25, *Chapter K*] ■

Characterization 1 on the preceding page relies on the understanding of the terms *'rationally describable'*, *'discrete dynamics'*, *'human assisted'*, *'solid'* and *'fluid'*. The last two will be explained later. By **rationally describable** we mean that what is described can be understood, including reasoned about, in a rational, that is, logical manner – in other words **logically tractable**.[48] By **discrete dynamics** we imply that we shall basically rule out such domain phenomena which have properties which are continuous with respect to their time-wise, i.e., dynamic, behaviour. By **human-assisted** we mean that the domains – that we are interested in modeling – have, as an important property, that they possess man-made entities.

B.1.2 Some Introductory Remarks

B.1.2.1 A Discussion of Our Characterization of a Concept of Domain

Characterization 1 on the facing page is our attempt to delineate the subject area. That is, "our" concept of *'domain'* is 'novel': *new and not resembling something formerly known or used*. As such it may be unfamiliar to most readers. So it takes time to digest that characterization. So the reader may have to return to the page, Page 200, to be reminded of the definition.

B.1.2.2 Formal Methods and Description Language

The reader is assumed to have a reasonable grasp of formal methods – such as espoused in [33, 34, 9, 69].

The descriptions evolving from the modeling approach of this paper are in the abstract, model-oriented specification language `RSL` [41] of the **R**aise[49] **S**pecification **L**anguage. But other abstract specification languages could be used: `VDM` [33, 34, 39], `Z` [69], `Alloy` [49], `CafeOBJ` [40], etc. We have chosen `RSL` since it embodies a variant of `CSP` [47] – being used to express domain behaviours.

B.1.2.3 Programming Languages versus Domain Semantics

From around the late 1960s, spurred on by the works of John McCarthy, Peter Landin, Christopher Strachey, Dana Scott and others, it was not unusual to see publications of entire formal definitions of programming language semantics. Widespread technical reports were [3, 2, 1969, 1974] Notably so was [54, 1976]. There was the 1978 publication [33, *Chapter 5, Algol 60*, 1978]. Others were [34, *Chapters 6–7, Algol 60 and Pascal*, 1982] As late as into the 1980s there were such publications [4, 1980].

Formal descriptions of domains, such as we shall unravel a method for their study, analysis and description, likewise amount to semantics for the terms of the professional languages spoken by stakeholders of domains. So perhaps it is time to take the topic serious.

[48] Another, "upside–down" – after the fact – [perhaps 'cheating'] way of defining 'describable' is: is it describable in terms of the method of this paper !

[49] **RAISE** stands for **R**igorous **A**pproach to **I**ndustrial **S**oftware **E**ngineering [42].

B.1.2.4 **A New Universe**

The concept of domain – such as we shall delineate and treat it – is novel. That is: new and not treated in this way before. Its presentation, therefore, necessarily involves the introduction of a new universe of concepts. Not the neat, well-defined concepts of neither "classical" computer science nor software engineering. It may take some concentration on the part of the reader to get used to this !

You will therefore be introduced to quite a universe of new concepts. You will find these concepts named in most display lines[50] and in Figs. B.1 on page 205 and B.2 on page 222.

B.2 **Six Languages**

This section is an artifice, an expedient.

It summarizes, from an unusual angle, an aspect of the presentation style of this paper. *The road ahead of us introduces rather many new and novel concepts. It is easy to get lost. The presentation alternates, almost sentence-by-sentence, between 5 languages. The below explication might help You to keep track of where the paper eventually shall lead us !* This section, in a sense, tells the story backwards ![51]

B.2.1 **The 6 Languages**

There are 6 languages at play in this paper:

- (i) technical English, as in most papers;
- (ii) RSL, the RAISE Specification Language [41];
- (iii) an augmented RSL language;
- (iv) the domain modeling language – which we can view as the composition of clauses from two [sub-ordinate] languages:
  - (v) a domain analysis language; and
  - (vi) a domain specification

  language.

(i) Technical English is the main medium, as in most papers, of what is conveyed. (ii) Domain descriptions are (to be) expressed in RSL. (iii) The [few places where we resort to the] augmented RSL language is needed for expressing names of RSL types as values. (iv) The domain modeling language consists of finite sequences domain analysis and domain description clauses. (v) The domain analysis language just consists of prompts, i.e., predicate functions used informally by the domain analyzer in inquiring the domain. They yield either truth values or possibly augmented RSL texts. (vi) The domain description language consists of a few RSL text yielding prompts.

[50] – that is, section, subsection, sub-subsection, paragraph and sub-paragraph lines

[51] Søren Kierkegaard: *Life is lived forwards but is understood backwards* [1843].

We presume that the reader is familiar with such languages as RSL. That is: VDM [33, 34, 39], Z [69], Alloy [49], etc. They could all be use instead of, as here, RSL.

We summarize some of the language issues.

The Domain Analysis Language: We list a few, cf. Fig. B.1 on page 205, of the predicate prompts, i.e., language prompts: is_entity [pg 208], is_endurant [pg 209], is_perdurant [pg 209], is_solid [pg 211], is_fluid [pg 211], is_part [pg 212], atomic [pg 212], is_compound [pg 213], is_Cartesian [pg 213], or is_part-set [pg 214]; and the extended RSL text yielding analysis prompts: record_Cartesian_type_names [pg 215], record_part_set_type_names [pg 215] and record_attribute_type_names [pg 221].

The Domain Description Language: RSL. We shall us a subset of RSL. That subset is a simple, discrete mathematics, primarily functional specification language in the style of VDM [33, 34, 39]. Emphasis is on sets, Cartesians, lists, and maps (i.e., finite definition set, enumerable functions).

Domain Description: A domain description consists of one or more domain specification units. A specification unit is of either of 10 kinds, all expressed in RSL. (1) a universe-of-discourse **type** clause [pg 210]; (2) a part **type** and **obs_**erver **value** clause [pg 215]; (3) a **value** clause; (4) a unique identifier **type** and (**uid_**) observer value (function) clause [pg 218]; (5) a mereology **type** and (**mereo_**) observer value (function) clause [pg 220]; (6) an attribute **type** and (**attr_**) observer value (function) definition clause [pg 221]; (7) an **axiom** clause; (8) a **channel** declaration clause [pg 226]; (9) a behaviour **value** (signature and definition) clause [pg 227 & pg 231]; and (10) a domain initialization clause [Sect. B.9.6 on page 235]. These clauses are often combined in 2-3 such clauses, and may, and usually do, include further RSL clauses.

The use of RSL "outside" the domain specification units should not be confused with the RSL of the specification unit schemas and examples.

B.2.2 Semiotics

In *Foundations of the theory of signs* [56] defines semiotics as "consisting" of syntax, semantics and pragmatics.

- **Syntax:** The syntax of domain analysis and domain description clauses are simple atomic clauses consisting of a prompt (predicate or function) identifier, see above, and an identifier denoting a domain entity. The syntax of the domain modeling language prescribes a sequence of one or more domain analysis and domain description clauses.

- **Semantics:** The meaning of a domain analysis clause is that of a function from a domain entity to either a truth value or some augmented RSL text. The meaning of a domain description clause is that of a function from a domain entity to a domain specification unit.

- **Pragmatics:** The pragmatics of a domain analysis predicate clause, as applied to a domain entity e, is that of prompting the domain analyzer to a next domain analysis step: either that of applying a [subsequent, cf. Fig. B.1] domain analysis predicate prompt to e; or applying a [subsequent, cf. Fig. B.1] domain analysis function to e, and noting – as writing down on a "to remember board" – the result of the [latter] query; or applying a [subsequent, cf. Fig. B.1] domain description function to e. The pragmatics of a domain description function is that of including the resulting RSL domain description text in the emerging domain description. There is no hint as to what to do next!

B.2.3 Speech Acts

The above explication of a pragmatics for the domain modeling language relates to the concepts of *speech acts*. We refer to [1, How to do things with words], [59, Speech Acts: An Essay in the Philosophy of Language] and [58, Brain mechanisms linking language and action]. A further study of the *illocutionary* and *locutionary* aspects of the domain analysis language seems in place.

B.3 Endurants and Perdurants, I

The above characterization hinges on the characterizations of endurants and perdurants.

Characterization: 2 *Endurants*: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster] ■

Endurants are either *natural* ["God-given"] or *artefactual* ["man-made"]. Endurants may be either solid (discrete) or fluid, and solid endurants, called parts, may be considered *atomic* or *compound* parts; or, as in this report solid endurants may be further unanalysed *living species: plants* and *animals* – including *humans*.

Characterization: 3 *Perdurants*: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster] ■

Perdurants are here considered to be *actions*, *events* and *behaviours*.

• • •

We exclude, from our treatment of domains, issues of living species, ethics, biology and psychology.

B.4 A Domain Analysis & Description Ontology

B.4.1 The Chosen Ontology

Figure B.1 expresses an ontology[52] for our analysis of domains. Not a taxonomy[53] for any one specific domain.

[52] An ontology is the philosophical study of being. It investigates what types of entities exist, how they are grouped into categories, and how they are related to one another on the most fundamental level (and whether there even is a fundamental level) Wikipedia.

[53] A taxonomy (or taxonomic classification) is a scheme of classification, especially a hierarchical classification, in which things are organized into groups or types Wikipedia.

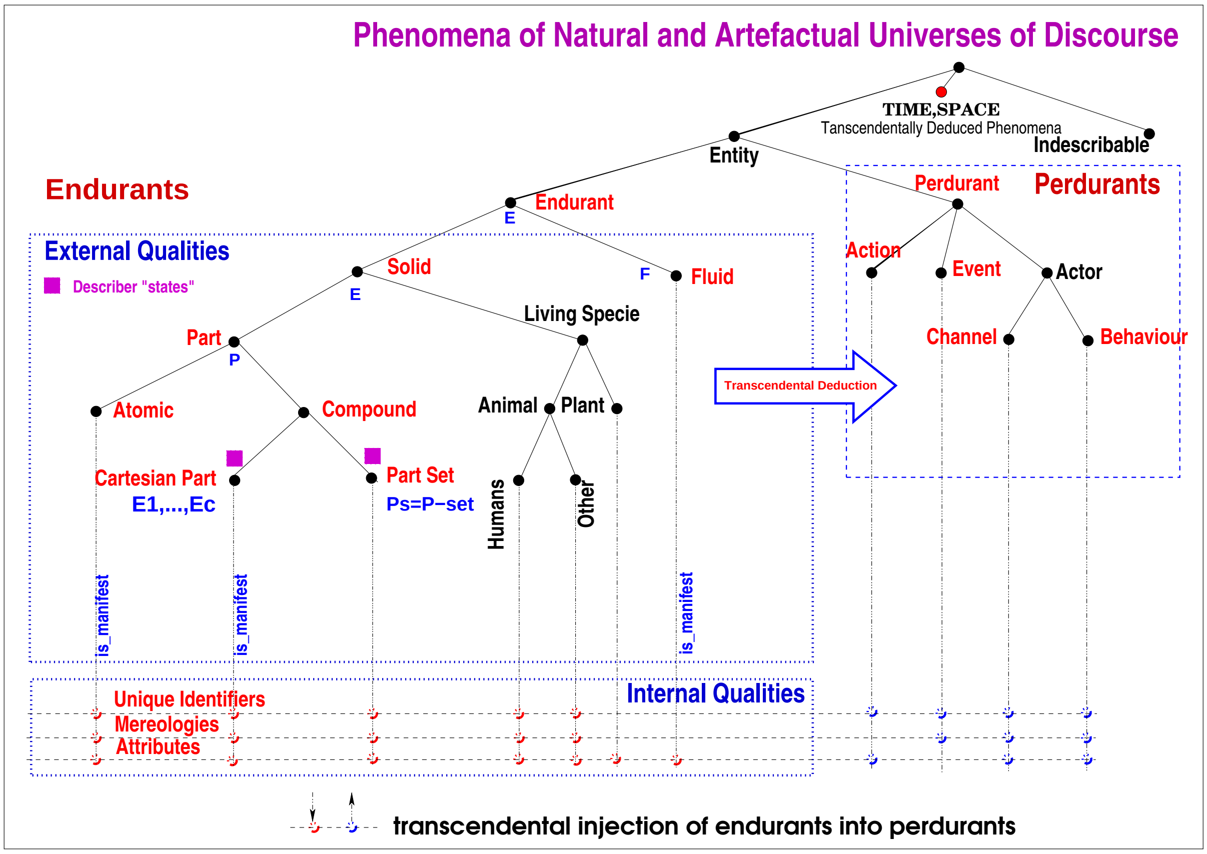


Figure B.1: A Domain Analysis & Description Ontology

The idea of Fig. B.1 is the following:

- It presents a recipe for how to **analyze** a domain.
- You, the *domain* **analyzer** *cum describer*, are *'confronted'*[54] with, or by a domain.
- You have Fig. B.1 in front of you, on a piece of paper, or in Your mind, or both.
- You are then asked, by the domain **analysis** & description method of this paper, to "start" at the uppermost •, just below and between the '**r**' and the first '**s**' in the main title, Phenomena of Natural and Artefactual Unive**rs**es of Discourse.
- The **analysis** & description ontology of Fig. B.1 then *directs* You to inquire as to whether the phenomenon – whichever You are "looking at/reading about/..." – is either *rationally describable*, i.e., is an *entity* (`is_entity`) or is *indescribable*.
- That is, You are, in general, "positioned" at a bullet, •, labeled α, "below" which there may be two alternative bullets, one, β, to the right and one to the left, γ.

[54] By 'confronted' we mean: You are reading about it, in papers, in books, in postings on the `Internet`, visiting it, talking with domain stakeholders: professional people working "in" the domain; You may, yourself, "be an entity" of that domain !

- It is Your decision whether the answer to the "query" that each such situation warrants, is yes, is_β, or no, is_γ.
- The characterizations of the concepts whose names, α, β, γ etc., are attached to the •s of Fig. B.1 are given in the following sections.
- Whether they are precise enough to guide You in Your obtaining reasonable answers, "yes" or "no", to the •ed queries is, of course, a problem. I hope they are.
- If Your answer is "yes", then Your **analysis** is to proceed "down the tree", usually indicated by "yes" or "no" answers.
- If one, or the other is a "leaf" of the ontology tree, then You have finished examining the phenomena You set out to **analyze**.
- If it is not a leaf, then further **analysis** is required.
- (We shall, in this report, leave out the analysis and hence description of *living species*.)
- If an **analysis** of a phenomenon has reached one of the (only) two ■'s, then the **analysis** at that • results in the domain describer **describing** some of the properties of that phenomenon.
- That **analysis** involves "setting aside", for subsequent **analysis & description**, one or more [thus **analysis** etc.-pending] phenomena (which are subsequently to be tackled from the "root" of the ontology).

We do not [need to] prescribe in which order You analyze & describe the phenomena that has been "set aside".

• • •

In Fig. B.1 on the previous page You will have noticed the positioning of the concepts of $\mathbb{TIME}$ and $\mathbb{SPACE}$ "right under" the *Phenomena* bullet •. These two concepts are neither endurants not perdurants. And they are not attributes of either. They can, however, as shown by Sørlander [65], be transcendentally deduced by rational reasoning.

B.4.2 Discussion of The Chosen Ontology

We shall in the following motivate the choice of the *ontological classification* reflected in Fig B.1 on the preceding page. We shall argue that this classification is not "an accidental choice". In fact, we shall try justify the classification with reference to the philosophy of Kai Sørlander [60, 63, 64, 65][55]. Kai Sørlander's aim in these books is to examine *that which is absolutely necessary, inevitable, in any description of the world.* In [20, *Chapter 2*] we present a summary of Sørlander's philosophy. In paragraphs, in the rest of this paper, marked ONTOLOGICAL CHOICE, we shall relate Sørlander's philosophy's "inevitability" to the ontology for studying domains.

[55] The 2022 book, [64], is presently a latest in Kai Sørlander's work. It refines and further develops the theme of the earlier, 1994–2016 books. [65] is an English translation of [64]

B.5 The Name, Type and Value Concepts

Domain *modeling*, as well as *programming*, depends, in their *specification*, on *separation of concerns:* which kind of *values* are subjectable to which kinds of *operations*, etc., in order to achieve ease of *understanding* a model or a program, ease of *proving properties* of a model, or *correctness* of a program.

B.5.1 Names

We name things in order to refer to them in our speech, models and programs. Names of types and values in models and programs are usually not so-called "first-citizens", i.e., values that can be arguments in functions, etc. The "science of names" is interesting.[56] In `botanical-society.org.za/the-science-of-names-an-introduction-to-plant-taxonomy` the authors actually speak of a "science of names" in connection with plant taxonomy: the "art" of choosing such names that reflect some possible classification of what they name.

B.5.2 Types

The type concept is crucial to programming and modeling.

Characterization: 4 *Type*: A *type* is a class, i.e., a further undefined set, of values ("of the same kind") ■

We name types.

Example: 2 *Type Names*: Some examples of type names are:

- RT – the class of all road transport instances: the *Metropolitan London Road Transport*, the *US Federal Freeway System*, etc.
- RN – the class of all road net instances (within a road transport).
- SA – the class of all automobiles (within a road transport) ■

You, the domain describer, choose type names. Choosing type names is a "serious affair". It must be done carefully. You can choose short (as above) or long names: Road_Transport, Road_Net, etc. We prefer short, but not cryptic names, like X, Y, Z, ... Names that are easy to *memorize*, i.e., *mnemonics*.

B.5.3 Values

Values are what programming and modeling, in a sense, is all about". In programming, values are the *data* "upon" which the program code specifies computations. In modeling values are, for example, what we observe: the entities in front of our eyes.

[56] The study of names is called *onomastics* or *onomatology*. *Onomastics* covers the naming of all things, including place names (toponyms) and personal names (*anthroponyms*).

B.6 **Phenomena and Entities**

Characterization: 5 *Phenomena*: By a *phenomenon* we shall understand a fact that is observed to exist or happen ■

Some phenomena are rationally describable – to some degree[57] – others are not.

Characterization: 6 *Entities*: By an entity By an *entity* we shall understand a more-or-less rationally describable phenomenon ■

Cue: 1 *is_entity*: We introduce the informal presentation language predicate is_entity. It holds for phenomena ϕ if ϕ is describable ■

A *prompt*[58] is an informal "advice" to the domain analyzer to "perform" a mental inquiry wrt. the real-life domain being studied.

Example: 3 *Phenomena and Entities*: Some, but not necessarily all aspects of a river can be rationally described, hence can be still be considered entities. Similarly, many aspects of a road net can be rationally described, hence will be considered entities ■

If You are not happy with this 'characterization', then substitute "rationally describable" with: *describable in terms of the endurants and perdurants brought forward in this paper: their external and internal qualities, unique identifiers, mereologies amd attributes, channels and behaviours !*

Ontological Choice: 1 *Phenomena*: We choose to "initialize" our ontological "search" to a question of whether a phenomenon is rationally describable – based on the tenet of Kai Sørlander's philosophy, namely that "whatever" we postulate is either *true* or *false* and that *a principle of contradiction* holds: *whatever we so express can not both hold and not hold* ■

Kai Sørlander then develops his inquiry – *as to what is absolutely necessary in any description of the world* – into the rationality of such descriptions necessarily be based on time and space and, from there, by a series of transcendental deductions, into a base in *Newton*'s physics. We shall, in a sense, stop there. That is, in the domain concept, such as we have delineated it, we shall not need to go into *Einsteinian* physics.

B.7 **Endurants and Perdurants, II**

We repeat our characterizations of endurants and perdurants.

[57] That is: It is up to the domain analyzer cum describer to decide as to how many rationally describable phenomena to select for analysis & description. Also in this sense one practices abstraction by "abstracting away" [the analysis & description of] phenomena that are irrelevant for the "current" (!) domain description.

[58] French: *mot-clé*, German: *stichwort*, Spanish: *palabra clave*

B.7.1 Endurants

We repeat characterization 2 on page 204.

Characterization: 7 *Endurant*: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster] ■

Example: 4 *Endurants*: Examples of endurants are: a street segment [link], a street intersection [hub], an automobile ■

Cue: 2 *is_endurant*: We introduce the informal presentation language predicate `is_endurant` to hold for entity `e` if `is_endurant(e)` holds ■

B.7.2 Perdurants

We repeat characterization 3 on page 204.

Characterization: 8 *Perdurant*: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster] ■

Example: 5 *Perdurant*: A moving automobile is an example of a perdurant ■

Cue: 3 *is_perdurant*: We introduce the informal presentation language predicate `is_perdurant` to hold for entity `e` if `is_perdurant(e)` holds ■

B.7.3 Ontological Choice

The **ontological choice** of entities being "viewed" as either endurants or perdurants is motivated as follows: The concept of endurants can be justified in terms of Newton's physics without going into kinematics, i.e., without including time considerations. The concept of perdurants can then, on one hand, be justified in terms of Newton's physics now taking time into consideration, hence kinematics, and from there causality, etc.; and, on the other hand, and as we shall see, by transcendentally deducing perdurants from solid endurants ■

B.8 External and Internal Endurant Qualities

The main contribution of this section is that of a calculus of domain analysis and description prompts. Two facets are being presented. Aspects of a domain science: of how we suggest domains can, and should, be viewed – ontologically. And aspects of a domain engineering: of how we suggest domains can, and should, be analyzed and described.

We begin by characterizing the two concepts: external and internal qualities.

Characterization: 9 *External Qualities*: External qualities of endurants of a manifest domain are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form.

Characterization: 10 *Internal Qualities*: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about ■

Perhaps we should instead label these two qualities tangible and intangible qualities.

Ontological Choice: 2 *Rationality*: The rational, analytic philosophy issues of the inevitability of these qualities is this: (i) can they be justified as inevitable, and (ii) can they be suitably "separated", i.e., both disjoint and exhaustive ? Or are they merely of empirical nature ? The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not 'either or'] ■

B.8.1 External Qualities – Tangibles

Example: 6 *External Qualities*: An example of external qualities of a domains is: the Cartesian[59] of sets of solid atomic street intersections, and of sets of solid atomic street segments, and of sets of solid automobiles of a road transport system where *Cartesian, sets, atomicity*, and *solidity* reflect external qualities ■

B.8.1.1 The Universe of Discourse

The most immediate external quality of a domain is the "entire" domain – "itself" ! So any domain analysis starts by identifying that "entire" domain ! By giving it a name, say UoD, for *universe of discourse*, Then describing it, in *narrative* form, that is, in natural language containing terms of professional/technical nature, the domain. And, finally, *formalizing* just the name: giving the name "status" of being a type name, that is, of the type of a class of domains whose further properties will be described subsequently.

Axiom: 1 *The Universe of Discourse*:

Narration:

The name, and hence the type, of the domain is UoD
The UoD domain can be briefly characterized by ...

Formalization:

type UoD ■

[59] Cartesian after the French philosopher, mathematician, scientist René Descartes (1596–1650)

B.8.1.2 **Solid and Fluid Endurants**

Given then that there are endurants we now postulate that they are either [mutually exclusive] *solid* (i.e., discrete) or *fluid.*

Ontological Choice: 3 *Solids vs. Fluids*: Here we [seem to] make a practical choice, not one based on a philosophical argument, one of logical necessity, but one based on empirical evidence. It is possible for endurants to either be solid or fluid; and here we shall not consider the case where solid [fluid] endurants, due to being heated [cooled], enters a fluid state [or vice versa] ■

B.8.1.2.1 Solid cum Discrete Endurants.

Characterization: 11 *Discrete cum Solid Endurants*: By a *solid* cum *discrete* endurant we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing, have body (or magnitude) of three-dimensions: length (or height), breadth and depth [53, *OED, Vol. II, pg. 2046*] ■

Example: 7 *Solid Endurants*: Pipeline system examples of solid endurants are *wells, pipes, valves, pumps, forks, joins* and *sinks* of pipelines. (These units may, however, and usually will, contain fluids, e.g., oil, gas or water.) ■

Cue: 4 `is_solid`: We introduce the informal presentation language predicate `is_solid` to hold for endurant `e` if `is_solid(e)` holds ■

B.8.1.2.2 Fluids.

Characterization: 12 *Fluid Endurants*: By a *fluid endurant* we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [53, *OED, Vol. I, pg. 774*] ■

Example: 8 *Fluid Endurants*: Examples of fluid endurants are: *water, oil, gas, compressed air, smoke* ■

Fluids are otherwise liquid, or gaseous, or plasmatic, or granular[60], or plant products, i.e., chopped sugar cane, threshed, or otherwise[61], et cetera. Fluid endurants will be analyzed and described in relation to solid endurants, viz. their "containers".

Cue: 5 `is_fluid`: We introduce the informal presentation language predicate `is_fluid` to hold for endurant `e` if `is_fluid(e)` holds ■

[60] This is a purely pragmatic decision. "Of course" sand, gravel, soil, etc., are not fluids, but for our modeling purposes it is convenient to "compartmentalise" them as fluids !

[61] See footnote 60.

B.8.1.3 Parts and Living Species Endurants

Given then that there are solid endurants we now postulate that they are either [mutually exclusive] *parts* or *living species*.

Ontological Choice: 4 *Parts and Living Species*: With Sørlander, [65, *Sect. 5.7.1, pages 71–72*] we reason that one can distinguish between parts and living species ■

B.8.1.3.1 Parts

Characterization: 13 *Parts*: The non-living solid species are what we shall call parts ■

Parts are the "work-horses" of man-made domains. That is, we shall mostly be concerned with the analysis and description of endurants into parts.

Example: 9 *Parts*: Example 7, of solids, is an example of parts ■

Cue: 6 *is_part*: We introduce the informal presentation language predicate `is_part` to hold for solid endurants e if `is_part(e)` holds ■

We distinguish between atomic and compound parts.

Ontological Choice: 5 *Atomic and Compound Parts*: It is an empirical fact that parts can be composed from parts. That possibility exists. Hence we can [philosophy-wise] reason likewise ■

B.8.1.3.1.1 Atomic Parts.

Characterization: 14 *Atomic Part*: By an *atomic part* we shall understand a part which the domain analyzer considers to be indivisible in the sense of not meaningfully consist of sub-parts ■

Example: 10 *Atomic Parts*: Examples of atomic parts are: hubs, H, i.e., street intersections; links, L, i.e., the stretches of roads between two neighbouring hubs; and automobiles, A:

type H, L, A ■

Cue: 7 *is_atomic*: We introduce the informal presentation language predicate `is_atomic` to hold for parts p if `is_atomic(p)` holds ■

B.8.1.3.1.2 Compound Parts.

Characterization: 15 *Compound Part*: Compound parts are those which are observed to [potentially] consist of several parts ■

Example: 11 *Compound Parts*: An example of a compound parts is: a road net consisting of a set of hubs, i.e., street intersections or "end-of-streets", and a set of links, i.e., street segments (with no contained hubs), is a Cartesian compound; and the sets of hubs and the sets of links are part set compounds ■

Cue: 8 *is_compound*: We introduce the informal presentation language predicate `is_compound` to hold for parts p if `is_compound(p)` holds ■

We, pragmatically, distinguish between Cartesian product- and set-oriented parts.

Ontological Choice: 6 *Cartesians*: The Cartesian versus set parts is an empirical choice. It is not justified in terms of philosophy, but in terms of mathematics – of mathematical expediency ! ■

B.8.1.3.1.3 Cartesians. Cartesians are product-like types – and are named after the French philosopher, scientist and mathematician René Descartes (1596–1640) `Wikipedia`.

Characterization: 16 *Cartesians*: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids) ■

Example: 12 *Cartesians: Road Transport*: A road transport, rt:RT, is observed to consist of an aggregate of a road net, rn:RN, and a set of automobiles, SA, where the road net is observed, i.e., abstracted, as a Cartesian of a set of hubs, ah:AH, i.e., street intersections (or specifically designated points segmenting an otherwise "straight" street into two such), and a set of links, al:AL, i.e., street segments between two "neighbouring" hubs.

```
type
    RT, RN, SA, AH = H-set, AL = L-set
value
    obs_RN: RT → RN, obs_SA: RT → SA,, obs_AH: RN → AH, obs_AL: RN → AL  ■
```

Cue: 9 *is_Cartesian*: We introduce the informal presentation language predicate `is_Cartesian` to hold for compound parts p if `is_Cartesian(p)` holds ■

Once a part, say p:P, has been analyzed into a Cartesian, we inquire as to the type names of the endurants[62] of which it consists. The inquiry: `record_Cartesian_part_type_names(p:P)`, we decide, then yields the type of the constituent endurants.

[62] We emphasize that the observed elements of a Cartesian part may be both solids, at least one, and fluids.

Cue: 10 *record-Cartesian-part-type-names*:

value
  record_Cartesian_part_type_names: P → $\mathbb{T}$-**set**
  record_Cartesian_part_type_names(p) **as** {ηE1,ηE2,...,ηEn} ■

Here $\mathbb{T}$ is the **name** of the type of all type names, and ηEi is the **name** of type Ei.

Please note the novel introduction of type names as values. Where a type identifier, say T, stands for, denotes, a class of values of that type, ηT denotes the name of type T.

Please also note that record_Cartesian_part_type_names is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" it to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example: 13 *Cartesian Parts*: The Cartesian parts of a road transport, rt:RT, is thus observed to consists of

- an aggregate of a road net, rn:RN, and
- an aggregate set of automobiles, sa:SA:

that is:

- record_Cartesian_part_type_names(rt:RT) = {ηRN,ηSA}

where the type name ηRT was – and the type names ηRN and ηSA are – coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer ■

B.8.1.3.1.4 Part Sets.

Characterization: 17 *Part Sets*: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts ■

Cue: 11 *is_part_set* : We introduce the informal presentation language predicate is_part_set to hold for compound parts e if is_part_set(e) holds■

Once a part, say e:E, has been analyzed into a part set we inquire as to the set of parts and their type of which it consists. The inquiry: record_part_set_part_type_names, we decide, then yields the (single) type of the constituent parts.

Cue: 12 *record-part-set-part-type-names*:

value
  record_part_set_part_type_names: E → $\mathbb{T}$Ps×$\mathbb{T}$P
  record_part_set_part_type_names(e:E) **as** (η Ps,η P) ■

Here the name of the value, e, and the type names η Ps and η P are coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer ■

Please also note that `record_part_set_part_type_names` is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" in to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example: 14 *Part Sets: Road Transport*: The road transport contains a set of automobiles. The part set type name has been chosen to be SA. It is then determined (i.e., analyzed) that SA is a set of Automobile of type A

- `record_part_set_part_type_names(sa:SA)` = (η As,η A) ■

B.8.1.3.1.5 Compound Observers. Once the domain analyzer cum describer has decided upon the names of atomic and compound parts, **obs**_erver functions can be applied to Cartesian and part set, e:E, parts:

Schema 1 *Describe Cartesians and Part Set Parts*

```
value
    let {η P1,η P2,...,η Pn} = record_Cartesian_part_type_names(e:E) in
    "type
            P1, P2, ..., Pn;
        value
            obs_P1: E→P1, obs_P2: E→P2,...n obs_Pn: E→Pn "
    end
```

respectively:

```
    let (η Ps,η P) = record_part_set_part_type_names(e:E) in
    "type
            P, Ps = P-set,
        value
            obs_Ps: E→Ps "
    end                                   ■
```

The "..." texts are the RSL texts "generated", i.e., written down, by the domain describer. They are *domain model specification units*. The "surrounding" RSL-like texts are not written down as phrases, elements, of the domain description. They are elements of the domain describers' "notice board", and, as such, elements of the development of domain models. We have introduced a core domain modeling tool the **obs**_... observer function, one to be "applied" mentally by the domain describer, and one that appears in ($\texttt{RSL}^{+}$`-Text`) domain descriptions The **obs**_... observer function is "applied" by the domain describer, it is not a computable function.

Please also note that `Describe Cartesians and Part Set Parts` schema, 1, is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" in to an observed endurant and notes down, but now in a final form, elements, that is *domain description units*.

• • •

A major step of the development of domain models has now been presented: that of the analysis & description of the external qualities of domains.

Schema 1 on the previous page is the first manifestation of the domain analysis & description method leading to actual domain description elements.

From unveiling a *science of domains* we have "arrived" at an *engineering of domain descriptions.*

B.8.1.4 States

Characterization: 18 *States*: By a *state* we shall mean any subset of the parts of a domain ■

Example: 15 *Road Transport State*:

```
variable
    hs:AH := obs_AH(obs_RN(rt)),
    ls:AL := obs_AL(obs_RN(rt)),
    as:SA := obs_SA(rt),
    σ:(H|L|A)-set := hs∪ls∪as ■
```

We have chosen to model domain states as **variable**s rather than as **value**s. The reason for this is that the values of monitorable, including biddable part attributes[63] can change, and that domains are often extended and "shrunk" by the addition, respectively removal of parts:

Example: 16 *Road Transport Development*: adding or removing hubs, links and automobiles ■

We omit coverage of the aspect of bidding changes to monitorable part attributes.

B.8.1.5 Validity of Endurant Observations

We remind the reader that the **obs**_erver functions, as all later such functions: **uid**_-, **mereo**_- and **attr**_-functions, are applied by humans and that the outcome of these "applications" is the result of human choices, and possibly biased by inexperience, taste, preference, bias, etc. How do we know whether a domain analyzer & describer's description of domain parts is valid ? Whether relevantly identified parts are modeled reasonably wrt. being atomic, Cartesians or part sets Whether all relevant endurants have been identified ? Etc. The short answer is: we never know. Our models are conjectures and may be refuted [57]. A social process of peer reviews, by domain stakeholders and other domain modelers is needed – as may a process of verifying[64] properties of the domain description held up against claimed properties of the (real) domain.

B.8.1.6 Summary of Endurant Analysis Predicates

Characterizations 6–17 imply the following analysis predicates (Char.: δ, Page π):

[63] The concepts of monitorable, including biddable part attributes is treated in Sect. B.8.2.3.2.

[64] testing, model checking and theorem proving

- is_entity, $\delta 6\,\pi\,208$
- is_endurant, $\delta 7\,\pi\,209$
- is_perdurant, $\delta 8\,\pi\,209$
- is_solid, $\delta 11\,\pi\,211$
- is_fluid, $\delta 12\,\pi\,211$
- is_part, $\delta 13\,\pi\,212$
- is_atomic, $\delta 14\,\pi\,212$
- is_compound, $\delta 15\,\pi\,213$
- is_Cartesian, $\delta 16\,\pi\,213$
- is_part_set, $\delta 17\,\pi\,214$

We remind the reader that the above predicates represent "formulas" in the presentation, **not** the description, language. They are not RSL$^+$-Text clauses. They are in the mind of the domain analyzers cum describers. They are "executed" by such persons. Their result, whether **true**, **false** or **chaos**[65], are noted by these persons and determine their next step of domain analysis.

B.8.1.7 "Trees are Not Recursive"

A 'fact', that seems to surprise many, is that parts are not "recursive". Yes, in all our domain modeling experiments, [25], we have not come across the need for recursively observing compound parts. Trees, for example, are not recursive in this sense. Trees have roots. Subtrees not. Banyan trees[66] have several "intertwined trees". But it would be 'twisting' the modeling to try fit a description of such trees to a 'recursion wim' ! Instead, trees are defined as nets, such as are road nets, where these nets then satisfy certain constraints [25, *Chapter B*] – usually modeled by a mereology, see Sect. B.8.2.2 on page 219.

B.8.2 Internal Qualities – Intangibles

The previous section has unveiled an ontology of the external qualities of endurants. The unveiling consisted of two elements: a set of analysis predicates, predicates 6–17, and analysis functions, schemas 10–12, and a pair of description functions, schema 1 on page 215.

The application of description functions result in RSL-Text.

That text conveys certain properties of domains: that they consists of such-and-such endurants, notably parts, and that these endurants "derive" from other endurants. But the RSL-Text description texts do not *"give flesh & blood"* to these endurants. Questions like: *'what are their spatial extents ?'*, *'how much do the weigh ?'*, *'what colour do they have ?'*, et cetera, are left unanswered. In the present section we shall address such issues. We call them *internal qualities*.

Characterization: 19 *Internal Qualities*: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about ■

Example: 17 *Internal qualities*: Examples of internal qualities are the *unique identity* of a part, the *mereological relation* of parts to other parts, and the endurant *attributes* such as temperature, length, colour, etc. ■

[65] The outcome of applying an analysis predicate of the prescribed kind may be **chaos** if the prerequisites for its application does not hold.

[66] https://www.britannica.com/plant/banyan

This section therefore introduces a number of domain description tools:

- **uid_**: the unique identifier observer of parts;
- **mereo_**: the mereology observer of parts;
- **attr_**: (zero,) one or more attribute observers of endurants; and
- **attributes_**: the attribute query of endurants.

B.8.2.1 Unique Identity

Ontological Choice: 7 *Unique Identity*: We postulate that separately discernible parts have unique identify. The issue, really, is a philosophical one. We refer to [20, *Sects. 2.2.2.3–2.2.2.4, pages 14–15*] for a discussion of the existence and uniqueness of entities ■

Characterization: 20 *Unique Identity*: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids[67] ■

The unique identity of a part p of type P is obtained by the postulated observer **uid_**P:

Schema 2 *Describe-Unique-Identity-Part-Observer*

```
“type
     P,PI
 value
     uid_P: P → PI” ■
```

Here PI is the type of the unique identifiers of parts of type P.

Example: 18 *Unique Road Transport Identifiers*: The unique identifierss of a road transport, rt:RT, consists of the unique identifiers of the

- road transport – rti:RTI,
- (Cartesian) road net – rni:RNI,
- (set of) automobiles – sa:SAI,
- automobile, ai:AI,
- (set of) hubs, hai:AHI,
- (set of) links, lai:LAI,
- hub, hi:HI, and
- link, li:LI,

where the type names are all coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer – though, as You can see, these names were here formed by “suffixing” Is to relevant part names ■

We have thus introduced a core domain modeling tool the **uid_**... observer function, one to be “applied” mentally by the domain describer, and one that appears in (RSL$^+$-Text) domain descriptions The **uid_**... observer function is “applied” by the domain describer, it is not a computable function.

[67] For pragmatic reasons we do not have to speculate as to whether “bodies” of fluids can be ascribed unique identity. The pragmatics is that we, in our extensive modeling experiments have not found a need for such ascription !

B.8.2.1.1 Uniqueness of Parts No two parts have the same unique identifier.

Example: 19 *Road Transport Uniqueness*:

variable
  hs_{uids}:HI-**set** := { **uid**_H(h) | h:H•u ∈ σ }
  ls_{uids}:LI-**set** := { **uid**_L(l) | l:L•u ∈ σ }
  as_{uids}:AI-**set** := { **uid**_A(a) | a:A•u ∈ σ }
  σ_{uids}:(HI|LI|AI)-**set** := { **uid**_(H|L|A)(u) | u:(H|L|A)•u ∈ σ }
axiom
  □ **card** σ = **card** σ_{uids} ■ For σ see Sect. B.8.1.4 on page 216.

We have chosen, for the same reason as given in Sect. B.8.1.4, to model a unique identifier state. The □ [*always*] prefix in the **axiom** then expresses that changes of parts or addition of parts to and deletions of parts from the domain shall maintain their uniqueness over time (i.e., always).

B.8.2.2 Mereology

The concept of mereology is due to the Polish mathematician, logician and philosopher Stanisław Leśniewski (1886–1939) [68, 13].

Characterization: 21 *Mereology*: Mereology is a theory of [endurant] part-hood relations: of the relations of an [endurant] parts to a whole and the relations of [endurant] parts to [endurant] parts within that whole ■

Ontological Choice: 8 *Mereology*: Stanisław Leśniewski was not satisfied with Bertrand Russell's "repair" of Gottlob Frege's axiom systems for set theory. Instead he put forward his axiom system for, as he called it, mereology. Both as a mathematical theory and as a philosophical reasoning ■

Example: 20 *Mereology*: Examples of mereologies are that a link is topologically *connected* to exactly one or, usually, two specific hubs, that hubs are *connected* to zero, one or more specific links, and that links and hubs are *open* to the traffic of specific subsets of automobiles ■

Mereologies can be expressed in terms of unique identifiers.

Example: 21 *Mereology Representation*: For our 'running road transport example' the mereologies of links, hubs and automobiles can thus be expressed as follows:

- **mereo**_L(l) = {hi′,hi″} where hi,hi′,hi″ are the unique identifiers of the hubs that the link connects, i.e., are in hs_{uids};
- **mereo**_H(h) = {li$_1$,li$_2$,...,li$_n$} where li$_1$,li$_2$,...,li$_n$ are the unique identifiers of the links that are imminent upon (i.e., emanates from) the hub, i.e., are in ls_{uids}; and

- **mereo**_A(a) = $\{ri_1, ri_2, \ldots, ri_m\}$ where $ri_1, ri_2, \ldots, ri_m$ are unique identifiers of the road (hub and link) elements that make up the road net, i.e., are in $hs_{uids} \cup ls_{uids}$ ■

Once the unique identifiers of all parts of a domain has been described we can analyses and describe their mereologies. The inquiry: **mereo_P**(p) yields a mereology type (name), say PMer, and its description[68]:

Schema 3 *Describe-Mereology*

```
"type
    PMer = 𝓜(PI1,PI2,...,PIm)
value
    mereo_P: P → PMer
axiom
    𝒜(pm:PMer)" ■
```

where $\mathscr{M}$(PI1,PI2,...,PIm) is a type expression over unique identifier types of the domain; **mereo**_P is the mereology observer function for parts p:P; and $\mathscr{A}$(pm:PMer) is an axiom that secures that the unique identifiers of any part are indeed of parts of the domain.

B.8.2.3 Attributes

Attributes are what finally gives "life" to endurants: The external qualities "only" named and gave structure to their atomic or compound types. The internal qualities of uniqueness and mereology are intangible quantities. The internal quality of attributes gives "flesh & blood" to endurants: they let us express endurant properties that we can more easily, i.e., concretely, relate to.

B.8.2.3.1 General

Characterization: 22 *Attributes*: Attributes are properties of endurants that can be measured either physically (by means of length (ruler) and spatial quantity measuring equipment, electronically, chemically, or otherwise) or can be objectively spoken about ■

Ontological Choice: 9 *Attributes*: First some empirical observation: in reasoning about "the world around us" we express its properties in terms of predicates. These predicates, for example: *"that building's wall is red"*, *building* refers to an endurant part whereas *wall* and *red* refers to attributes. Now the "rub": endurant attributes is what give *"flesh & blood"* to domains ■

Attributes are of types and, accordingly have values.

We postulate an informal domain analysis function, `record_attribute_type_names`: The domain analyzer, in observing a part, *p*:*P*, analyzes it into the set of attribute names of parts *p*:*P*

[68] Cf. Sect. B.8.1.3.1.5

Schema 4 *record-attribute-type-names*

```
value
    record_attribute_type_names: P → $\eta\mathbb{T}$-set
    record_attribute_type_names(p:P) as $\eta$T-set ■
```

Example: 22 *Road Net Attributes, I*: Examples of attributes are: hubs have states, hσ:HΣ: the set of pairs of link identifiers, ($f$li,$t$li), of the links $f$rom and to which automobiles may enter, respectively leave the hub; and hubs have state spaces, hω:HΩ: the set of hub states "signaling" which states are open/closed, i.e., **green**/**red**; links that have lengths, LEN; and automobiles have road net positions, APos, either *at a hub*, atH, or *on a link*, onL, some fraction, f:**Real**, down a link, identified by li, from a hub, identified by fhi, towards a hub, identified by thi. Hubs and links have *histories:* time-stamped, chronologically ordered sequences of automobiles entering and leaving links and hubs, with automobile histories similarly recording hubs and links entered and left.

```
type
    HΣ = (LI×LI)-set
    HΩ = HΣ-set
    LEN = Nat m
    APos = atH | onL
    atH :: HI
    onL :: LI × (fhi:HI × f:Real × thi:HI)
    HHis,LHis = ($\mathbb{TIME}$×AI)$^*$
    AHis =  ($\mathbb{TIME}$×(HI|LI))$^*$
value
    attr_HΣ: H → HΣ
    attr_HΩ: H → HΩ
    attr_LEN: L → LEN
    attr_APos: A → APos
    attr_HHis: H → HHis
    attr_LHis: L → LHis
    attr_AHis: A → AHis
axiom
    ∀ (li,(fhi,f,thi)):onL • 0<f<1
        ∧li∈$ls_{uids}$∧{fhi,thi}⊆$hs_{uids}$∧... ■
```

Schema 5 *Describe-endurant-attributes(e:E)*

```
    let {ηA1,ηA2,...,ηAn}  = record_attribute_type_names(e:E) in
    " type
          A1, A2, ..., An
      value
          attr__A1: E → A1, attr__A2: E → A2, ..., attr__An: E → An
      axiom
          ∀ a1:A1, a2:A2, ..., an:An: $\mathscr{A}$(a1,a2,...,an) "
    end ■
```

B.8.2.3.2 Michael A. Jackson's Attribute Categories Michael A. Jackson [50] has suggested a hierarchy of attribute categories:from *static* (`is_static`[69]) to *dynamic* (`is_dynamic`[70]) values – and within the dynamic value category: *inert* values (`is_inert`[71]), *reactive* values (`is_reactive`[72]), *active* values (`is_active`[73]) – and within the dynamic active value category: *autonomous* values (`is_autonomous`[74]), *biddable* values (`is_biddable`[75]), and *programmable* values (`is_programmable`[76]). We postulate informal domain analysis predicates, "performed" by the domain analyzer:

value

is_static,is_autonomous,is_biddable,is_programmable [etc.]: η T→**Bool**

We refer to [50] and [20] [*Chapter 5, Sect. 5.4.2.3*] for details. We suggest a minor revision of Michael A. Jackson's attribute categorization, see left side of Fig. B.2. We single out the inert from the ontology of Fig. B.2, left side. Inert attributes seem to be "set externally" to the endurant. So we now distinguish between `is_external` and `is_internal` dynamic attributes. We summarize Jackson's attribute and our revised categorization in Fig. B.2.

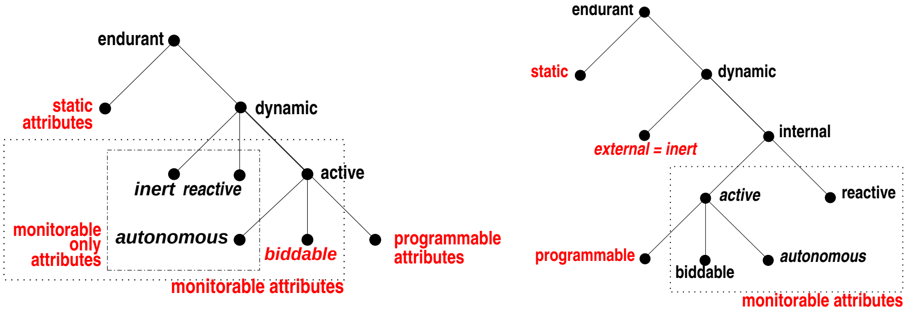


Figure B.2: Michael Jackson's [Revised] Attribute Categories

This distinction has [pragmatic] consequences for how we treat arguments of the behaviours of parts, cf. Sect. B.9.5.1 (page 227).

Example: 23 *Road Net Attributes, II*: The link length and hub state space attributes are static, hub states and automobile positions programmable. Automobile speed and acceleration attributes, which we do not model, are monitorable ■

The attributes categorization determines, in the next major section on perdurants, the treatment of hub, link and automobile behaviours.

[69] `static`: values are constants, cannot change

[70] `dynamic`: values are variable, can change

[71] `inert`: values can only change as the result of external stimuli where these stimuli prescribe new values

[72] `reactive`: values, if they vary, change in response to external stimuli, where these stimuli either come from outside the domain of interest or from other endurants.

[73] `active`: values can change (also) on their own volition

[74] `autonomous`: values change only "on their own volition"; the values of an autonomous attributes are a "law onto themselves and their surroundings".

[75] `biddable`: values are prescribed but may fail to be observed as such

[76] `programmable`: values can be prescribed

B.8.2.3.3 Analytic Attribute Extraction Functions: For later purpose we need characterize three specific attribute category extraction functions: `static_attributes`, `monitorable_attributes`, and `programmable_attributes`:

```
value
    p:P
    tns = record_attribute_type_names(p)

    static_attributes: ηT-set → ηT-set
    static_attributes(tns) ≡ { ηtn | ηtn:ηT • ηtn ∈ tns ∧ is_static(tn) }

    inert_attributes: ηT-set → ηT-set
    inert_attributes(tns) ≡ { ηtn | ηtn:ηT • ηtn ∈ tns ∧ is_inert(tn) }

    monitorable_attributes ηT-set → ηT-set
    monitorable_attributes(tns) ≡ { ηtn | ηtn:ηT • ηtn ∈ tns ∧ is_monitorable(tn) }

    programmable_attributes ηT-set → ηT-set
    programmable_attributes(tns) ≡ { ηtn | ηtn:ηT • ηtn ∈ tns ∧ is_programmable(tn) }

    is_monitorable: T → Bool
    is_monitorable(t) ≡ ∼is_static(t) ∧ ∼is_inert(t) ∧ ∼is_programmable(t)
```

Please be reminded that these functions are informal. They are part of the presentation language. Do not be confused by their RSL-Text-like appearance.

B.8.3 Intentional Pull

Ontological Choice: 10 *Intentional Pull*: In [63, *pages 167–168*] Sørlander argues wrt. *"how can entities be the source of forces ?"* and thus reasons for *gravitational pull*. That same kind of reasoning, with proper substitution of terms, leads us to the concept of *intentional pull* ■

Two or more parts of different sorts, but with overlapping sets of intents[77] may excert an intentional "pull" on one another. This *intentional "pull"* may take many forms. Let $p_x : X$ and $p_y : Y$ be two parts of *different sorts* (X,Y), and with *common intent*, ι. *Manifestations* of these, their common intent must somehow be *subject to constraints*, and these must be *expressed predicatively*.

When a compound artifact models "itself" as put together with a number of other endurants then it does have an intentionality and the components' individual intentionalities does, i.e., shall relate to that.

Example: 24 *Road Transport Intentionality*: *Automobiles* include the *intent:* `transport`, and so do *hubs* and *links*. *Manifestations* of `transport` are reflected in *hubs, links* and *automobiles* having the *history* attribute. The *intentional "pull"* of these manifestations is

[77] Intent: purpose; God-given or human-imposed !

this: For every automobile, if it records being in some hub or on some link at time τ, then the designated hub, respectively link, records exactly that automobile; and vice versa: for every hub [link], if it records the visit of some automobile at time τ, then the designated automobile records exactly that hub [link]. We leave the formalization of the above to the reader ■

Example: 25 *Double-entry Bookkeeping*: Another example of intentional "pull" is that of double-entry bookkeeping. Here the income/expense ledger must balance the actives/passives ledger ■

Example: 26 *The Henry George Theorem.*: The Henry George theorem states that under certain conditions, aggregate spending by government on public goods will increase aggregate rent based on land value (land rent) more than that amount, with the benefit of the last marginal investment equaling its cost ■[78,79]

B.8.4 Summary of Endurants

We have completed our treatment of endurants. That treatment was based on an ontology for the observable phenomena of domains – such as we have delineated the concept of domains. The treatment was crucially based on an ontology for the structure of domain phenomena, and, in a sense, "alternated" between analysis predicates, analysis functions, and description functions. We have carefully justified this ontology in **'Ontological Choice'** paragraphs

B.9 Perdurant Concepts

The main contribution of this section is that of *transcendentally deducing* perdurants from endurant parts, in particular *behaviours* "of" parts.

Major perdurants are those of actions, events and behaviours with behaviours generally being sets of sequences of **actions, events** and **behaviours.**

B.9.1 "Morphing" Parts into Behaviours

As already indicated we shall transcendentally deduce (perdurant) behaviours from those (endurant) parts which we, as domain analyzers cum describers, have endowed with all three kinds of internal qualities: unique identifiers, mereologies and attributes. We shall use the CSP [47] constructs of RSL-Text (derived from RSL [41]) to model concurrent behaviours.

[78] Stiglitz, Joseph (1977). "The Theory of Local Public Goods". In Feldstein, M.S.; Inman, R.P. (eds.). The Economics of Public Services. Palgrave Macmillan, London. pp. 274333. doi:10.1007/978-1-349-02917-4_12. ISBN 978-1-349-02919-8.

[79] Henry George (September 2, 1839 – October 29, 1897) was an American political economist and journalist. His writing was immensely popular in 19th-century America and sparked several reform movements of the Progressive Era. He inspired the economic philosophy known as Georgism, the belief that people should own the value they produce themselves, but that the economic value of land (including natural resources) should belong equally to all members of society. George famously argued that a single tax on land values would create a more productive and just society.

B.9.2 Transcendental Deduction

Transcends is the basic ground concept from the word's literal meaning of climbing or going beyond, albeit with varying connotations in its different historical and cultural stages.

Characterization: 23 *Transcendence*: By **transcendence** we shall understand the notion: **the a priori or intuitive basis of knowledge, independent of experience** ■

A priori knowledge or intuition is central: By *a priori* we mean that it not only precedes, but also determines rational thought.

Characterization: 24 *Transcendental Deduction*: By a **transcendental deduction** we shall understand the philosophical notion: **a transcendental "conversion" of one kind of knowledge into a seemingly different kind of knowledge** ■

Example: 27 *Transcendental Deductions – Informal Examples*: We give some intuitive examples of transcendental deductions. They are from the "domain" of programming languages. There is the syntax of a programming language, and there are the programs that supposedly adhere to this syntax. Given that, the following are now transcendental deductions.

The software tool, **a syntax checker**, that takes a program and checks whether it satisfies the syntax, including the statically decidable context conditions, i.e., the *statics semantics* – such a tool is one of several forms of transcendental deductions.

The software tools, **an automatic theorem prover** and **a model checker**, for example `SPIN` [48], that takes a program and some `theorem`, respectively a `Promela` statement, and proves, respectively checks, the program correct with respect the theorem, or the statement.

A **compiler** and an **interpreter** for any programming language.

Yes, indeed, any **abstract interpretation** [37] reflects a transcendental deduction: firstly, these examples show that there are many transcendental deductions; secondly, they show that there is no single-most preferred transcendental deduction ■

Ontological Choice: 11 *Transcendental Deduction of Behaviours from Parts*: So this, then, is, in a sense, our "final" ontological choice: that of transcendentally deducing behaviours from parts ■

B.9.3 Actors – A Synopsis

This section provides a summary overview.

Characterization: 25 *Actors*: An actor is anything that can initiate an **action**, **event** or **behaviour** ■

B.9.3.1 Action

Characterization: 26 *Actions*: An action is a function that can purposefully change a state ■

Example: 28 *Road Net Actions*: These are some road transport actions: an automobile leaving a hub, entering a link; leaving a link, entering a hubs; entering the road net; and leaving the road net ■

B.9.3.2 **Event**

Characterization: 27 *Events*: An event is a function that surreptitiously changes a state ■

Example: 29 *Road Net Events*: These are some road net events: The blocking of a link due to a mud slide; the failing of a hub traffic signal due to power outage; an automobile failing to drive; and the blocking of a link due to an automobile accident ■

We shall not formalize events.

B.9.3.3 **Behaviour**

Characterization: 28 *Behaviours*: Behaviours are sets of sequences of actions, events and behaviours ■

Concurrency is modeled by the *sets* of sequences. Synchronization and communication of behaviours are effected by CSP *output/inputs*: ch[{i,j}] !value/ch[{i,j}] ?.

Example: 30 *Road Net Traffic*: Road net traffic can be seen as a behaviour of all the behaviours of automobiles, where each automobile behaviour is seen as sequence of start, stop, turn right, turn left, etc., actions; of all the behaviours of links where each link behaviour is seen as a set of sequences (i.e., behaviours) of "following" the link entering, link leaving, and movement of automobiles on the link; of all the behaviours of hubs (etc.); of the behaviour of the aggregate of roads, viz. *The Department of Roads*, and of the behaviour of the aggregate of automobiles, viz, *The Department of Vehicles*.

B.9.4 **Channel**

Characterization: 29 *Channel*: A channel is anything that allows synchronization and communication of values between behaviours ■

Schema 6 *Channel*

We suggest the following schema for describing channels:

channel { ch[{ui,uj}] | ui,ij:UI • ... } : M

where ch is the describer-chosen name for an array of channels, ui,uj are channel array indices of the unique identifiers, UI, of the chosen domain ■

Example: 31 *Road Transport Interaction Channel*:

channel { ch[{ui,uj}] | {ui,ij}:(HI|LI|AI)**-set** • ui$\neq$uj$\wedge${ui,uj}$\subseteq\sigma_{uids}$ } : M

Channel array ch is indexed by a "pair" of distinct unique part identifiers of the domain. We shall later outline M, the type of the "messages" communicated between behaviours ■

B.9.5 Behaviours

We single out the perdurants of behaviours – as they relate directly to the parts of Sect. B.8. The treatment is "divided" into three sections.

B.9.5.1 Behaviour Signature

Schema 7 *Behaviour Signature*

By the *behaviour signature*, for a part p, we shall understand a pair: the name of the behaviour, B_p, and a function type expression as indicated:

value

B_p: $\mathsf{Uid}_p \to$[80] $\mathsf{Mereo}_p \to \mathsf{Sta\_Vals}_p \to \mathsf{Inert\_Vals}_p \to \mathsf{Mon\_Refs}_p \to \mathsf{Prgr\_Vals}_p \to$ { ch[{i,j}] | ... }

Unit

We explain:

- Uid_p is the type of unique identifiers of part p, **uid**_P(p) = Uid_p;
- Mereo_p is the type of the mereology of part p, **mereo**_P(p) = Mereo_p;
- $\mathsf{Sta\_Vals}_p$ is a Cartesian of the type of inert attributes of part p. Given `record_attribute_type_names(p)` `static_attributes(record_attribute_type_names(p))` yields $\mathsf{Sta\_Vals}_p$;
- $\mathsf{Inert\_Vals}_p$ is a Cartesian of the type of static attributes of part p. Given `record_attribute_type_names(p)` `inert_attributes(record_attribute_type_names(p))` yields $\mathsf{Inert\_Vals}_p$;
- $\mathsf{Mon\_Refs}_p$ is a Cartesian of the **attr**_ibute observer functions of the types of monitorable attributes of part p. Given `record_attribute_type_names(p)` analysis function `monitorable_attributes(record_attribute_type_names(p))` yields $\mathsf{Mon\_Vals}_p$;
- $\mathsf{Prgr\_Vals}_p$ is a Cartesian of the type of programmable attributes of part p. Given `record_attribute_type_names(p)` analysis function `programmable_attributes(record_attribute_type_names(p)).` yields $\mathsf{Prgr\_Vals}_p$;
- { ch[{i,j}] | ... } specifies the channels over which part p behaviours, B_p, may communicate; and
- **Unit** is the type name for the () value[81] ■

The Cartesian arguments may "degenerate" to the non-Cartesian of no, or just one type identifier, In none, i.e., (), then () may be skipped. If one, e.g., (a), then (a) is listed.

Example: 32 *Road Transport Behaviour Signatures*:

[80] We have Schönfinckel'ed `https://en.wikipedia.org/wiki/Moses_Schönfinkel#Further_reading` (Curried `https://en.wikipedia.org/wiki/Currying`) the function type

[81] – You may "read' () as the value yielded by a statement, including a never-terminating function

value

$\mathsf{hub}\colon \mathsf{HI}\to\mathsf{MereoH}\to(\mathsf{H\Omega}\times...)\to(...)\to(\mathsf{HHist}\times...)$
$\quad\to\{\mathsf{ch}[\{\mathbf{uid}\_\mathsf{H}(p),\mathsf{ai}\}]|\mathsf{ai}{:}\mathsf{AI}\bullet\mathsf{ai}\in as_{uid}\}$ **Unit**
$\mathsf{link}\colon \mathsf{LI}\to\mathsf{MereoL}\to(\mathsf{LEN}\times...)\to(...)\to(\mathsf{LHist}\times...)$
$\quad\to\{\mathsf{ch}[\{\mathbf{uid}\_\mathsf{L}(p),\mathsf{ai}\}]|\mathsf{ai}{:}\mathsf{AI}\bullet\mathsf{ai}\in as_{uid}\}$ **Unit**
$\mathsf{automobile}\colon \mathsf{AI}\to\mathsf{MereoA}\to(...)\to(\mathbf{attr}\_\mathsf{AVel}\times\mathbf{attr}\_\mathsf{HAcc}\times...)\to(\mathsf{APos}\times\mathsf{AHist}\times...)$
$\quad\to\{\mathsf{ch}[\{\mathbf{uid}\_\mathsf{H}(p),\mathsf{ri}\}]|\mathsf{ri}{:}(\mathsf{HI}|\mathsf{LI})\bullet\mathsf{ri}\in hs_{uid}\cup ls_{uid}\}$ **Unit**

Here we have suggested additional part attributes: monitorable automobile velocity and acceleration, AVel, AAcc, and omitted other attributes ■

B.9.5.2 Inert Arguments: Some Examples

Let us give some examples of inert attributes of automobiles. (i) Driving uphill, one a level road, or downhill, excert some `inert` "drag" or "pull". (ii) Velocity can be treated as a reactive attribute – but it can be [approximately] calculated on the basis of, for example, these `inert` attributes: drag/pull and accelerator pedal pressure, and the `static` engine power attribute.

B.9.5.3 Behaviour Definitions

A typical, informal rendition of abstracted behaviours, `BA, BC, BD, ...` is shown in Fig. B.9.5.3 on the next page.

Figure B.9.5.3 on the facing page should be understood as follows:[82] The **bold faced** labels **BA, BB, BC, ...** are meant to designate behaviours. The **black arrows**, from behaviour **Bx** to behaviour **By** are meant to designate CSP-like *communications* from **Bx** to **By**. The **open arrows** ("white"), from behaviour **Bx** to behaviour **By** are meant to designate possible CSP-like *communications* from **Bx** to **By**. These latter communications, the "possible" ones, are then thought of as *in response* to the "earlier", in the figure: "immediately prior, next to" communication from **Bx** to **By**.

Figure B.9.5.3 on the next page is now given a more precise "meaning" – with this "meaning" suggesting a general "pattern" for behaviour definitions:

30. There are behaviours `B, ...` with identities `bi, ...` .
    (a) These behaviours,typically, have the form of internal, $\sqcap$, non-deterministically "choosing" between
    (b) *pro-actively* initiating communications with other behaviors
    (c) and *re-actively* responding to such initiatives.

value
30a. B(bi)(mereo)(stat)(mon)(prg) $\equiv$
30b. pro_active_B(bai)(mereo)(stat)(mon)(prg)
30c. $\sqcap$ re_active_B(bai)(mereo)(stat)(mon)(prg)

[82] The explanation of Fig. B.9.5.3 is in now way an attempt to explain the semantics of behaviours. That is left to the RSL^{+} formalization's.

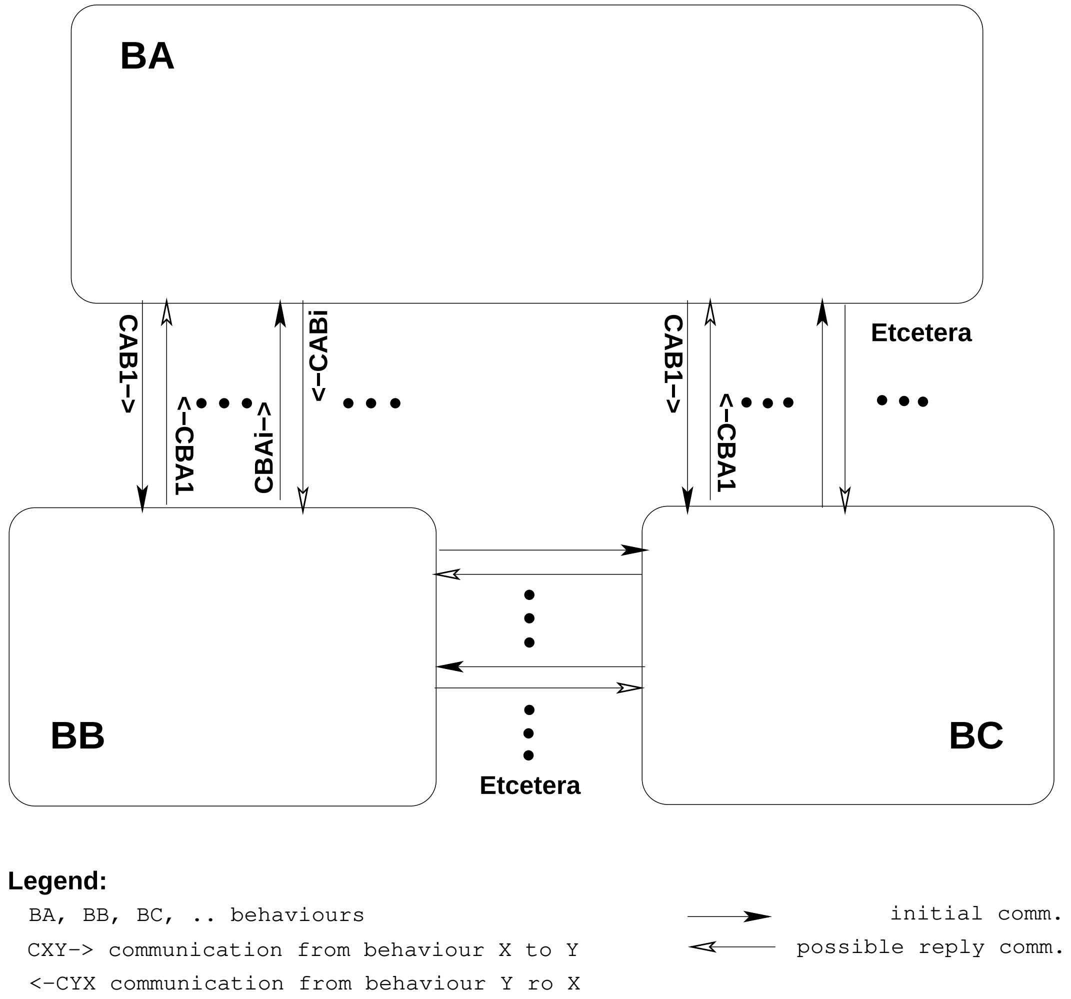

BA
CAB1->
<-CBA1
CBAi->
<-CABi
CAB1->
<-CBA1
Etcetera
BB
BC
Etcetera
Legend:
BA, BB, BC, .. behaviours
CXY-> communication from behaviour X to Y
<-CYX communication from behaviour Y ro X
initial comm.
possible reply comm.

Figure B.3: Communicating Behaviours

31. ι30b π228 The pro-active behaviour (B) internal deterministically (⌈⌉) choosing between a number of initiating actions:
    (a) action 1,
    (b) action 2,
    (c) ...,
    (d) action n.

value

```
31., ι30b π228. pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
31a.                 B_action_1(bi)(mereo)(stat)(mon)(prg)
31b.              ⌈⌉ B_action_2(bi)(mereo)(stat)(mon)(prg)
31c.              ⌈⌉ ...
31d.              ⌈⌉ B_action_m(bi)(mereo)(stat)(mon)(prg)
```

32. ι30b π228. The `responding` behaviour (`B`) reacts to a number of such initiating actions by
    (a) external non-deterministically ([]) offering to accept messages from responding behaviours,
    (b) and then performing corresponding actions.

```
value
32a., ι30b π228.  respond_B(bi)(mereo)(stat)(mon)(prg) ≡
32a.              let msg = [] { comm[{bj,bi}] ? | bj:BI • bj ∈ bis } in
32b.              react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end
```

33. The `react_behaviour_B` inquires as to the type of the message, say, a command, received (**?**): if it is:
    (a) of type `Cmd_i` then it performs action `act_cmd_i`,
    (b) of type `Cmd_j` then it performs action `act_cmd_j`,
    (c) ..., or
    (d) of type `Cmd_k` then it performs action `act_cmd_k`.
    (e) If it is of neither of these types then it "skips" treatment of that response by resuming to be the behaviour `B`.

```
value
33.  react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) ≡
33a.     is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
33b.     is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
33c.     ...,
33d.     is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
33e.     _ → B(bi)(mereo)(stat)(mon)(prg)
```

B.9.5.4 Action Definitions

"Actions are what makes behaviours meaningful" We remind the reader that our function (incl. behaviour) definitions are expressed in a functional, "applicative", style. [that is, there are no assignable variables] The actions elaborate to **values**. These values may be Booleans, numbers, sets, Cartesians, lists, maps and functions (over these), or the values by be `()`, of type **Unit**, as are the values (also of never-ending) behaviours.

Action signatures usually "follow that", i.e., are the same as "their" initiating behaviour signatures.

34. Actions, as semantic quantities,
    (a) evaluate some values,
    (b) typically change some programmable attributes,
    (c) and may communicate, “issue” or inform, to some other behaviours, some requests, respectively information –
    (d) whereupon the “revert”, “tail-recursively” to the activating Behaviour.

```
34.  action_i(bi)(mereo)(stat)(mon)(prg) ≡
34a.     let v = evaluate_i(bi)(mereo)(stat)(mon)(prg) in
34b.     let (bj,prg′) = elaborate_i(v)(bi)(mereo)(stat)(mon)(prg) in
34c.     comm[{bi,bj}] ! ℰ(prg′) ;
34d.     behaviour(bi)(mereo)(stat)(mon)(prg′)
34.      end end
```

Variants of Item ι34c π231 are also used:

```
{ comm[{bi,bj}] ! ℰ(prg′) | bj ∈ bis } ;
```

where `bj` ranges over `bis`, a set of behaviour identities.

B.9.5.5 Behaviour Invocation

Schema 8 *Behaviour Invocation*

Behaviours are invoked as follows:

```
“ $B_p$(uid_$_p$(p))[83]
      (mereo_P(p))
            (attr_staA$_1$(p),...,attr_staA$_s$(p))
                  (attr_inertA$_1$(p),...,attr_inertA$_i$(p))
                        (attr_monA$_1$,...,attr_monA$_m$)
                              (attr_prgA$_1$(p),...,attr_prgA$_p$(p)) ”
```

- All arguments are passed *by value*.
- The *uid* value is never changed.
- The *mereology* value is usually not changed.
- The *static attribute* values are fixed, never changed.

[83] We show the arguments of the invocation on separate lines only for readability. That is: normally we show the invocation arguments as B(...)(...)(...)(...)(...).

- The *inert attribute* values are fixed, but can be updated by receiving explicit input communications.
- The *monitorable attribute* values are functions, i.e., it is as if the "actual" monitorable values are passed *by name* !
- The *programmable attribute* values are usually changed, "updated", by actions described in the behaviour definition ■

B.9.5.6 Argument References

Within behaviour descriptions, see next section, references are made to the behaviour arguments. References, a, to *unique identifier, mereology, static* and *progammable attribute* arguments yield their value. References, a, to *monitorable attribute* arguments also yield their value. This value is an **attr**_A observer function. To yield, i.e., read, the monitorable attribute value this function is applied to that behaviour's uniquely identified part, p_{uid}, in the global part state, σ. To update,, i.e., write, say, to a value v, for the case of a *biddable, monitorable* attribute, that behaviour's uniquely identified part, p_{uid}, in the global part state, σ, shall have part p_{uid}'s A attribute changed to v – with all other attribute values of p_{uid} unchanged. Common to both the read and write functions is the *retrieve part* function:

* Given a unique part identifier, pi, assumed to be that of an existing domain part,

* retr_part *read*s the global [all parts] variable σ to retrieve that part p whose unique part identifier is pi.

value

```
[∗]   retr_part: PI → P  read
[∗]   retr_part(pi) ≡ let p:P • p ∈ c σ ∧ uid_P(p)=pi in p end
[∗]   pre: ∃ p:P • p ∈ c σ ∧ uid_P(p)=pi
```

You may think of the functions being illustrated in this section, Sect. B.9.5.6, retr_part, read_A_from_P and update_P_with_A, as "belonging" to the description language, but here suitably expressed for any domain, that is, with suitable substitutions for A and P.

B.9.5.6.1 Evaluation of Monitorable Attributes.

35. Let pi:PI be the unique identifier of any part, p, with monitorable attributes, let A be a monitorable attribute of p, and let ηA be the name of attribute A.
36. Evaluation of the [current] attribute A value of p is defined by function read_A_from_P.

value

```
35.   pi:PI, a:A, ηA:η𝕋
36.   read_A_from_P: PI × 𝕋 → read σ A
36.   read_A(pi,ηA) ≡ attr_A(retr_part(pi))
```

B.9.5.6.2 Update of Biddable Attributes

37. The update of a monitorable attribute A, with attribute name ηA of part p, identified by pi, to a new value **write**s to the global part state σ.

38. Part p is retrieved from the global state.

39. A new part, p′ is formed such that p′ is like part p:

    (a) same unique identifier,

    (b) same mereology,

    (c) same attributes values,

    (d) except for A.

40. That new p' replaces p in σ.

```
value
37.   $\sigma$, a:A, pi:PI, $\eta$A:$\eta\mathbb{T}$

37.   update_P_with_A: PI $\times$ A $\times$ $\eta\mathbb{T}$ $\rightarrow$ write $\sigma$
37.   update_P_with_A(pi,a,$\eta$A) $\equiv$
38.       let p = retr_part(pi) in
39.       let p′:P •
39a.             uid_P(p′)=pi
39b.             $\wedge$ mereo_P(p)=mereo_P(p′)
39c.             $\wedge$ $\forall$ $\eta$A′ $\in$ record_attribute_type_names(p)\{$\eta$A}
39c.                 $\Rightarrow$ attr_A′(p)=attr_A′(p′)
39d.             $\wedge$ attr_A(p′)=a in
40.       $\sigma$ := c $\sigma$ \ {p} $\cup$ {p′}
37.       end end
38.     pre: $\exists$ p:P • p $\in$ c $\sigma$ $\wedge$ uid_P(p)=pi
```

B.9.5.7 **Behaviour Description – Examples**

Behaviour descriptions rely strongly on CSPs' [47] expressivity. Leaving out some details (_, '...'), and *without "further ado"*, we exemplify.

Example: 33 *Automobile Behaviour at Hub*:

41. We abstract automobile behaviour at a Hub (hi).

    (a) Either the automobile remains in the hub,

    (b) or, internally non-deterministically,

    (c) leaves the hub entering a link,

    (d) or, internally non-deterministically,

(e) stops.

```
41   automobile(ai)(ris)(...)(atH(hi),ahis,_) ≡
41a      automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_)
41b      ⌈⌉
41c      automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis,_)
41d      ⌈⌉
41e      automobile_stop(ai)(ris)(...)(atH(hi),ahis,_)
```

42. [41a] The automobile remains in the hub:
    (a) time is recorded,
    (b) the automobile remains at that hub, “idling”,
    (c) informing (“first”) the hub behaviour.

```
42   automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_) ≡
42a      let τ = record_𝕋𝕀𝕄𝔼 in
42c      ch[{ai,hi}] ! τ ;
42b      automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩⁀ahis,_) end
```

43. [41c] The automobile leaves the hub entering link li:
    (a) time is recorded;
    (b) hub is informed of automobile leaving and link that it is entering;
    (c) “whereupon” the vehicle resumes (i.e., “while at the same time” resuming) the vehicle behaviour positioned at the very beginning (0) of that link.

```
43   automobile_leaving_hub(ai)({li}∪ris)(...)(atH(hi),ahis,_) ≡
43a      let τ = record_𝕋𝕀𝕄𝔼   in
43b      (ch[{ai,hi}] ! τ ∥ ch[{ai,li}] ! τ) ;
43c      automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩⁀ahis,_) end
43       pre: [hub is not isolated]
```

The choice of link entered is here expressed (43) as a non-deterministic choice[84]. One can model the leave hub/enter link otherwise.

44. [41e] Or the automobile “disappears — off the radar” !

```
44   automobile_stop(ai)(ris),(...)(atH(hi),ahis,_) ≡ stop ■
```

rm

[84] – as indicated by the **pre**- condition: the hub mereology must specify that it is not isolated. Automobiles can never leave isolated hubs.

B.9.6 Behaviour Initialization.

For every manifest part it must be described how its behaviour is initialized.

Example: 34 *Road Transport Initialization*: We "wrap up" the main example of this report-paperreport: We omit treatment of monitorable attributes.

45. Let us refer to the system initialization as an action.
46. All hubs are initialized,
47. all links are initialized, and
48. all automobiles are initialized.

```
value
45. rts_initialisation: Unit → Unit
45. rts_initialisation() ≡
46.     || { hub(uid_H(l))(mereo_H(l))(attr_HΩ(l),...)(attr_HΣ(l),...)| h:H • h ∈ hs }
47.     || || { link(uid_L(l))(mereo_L(l))(attr_LEN(l),...)(attr_LΣ(l),...)| l:L • l ∈ ls }
48.     || || { automobile(uid_A(a))(mereo_A(a))(attr_APos(a)attr_AHis(a),...) | a:A • a ∈ as }
```

We have here omitted possible monitorable attributes. For *hs*, *ls*, *as* we refer to Sect. B.8.1.4
■

B.10 Facets

In this section we shall briefly overview the concept of facets. By a *domain facet* we shall understand one amongst a finite set of generic ways of analyzing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain.

We leave it to [20, *Chapter 8, pages 205–240*] to detail the principles, procedures, techniques and tool for describing facets.

These are the facets that we have so far identified:

- intrinsics
- support technology
- rules & regulations
- scripts
- license languages
- management & organization
- human behaviour

B.10.1 Intrinsics

By domain intrinsics we shall understand those *phenomena and concepts* of a domain *which are basic* to any of the other facets, with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.

B.10.2 **Support Technology**

By a domain support technology we shall understand ways and means of *implementing* certain observed phenomena or certain conceived concepts.

B.10.3 **Rules & Regulations**

- By a *domain rule* we shall understand some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duties, respectively when performing their functions.

- By a *domain regulation* we shall understand some text (in the domain) which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.

B.10.4 **Scripts**

By a domain script we shall understand the structured, almost, if not outright, formally expressed, wording of a procedure on how to proceed, one that has legally binding power, that is, which may be contested in a court of law.

A special "subclass" of scripts are those of commands.

Commands are syntactic entities. Semantically they denote state changes. The state referred to is the state of the domain. Domain facets, as a wider concept than just commands, were first treated in [19, *Chapter 8*] which places facets in the wider context of domain modeling. Commands are but just one of the many kinds of script facets.

Commands are defined syntactically, and given semantics in the definition of perdurant behaviours, one set of simple actions per command.

B.10.5 **License Languages**

A *license* is a right or permission granted in accordance with law by a competent authority to engage in some business or occupation, to do some act, or to engage in some transaction which but for such license would be unlawful.

A *license language* is a ["small"] language (with syntax, semantics and pragmatics) in which to describe licenses.

B.10.6 **Management & Organization**

- By domain management we shall understand such people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, Sect. 8.4) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who "backstops" complaints from lower management levels and from "floor" staff.

- By domain organization we shall understand (vi) the structuring of management and non-management staff "oversee-able" into clusters with "tight" and "meaningful" relations; (vii) the allocation of strategic, tactical and operational concerns to within management and non-management staff clusters; and hence (viii) the "lines of command": who does what, and who reports to whom, administratively and functionally.

B.10.7 Human Behaviour

By domain human behaviour we shall understand any of a quality spectrum of carrying out assigned work: from (i) careful, diligent and accurate, via (ii) sloppy dispatch, and (iii) delinquent work, to (iv) outright criminal pursuit.

B.11 Conclusion

We have summarized a method to be used by [human] domain analyzers cum describers in studying and modeling domains. Our previous publications [16, 17, 20] have, with this paper, found its most recent, we risk to say, for us, final form.

Of course, domain models can be developed without the calculi presented in this paper. And was for many years. From the early 1990s a number of formal models of railways were worked out [43, 6, 8, 32, 7]. The problem, though, was still, between 1992 and 2016, *"where to begin, how to proceed and when to end"*. The domain analysis & description ontology and, hence calculus, of this paper shows how. The systematic approach to domain modeling of this ontology and calculus has stood its test of time. The `Internet` 'publication' `https://www.imm.dtu.dk/~dibj/2021/dd/dd.pdf` include the following domain models[85] from the 2007–2024 period. Their development has helped hone the method of the present paper.

B.11.1 Previous Literature

To the best of my knowledge there is no prior, comparable publications in the field of domain science and engineering. Closest would be Michael A. Jackson's [52]. Well, most computer scientists working in the field of correctness of programs, from somewhat "early on", stressed the importance of making proper assumptions about the domain, They would then express these "in-line", as appropriate predicates, with their proofs. Michael A. Jackson, lifted this, to a systematic treatment of the domain in his triplet 'Problem Frame Approach': *program, machine, domain* [51]. But Jackson did not lift his problem frame concern into a proper study of domains.

[85]

- *Graphs*,
- *Rivers*,
- *Canals*,
- *Railways*,
- *Road Transport*,
- *The "7 Seas"*,
- *The "Blue Skies"*,
- *Credit Cards*,
- *Weather Information*,
- *Documents*,
- *Urban Planning*,
- *Swarms of Drones*,
- *Container Terminals*,
- *A Retailer Market*,
- *Assembly Lines*,
- *Bookkeeping*,
- *Shipping*,
- *Stock Exchanges*,
- *Web Transactions*, etc.

B.11.2 **The Method**

So the method procedure is this: (1) First analyze and describe the *external qualities* of the chosen domain. (2) For each of the so-described endurants You then analyze and describe their *internal qualities.* (2.1) First their *unique identification.* (2.2) Then their *mereology.* (2.3) Then their *attributes.* (2.4) And finally possible *intentional pulls.* (3) First then are You ready to tackle the issue of perdurants. (3.1) Decide upon the *state.* (That may already have been done in connection with (1).) (3.2) Then describe the *channels.* (3.3) Then analyze and describe [part] *behaviour signatures.* (3.4) Then describe *behaviour invocation.* (3.5) Then *behaviour (body) definitions.* (4) Finally describe *domain initialization.*

B.11.3 **Specification Units**

The method thus focuses, step-by-step, on the development of the following *specification units:* **type** specification units, **value** specification units, **axiom** specification units, **variable** declaration units, and **channel** declaration units.

There are two forms of *type* specifications: (α) introduction of sorts, i.e., type names, and (β) specification of types: pairs of new type names and type expressions – as atomic, alternate or composite types: set, Cartesian, list, map or function types.

There are basically three forms of value specification units: (i) (“simple”) naming of values, (ii) signature of functions: function name and function type, and (iii) signature of (endurant **obs_**, unique identifier **uid_**, mereology, **mereo_**, and attribute **attr_**) observer functions.

B.11.4 **Object Orientation**

So far we have not used the term ‘object’ !

We shall now venture the following:

The combined description of endurant parts and their perdurant behaviour form an object definition.

You can then, for yourself, develop a way of graphically presenting these object definitions such that each part type is represented by a box that contains the specification units for [all] external and internal endurant qualities as well as for the perdurant [part] behaviour signatures and definitions; and such that the mereologies of these parts is represented by [possibly directed] lines connecting relevant boxes.

That is, an object concept solely based on essentially inescapable world description facts – as justified by Sørlander’s Philosophy [60, 63, 64, 65] ! No “finicky” programming language “tricks” !

We leave it to the reader to compare this definition to those of so-called object-oriented programming languages.

B.11.5 **Other Domain Modeling Approaches**

[67] shows fragments of a number of expertly expressed domain models. They are all expressed in `RAISE`.[86] But they are not following the method of this paper. In other words, it is

[86] Other approaches could also be used: `VDM` [33, 34, 39], `Z` [69], `Alloy` [49], `CafeOBJ` [40], etc.

possible to develop domain models not using the method ! This author has found, however, that following the method – developed after the projects reported in [67] – leads to far less problematic situations – in contrast to my **not** adhering strictly to the method. In other words, based on this subjective observation, we advice using the method.

There is thus no proof that following the method does result in simpler, straightforward developments.

But we do take the fact that we can justify the method, cf. Fig. B.1 on page 205, on the basis on the inevitability of describing the world as per philosophy of Kai Sørlander [60, 63, 64, 65], and that that may have a bearing on the experienced shorter domain description development efforts.

B.11.6 How Much ? How Little ?

How wide must we *cast the net* when studying a domain ? The answer to that question depends, we suggest, on whether our quest is for studying a domain in general, to see what might come out, or whether it is a study aiming at a specific model for a specific software development. In the former case *we cast the net* as we please – we suggest: as wide as possible, wider that for specific quests. In the latter case *we should cast the net* as "narrowly" as is reasonable: to fit those parts of a domain that we expect the requirements and software to deal with ! In this latter case we should assume that someone, perhaps the same developers, has first "tried their hand" on a wider domain.

B.11.7 Correctness

Today, 2024, software correctness appears focused on the correctness of algorithms, possibly involving concurrency. Correctness, of software, in the context of a specific domain, means that the software requirements are "correctly" derived from a domain description, and that the software design is correctly derived from the domain requirements, that is: $\mathbb{D},\mathbb{S} \models \mathbb{R}$. Advances in program proofs helps little if not including proper domain and requirements specifications.

B.11.8 Domain Facets

There is more to *domain modeling* than covered in this paper. In [12] and in [20, *Chapter 8*] we cover the concept of *domain facets*. General examples of domain facets are *support technologies*, *rules & regulations*, *scripts*, *license languages*, *management & organization*, and *human behaviour*.

B.11.9 Perspectives

Domain models can be developed for either of a number of reasons:

- (i) in order to *understand* a human-artifact domain;
- (ii) in order to *re-engineer the business processes* of a human-artifact domain; or
- (iii) in order to develop *requirements prescriptions* and, subsequently *software application* "within" that domain.

[(ii)] We refer to [44, 45] and [9, *Vol. 3, Chapter 19, pages 404–412*] for the concept of *business process engineering*. [(iii)] We refer to [20, *Chapter 9*] for the concept of *requirements engineering*.

B.11.10 The Semantics of Domain Models

The meaning of domain models, such as we describe them in this paper, is, "of course", the actual, real domain "out there" ! One could, and, perhaps one should, formulate a mathematical semantics of the models, that is, of the **is_...**, **obs_...**, **uid_...**, **mereo_...** and **attr_...** analysis and description functions and what they entail (e.g., the type name labels: $\eta\mathbb{T}$'s; etc.). An early such semantics description is given in [14].

B.11.11 Further on Domain Modeling

Additional facets of domain modeling are covered in [10] and [20, *Chapter 8: Domain Facets.*]

B.11.12 Software Development

[10] and [20, *Chapter 9 Requirements*] show how to develop $\mathscr{R}$equirements prescriptions from $\mathscr{D}$omain descriptions. [9] shows how to develop $\mathscr{S}$oftware designs from $\mathscr{R}$equirements prescriptions.

B.11.13 Modeling

Domain descriptions, such as outlined in this paper, are models of domains, that is, of some reality. They need not necessarily lead to or be motivated by possible development of software for such domains. They can be experimentally researched and developed just for the sake of understanding domains in which man has had an significantly influence. They are models. We refer to [38] for complementary modeling based on Petri nets. The current author is fascinated by the interplay between graphical and textual descriptions of `HERAKLIT`, well, in general `Petri Nets`.

B.11.14 Philosophy of Computing

The Danish philosopher Kai Sørlander [60, 63, 64, 65] has shown that there is a foundation in philosophy for domain analysis and description. We refer to [21, *Chapter 2*] for a summary of his findings.

B.11.15 A Manifesto

So there is no excuse, anymore ! Of course we have developed interpreters and compilers for programming languages by first developing formal semantics for those languages [35, 36]. Likewise we must now do for the languages of domain stakeholders, at least for the domains covered by this paper. There really is no excuse !

Appendix C

The Tokyo Stock Exchange

This chapter was begun on January 24, 2010. It is being released, first time, January 28.

C.1 Introduction

This chapter shall try describe: narrate and formalise some facets of the (now "old"[87]) stock trading system of the TSE: Tokyo Stock Exchange (especially the 'matching' aspects).

C.2 The Problem

The reason that I try tackle a description (albeit of the "old" system) is that Prof. Tetsuo Tamai published a delightful paper [66, IEEE Computer Journal, June 2009 (vol. 42 no. 6) pp. 58-65)], Social Impact of Information Systems, in which a rather sad story is unfolded: a human error key input: an offer for selling stocks, although "ridiculous" in its input data (*"sell 610 thousand stocks, each at one (1) Japanese Yen"*, whereas one stock at 610,000 JPY was meant), and although several immediate — within seconds — attempts to cancel this "order", could not be canceled ! This lead to a loss for the selling broker at around 42 Billion Yen, at today's exchange rate, 26 Jan. 2010, 469 million US $s ![88] Prof. Tetsuo Tamai's paper gives a, to me, chilling account of what I judge as an extremely sloppy and irresponsible design process by TSE and Fujitsu. It also leaves, I think, a strong impression of arrogance on the part of TSE. This arrogance, I claim, is still there in the documents listed in Footnote 87.

So the problem is a threefold one of

[87] We write "old" since, as of January 4, 2010, that 'old' stock trading system has been replaced by the so-called arrowhead system. We refer to the following documents:

- http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet.html
- http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet-e.pdf
- http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet1e.pdf
- http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet2e.html

[88] So far three years of law court case hearing etc., has, on Dec. 4, 2009, resulted in complainant being awarded 10.7 billion Yen in damages. See http://www.ft.com/cms/s/0/e9d89050-e0d7-11de-9f58--00144feab49a.html.

- **Proper Requirements:** How does one (in this case a stock exchange) prescribe (to the software developer) what is required by an appropriate hardware/software system for, as in this case, stock handling: acceptance of buy bids and sell offers, the possible withdrawal (or cancellation) of such submitted offers, and their matching (i.e., the actual trade whereby buy bids are marched in an appropriate, clear and transparent manner).

- **Correctness of Implementation:** How does one make sure that the software/hardware system meets customers' expectations.

- **Proper Explanation to Lay Users:** How does one explain, to the individual and institutional customers of the stock exchange, those offering stocks for sale of bids for buying stocks – how does one explain – in a clear and transparent manner the applicable rules governing stock handling.[89]

I shall only try contribute, in this document, to a solution to the first of these sub-problems.

C.3 A Domain Description

C.3.1 Market and Limit Offers and Bids

49. A market sell offer or buy bid specifies
    (a) the unique identification of the stock,
    (b) the number of stocks to be sold or bought, and
    (c) the unique name of the seller.
50. A limit sell offer or buy bid specifies the same information as a market sell offer or buy bid (i.e., Items 49a–49c), and
    (d) the price at which the identified stock is to be sold or bought.
51. A trade order is either a (mkMkt marked) market order or (mkLim marked) a limit order.
52. A trading command is either a sell order or a buy bid.
53. The sell orders are made unique by the mkSell "make" function.
54. The buy orders are made unique by the mkBuy "make" function.

```
type
   49  Market = Stock_id × Number_of_Stocks × Name_of_Customer
   49a     Stock_id
   49b     Number_of_Stocks = {|n•n:Nat∧n≥1|}
```

[89] The rules as explained in the Footnote 87 on the preceding page listed documents are far from clear and transparent: they are full of references to fast computers, overlapping processing, etc., etc.: matters with which these buying and selling customers should not be concerned — so, at least, thinks this author !

```
49c     Name_of_Customer
50  Limit = Market × Price
50d     Price = {|n•n:Nat∧n≥1|}
51  Trade == mkMkt(m:Market) | mkLim(l:Limit)
52  Trading_Command = Sell_Order | Buy_Bid
53  Sell_Order == mkSell(t:Trade)
54  Buy_Bid == mkBuy(t:Trade)
```

C.3.2 Order Books

55. We introduce a concept of linear, discrete time.

56. For each stock the stock exchange keeps an order book.

57. An order book for stock s_{id} : *SI* keeps track of limit buy bids and limit sell offers (for the *id*entified *s*tock, s_{id}), as well as the market buy bids and sell offers; that is, for each price

    (d) the number of stocks, designated by unique order number, offered for sale at that price, that is, limit sell orders, and

    (e) the number of stocks, by unique order number, bid for buying at that price, that is, limit buy bid orders;

    (f) if an offer is a market sell offer, then the number of stocks to be sold is recorded, and if an offer is a market buy bid (also an offer), then the number of stocks to be bought is recorded,

58. Over time the stock exchange displays a series of full order books.

59. A trade unit is a pair of a unique order number and an amount (a number larger than 0) of stocks.

60. An amount designates a number of one or more stocks.

type

```
55  T, On  [Time, Order number]
56  All_Stocks_Order_Book = Stock_Id $\underset{m}{\rightarrow}$ Stock_Order_Book
57  Stock_Order_Book = (Price $\underset{m}{\rightarrow}$ Orders) × Market_Offers
57  Orders:: so:Sell_Orders × bo:Buy_Bids
57d     Sell_Orders = On $\underset{m}{\rightarrow}$ Amount
57e     Buy_Bids = On $\underset{m}{\rightarrow}$ Amount
57f  Market_Offers :: mkSell(n:Nat) × mkBuy(n:Nat)
58  TSE = T $\underset{m}{\rightarrow}$ All_Stocks_Order_Book
59  TU = On × Amount
60  Amount = {|n•Nat∧n≥1|}
```

C.3.3 Aggregate Offers

61. We introduce the concepts of aggregate sell and buy orders for a given stock at a given price (and at a given time).

62. The aggregate sell orders for a given stock at a given price is

    (g) the stocks being market sell offered and

    (h) the number of stocks being limit offered for sale at that price or lower

63. The aggregate buy bids for a given stock at a given price is

    (i) including the stocks being market bid offered and

    (j) the number of stocks being limit bid for buying at that price or higher

```
value
  62  aggr_sell: All_Stocks_Order_Book × Stock_Id × Price → Nat
  62  aggr_sell(asob,sid,p) ≡
  62    let ((sos,_),(mkSell(ns),_)) = asob(sid) in
  62g      ns +
  62h      all_sell_summation(sos,p) end
  63  aggr_buy: All_Stocks_Order_Book × Stock_Id × Price → Nat
  63  aggr_buy(asob,sid,p) ≡
  63    let ((_,bbs),(_,mkBuy(nb))) = asob(sid) in
  63i     nb +
  63j     nb + all_buy_summation(bbs,p) end

  all_sell_summation: Sell_Orders × Price → Nat
  all_sell_summation(sos,p) ≡
     let ps = {p′|p′:Prices • p′ ∈ dom sos ∧ p′ ≥ p} in accumulate(sos,ps)(0) end

  all_buy_summation: Buy_Bids × Price → Nat
  all_buy_summation(bbs,p) ≡
     let ps = {p′|p′:Prices • p′ ∈ dom bos ∧ p′ ≤ p} in accumulate(bbs,ps)(0) end
```

The auxiliary accumulate function is shared between the all_sell_summation and the all_buy_summation functions. It sums the amounts of limit stocks in the price range of the accumulate function argument ps. The auxiliary sum function sums the amounts of limit stocks — “pealing off” the their unique order numbers.

```
value
  accumulate: (Price →m Orders) × Price-set → Nat → Nat
  accumulate(pos,ps)(n) ≡
     case ps of {} → n, {p}∪ ps′ → accumulate(pos,ps′)(n+sum(pos(p)){dom pos(p)}) end

  sum: (Sell_Orders|Buy_Bids) → On-set → Nat
  sum(ords)(ns) ≡
     case ns of {} → 0, {n}∪ ns′ → ords(n)+sum(ords)(ns′) end
```

To handle the sub_limit_sells and sub_limit_buys indicated by Item 65c of the Itayose "algorithm" we need the corresponding sub_sell_summation and sub_buy_summation functions:

```
value
   sub_sell_summation: Stock_Order_Book × Price → Nat
   sub_sell_summation(((sos,_),(ns,_)),p) ≡ ns +
      let ps = {p′|p′:Prices • p′ ∈ dom sos ∧ p′ > p} in accumulate(sos,ps)(0) end

   sub_buy_summation: Stock_Order_Book × Price → Nat
   sub_buy_summation(((_,bbs),(_,nb)),p) ≡ nb +
      let ps = {p′|p′:Prices • p′ ∈ dom bos ∧ p′ < p} in accumulate(bbs,ps)(0) end
```

C.3.4 The TSE Itayose "Algorithm"

64. The TSE practices the so-called Itayose "algorithm" to decide on opening and closing prices[90]. That is, the Itayose "algorithm" determines a single so-called 'execution' price, one that matches sell and buy orders[91]:

65. The "matching sell and buy orders" rules:

    (a) *All market orders must be 'executed'*[92].

    (b) *All limit orders to sell/buy at prices higher/lower*[93] *than the 'execution price'*[94] *must be executed.*

    (c) *The following amount of limit orders to sell or buy at the execution prices must be executed: the entire amount of either all sell or all buy orders, and at least one 'trading unit'*[95] *from 'the opposite side of the order book'*[96].

- The 28 January 2010 version had lines
    - 65c$'_{\exists}$ name some_priced_buys, should have been, as now, some_priced_sells and
    - 65c$''_{\forall}$ name all_priced_buys, should have been, as now, all_priced_sells.

- My current understanding of *and assumptions* about the TSE is
    - that each buy bid or sell order concerns a number, *n*, of one or more of the same kind of stocks (i.e. sid).
    - that each buy bid or sell order when being accepted by the TSE is assigned a unique order number *on*, and

[90] [66, pp 59, col. 1, lines 4-3 from bottom, cf. Page 250]

[91] [66, pp 59, col. 2, lines 1–3 and Items 1.–3. after yellow, four line 'insert', cf. Page 250] These items 1.–3. are reproduced as "our" Items 65a–65c.

[92] To execute an order: ?????

[93] Yes, it should be: "higher/lower"

[94] Execution price: ?????

[95] Trading unit: ?????

[96] The opposite side of the order book: ?????

  - that this is reflected in some Sell_Orders or Market_Bids entry being augmented.[97]

- For current (Monday 22 Feb., 2010) lack of a better abstraction[98], I have structured the Itayose "Algorithm" as follows:
  - (65′) either a match can be made based on
    - all buys and
    - some sells,
  - ($65'_\vee$) or
  - (65″) a match can be made based on
    - aome buys and
    - all sells.

value

```
65 match: All_Stocks_Order_Book × Stock_Id → Price-set
65 match(asob,sid) as ps
65    pre: sid ∈ dom asob
65    post: ∀ p′:Price • p′ ∈ ps ⇒
65′          all_buys_some_sells(p′,ason,sid,ps) ∨
65′∨         ∨
65″          some_buys_all_sells(p′,ason,sid,ps)
```

- (65′) The all_buys_some_sells part of the above disjunction "calculates" as follows:
  - The all_buys... part includes
    - all the market_buys
    - all the buys properly below the stated price, and
    - all the buys at that price.
  - The ...some_sells part includes
    - all the market_sells
    - all the sells properly below the stated price, and
    - some of the buys at that price.

```
65′ all_buys_some_sells(p′,ason,sid,ps) ≡
65′   ∃ os:On-set •
65a′         all_market_buys(asob(sid))
65b′      + all_sub_limit_buys(asob(sid))(p′)
65c′      + all_priced_buys(asob(sid))(p′)
65a′    =   all_market_sells(asob(sid))
65b′      + all_sub_limit_sells(asob(sid))(p′)
65c′∃     + some_priced_sells(asob(sid))(p′)(os)
```

[97] The present, 22.2.2010, model "lumps" all market orders. This simplification must be corrected, as for the Sell_Orders and Market_Bids, the Market_Offers must be modeled as are Orders.

[98] One that I am presently contemplating is based on another set of **pre/post** conditions.

- ($65''$) As for the above, only “versed”.

```
65″  some_buys_all_sells(p′,ason,sid,ps) ≡
65″    ∃ os:On-set •
65a″           all_market_buys(asob(sid))
65b″         + all_sub_limit_buys(asob(sid))(p′)
65c″         + some_priced_buys(asob(sid))(p′)(os)
65a″     =     all_market_sells(asob(sid))
65b″         + all_sub_limit_sells(asob(sid))(p′)
65c″∀        + all_priced_sells(asob(sid))(p′) ∨
```

The match function calculates a set of prices for each of which a match can be made. The set may be empty: there is no price which satisfies the match rules (cf. Items 65a–65c below). The set may be a singleton set: there is a unique price which satisfies match rules Items 65a–65c. The set may contain more than one price: there is not a unique price which satisfies match rules Items 65a–65c. The single ($'$) and the double ($''$) quoted (65a–65c) group of lines, in the match formulas above, correspond to the Itayose “algorithm”'s Item 65c *‘opposite sides of the order book’* description. The existential quantification of a set of order numbers of lines $65'$ and $65''$ correspond to that “algorithms” (still Item 65c) point of *at least one ‘trading unit’*. It may be that the **post** condition predicate is only fulfilled for all trading units – so be it.

```
value
   all_market_buys: Stock_Order_Book → Amount
   all_market_buys((_,(_,mkBuys(nb))),p) ≡ nb

   all_market_sells: Stock_Order_Book → Amount
   all_market_sells((_,(mkSells(ns),_)),p) ≡ ns

   all_sub_limit_buys: Stock_Order_Book → Price → Amount
   all_sub_limit_buys(((,bbs),_))(p) ≡ sub_buy_summation(bbs,p)

   all_sub_limit_sells: Stock_Order_Book → Price → Amount
   all_sub_limit_sells((sos,_))(p) ≡ sub_sell_summation(sos,p)

   all_priced_buys: Stock_Order_Book → Price → Amount
   all_priced_buys((_,bbs),_)(p) ≡ sum(bbs(p))

   all_priced_sells: Stock_Order_Book → Price → Amount
   all_priced_sells((sos,_),_)(p) ≡ sum(sos(p))

   some_priced_buys: Stock_Order_Book → Price → On-set → Amount
   some_priced_buys((_,bbs),_)(p)(os) ≡
```

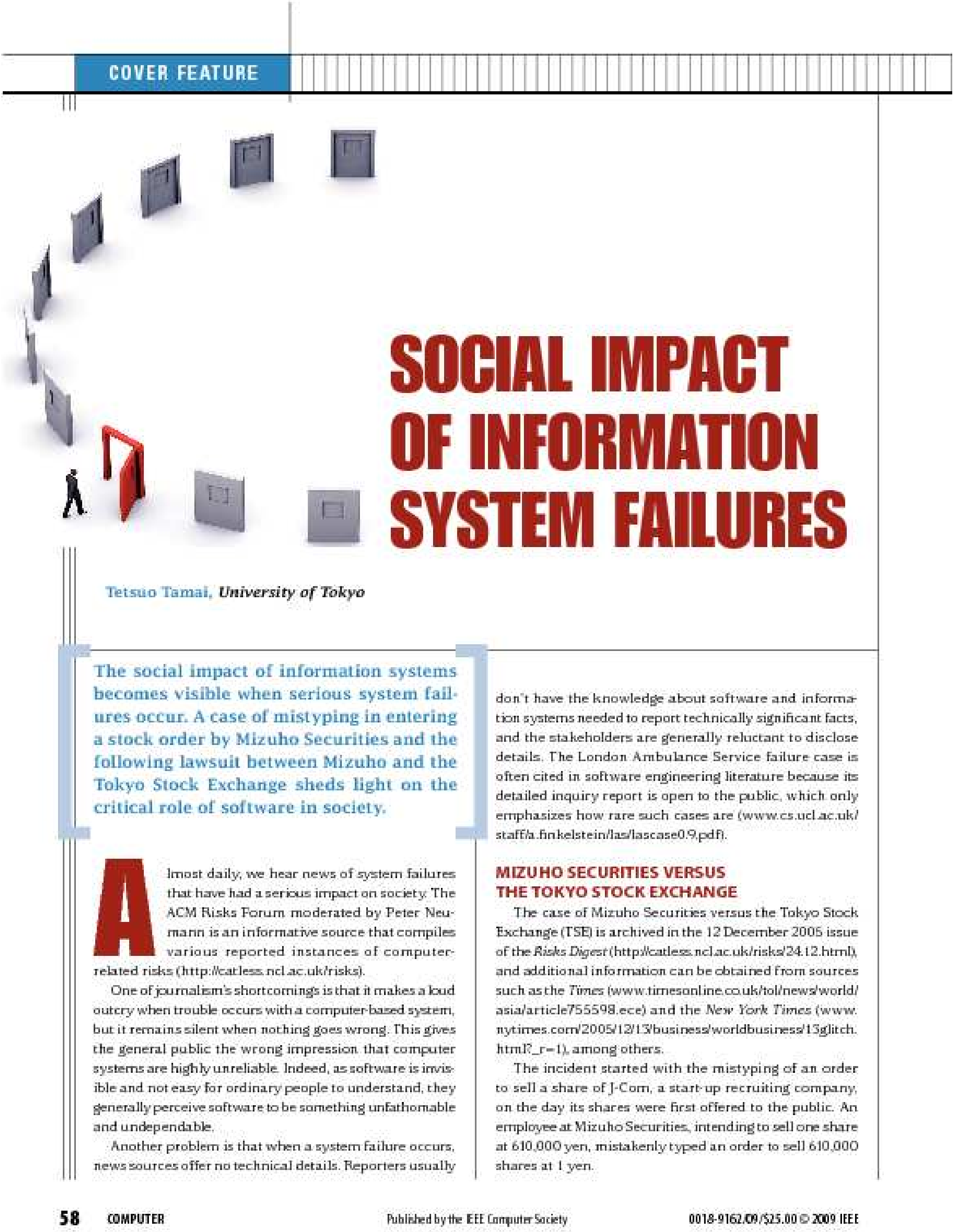
```
    let tbs = bbs(p) in if {}≠os∧os⊆dom tbs then sum(tbs)(os) else 0 end end

some_priced_sells: Stock_Order_Book → Price → On-set → Amount
some_priced_sells((sos,__),__)(p)(os) ≡
    let tss = sos(p) in if {}≠os∧os⊆dom tss then sum(tss)(os) else 0 end end
```

The formalization of the Itayose "algorithm", as well as that "algorithm" [itself], does not guarantee a match where a match "ought" be possible. The "stumbling block" seems to be the Itayose "algorithm"'s Item 65c. There it says: *'at least one trading unit'*. We suggest that a match could be made in which some of the stocks of a candidate trading unit be matched with the remaining stocks also being traded, but now with the stock exchange being the buyer and with the stock exchange immediately "turning around" and posting those remaining stocks as a TSE marked trading unit for sale.

66. It seems to me that the Tetsuo Tamai paper does not really handle
    (a) the issue of order numbers,
    (b) therefore also not the issue of the number of stocks to be sold or bought per order number.

67. Therefore the Tetsuo Tamai paper does not really handle
    (a) the situation where a match "only matches" part of a buy or a sell order.

For private, limited circulation only, I take the liberty of enclosing Tetsuo Tamai's IEEE Computer Journal paper.



COVER FEATURE

SOCIAL IMPACT OF INFORMATION SYSTEM FAILURES

Tetsuo Tamai, *University of Tokyo*

The social impact of information systems becomes visible when serious system failures occur. A case of mistyping in entering a stock order by Mizuho Securities and the following lawsuit between Mizuho and the Tokyo Stock Exchange sheds light on the critical role of software in society.

Almost daily, we hear news of system failures that have had a serious impact on society. The ACM Risks Forum moderated by Peter Neumann is an informative source that compiles various reported instances of computer-related risks (http://catless.ncl.ac.uk/risks).

One of journalism's shortcomings is that it makes a loud outcry when trouble occurs with a computer-based system, but it remains silent when nothing goes wrong. This gives the general public the wrong impression that computer systems are highly unreliable. Indeed, as software is invisible and not easy for ordinary people to understand, they generally perceive software to be something unfathomable and undependable.

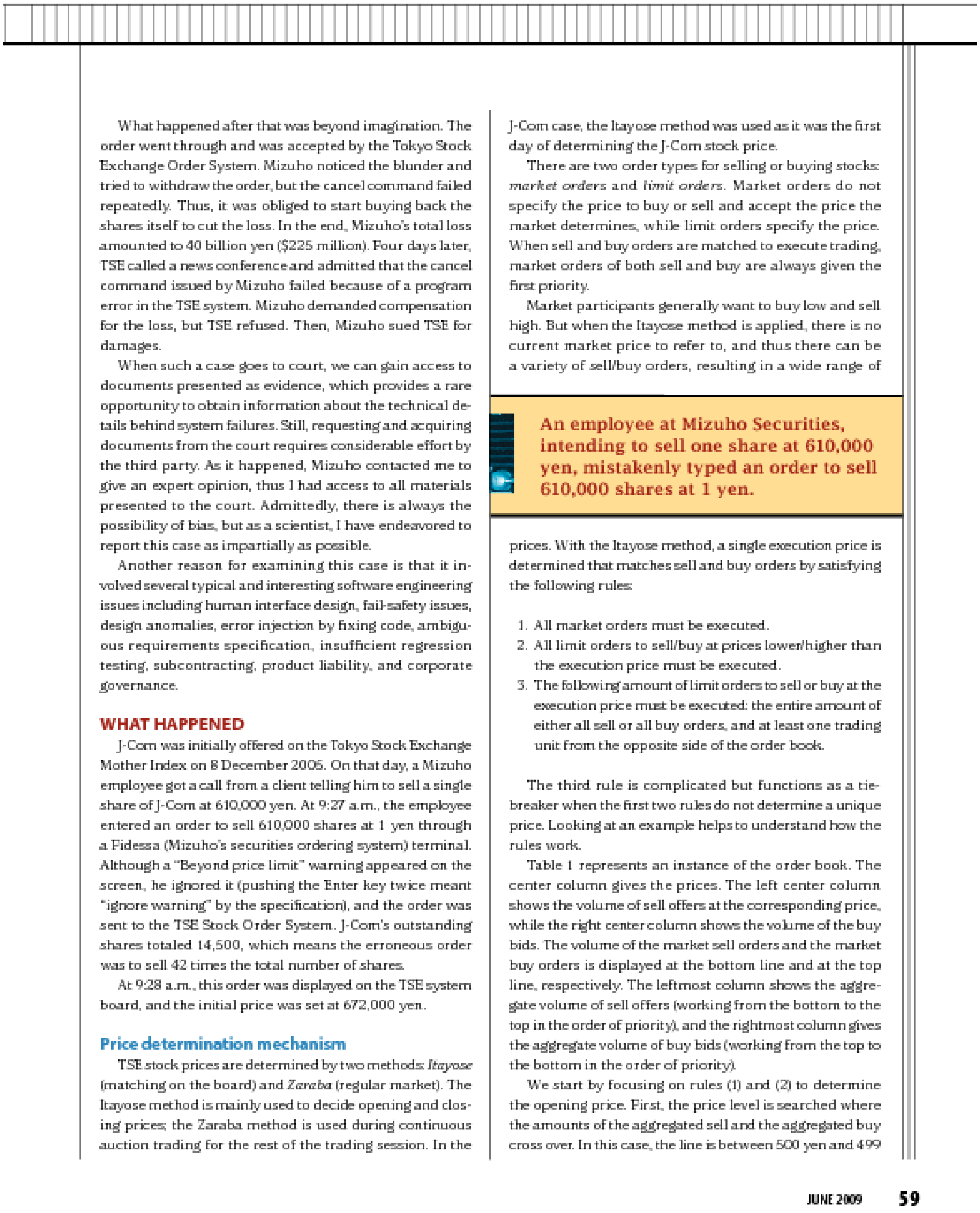
Another problem is that when a system failure occurs, news sources offer no technical details. Reporters usually don't have the knowledge about software and information systems needed to report technically significant facts, and the stakeholders are generally reluctant to disclose details. The London Ambulance Service failure case is often cited in software engineering literature because its detailed inquiry report is open to the public, which only emphasizes how rare such cases are (www.cs.ucl.ac.uk/staff/a.finkelstein/las/lascase0.9.pdf).

MIZUHO SECURITIES VERSUS THE TOKYO STOCK EXCHANGE

The case of Mizuho Securities versus the Tokyo Stock Exchange (TSE) is archived in the 12 December 2005 issue of the *Risks Digest* (http://catless.ncl.ac.uk/risks/24.12.html), and additional information can be obtained from sources such as the *Times* (www.timesonline.co.uk/tol/news/world/asia/article755598.ece) and the *New York Times* (www.nytimes.com/2005/12/13/business/worldbusiness/13glitch.html?_r=1), among others.

The incident started with the mistyping of an order to sell a share of J-Com, a start-up recruiting company, on the day its shares were first offered to the public. An employee at Mizuho Securities, intending to sell one share at 610,000 yen, mistakenly typed an order to sell 610,000 shares at 1 yen.

Published by the IEEE Computer Society 0018-9162/09/$25.00 © 2009 IEEE



What happened after that was beyond imagination. The order went through and was accepted by the Tokyo Stock Exchange Order System. Mizuho noticed the blunder and tried to withdraw the order, but the cancel command failed repeatedly. Thus, it was obliged to start buying back the shares itself to cut the loss. In the end, Mizuho's total loss amounted to 40 billion yen ($225 million). Four days later, TSE called a news conference and admitted that the cancel command issued by Mizuho failed because of a program error in the TSE system. Mizuho demanded compensation for the loss, but TSE refused. Then, Mizuho sued TSE for damages.

When such a case goes to court, we can gain access to documents presented as evidence, which provides a rare opportunity to obtain information about the technical details behind system failures. Still, requesting and acquiring documents from the court requires considerable effort by the third party. As it happened, Mizuho contacted me to give an expert opinion, thus I had access to all materials presented to the court. Admittedly, there is always the possibility of bias, but as a scientist, I have endeavored to report this case as impartially as possible.

Another reason for examining this case is that it involved several typical and interesting software engineering issues including human interface design, fail-safety issues, design anomalies, error injection by fixing code, ambiguous requirements specification, insufficient regression testing, subcontracting, product liability, and corporate governance.

WHAT HAPPENED

J-Com was initially offered on the Tokyo Stock Exchange Mother Index on 8 December 2005. On that day, a Mizuho employee got a call from a client telling him to sell a single share of J-Com at 610,000 yen. At 9:27 a.m., the employee entered an order to sell 610,000 shares at 1 yen through a Fidessa (Mizuho's securities ordering system) terminal. Although a "Beyond price limit" warning appeared on the screen, he ignored it (pushing the Enter key twice meant "ignore warning" by the specification), and the order was sent to the TSE Stock Order System. J-Com's outstanding shares totaled 14,500, which means the erroneous order was to sell 42 times the total number of shares.

At 9:28 a.m., this order was displayed on the TSE system board, and the initial price was set at 672,000 yen.

Price determination mechanism

TSE stock prices are determined by two methods: *Itayose* (matching on the board) and *Zaraba* (regular market). The Itayose method is mainly used to decide opening and closing prices; the Zaraba method is used during continuous auction trading for the rest of the trading session. In the J-Com case, the Itayose method was used as it was the first day of determining the J-Com stock price.

There are two order types for selling or buying stocks: *market orders* and *limit orders*. Market orders do not specify the price to buy or sell and accept the price the market determines, while limit orders specify the price. When sell and buy orders are matched to execute trading, market orders of both sell and buy are always given the first priority.

Market participants generally want to buy low and sell high. But when the Itayose method is applied, there is no current market price to refer to, and thus there can be a variety of sell/buy orders, resulting in a wide range of prices. With the Itayose method, a single execution price is determined that matches sell and buy orders by satisfying the following rules:

An employee at Mizuho Securities, intending to sell one share at 610,000 yen, mistakenly typed an order to sell 610,000 shares at 1 yen.

1. All market orders must be executed.
2. All limit orders to sell/buy at prices lower/higher than the execution price must be executed.
3. The following amount of limit orders to sell or buy at the execution price must be executed: the entire amount of either all sell or all buy orders, and at least one trading unit from the opposite side of the order book.

The third rule is complicated but functions as a tie-breaker when the first two rules do not determine a unique price. Looking at an example helps to understand how the rules work.

Table 1 represents an instance of the order book. The center column gives the prices. The left center column shows the volume of sell offers at the corresponding price, while the right center column shows the volume of the buy bids. The volume of the market sell orders and the market buy orders is displayed at the bottom line and at the top line, respectively. The leftmost column shows the aggregate volume of sell offers (working from the bottom to the top in the order of priority), and the rightmost column gives the aggregate volume of buy bids (working from the top to the bottom in the order of priority).

We start by focusing on rules (1) and (2) to determine the opening price. First, the price level is searched where the amounts of the aggregated sell and the aggregated buy cross over. In this case, the line is between 500 yen and 499

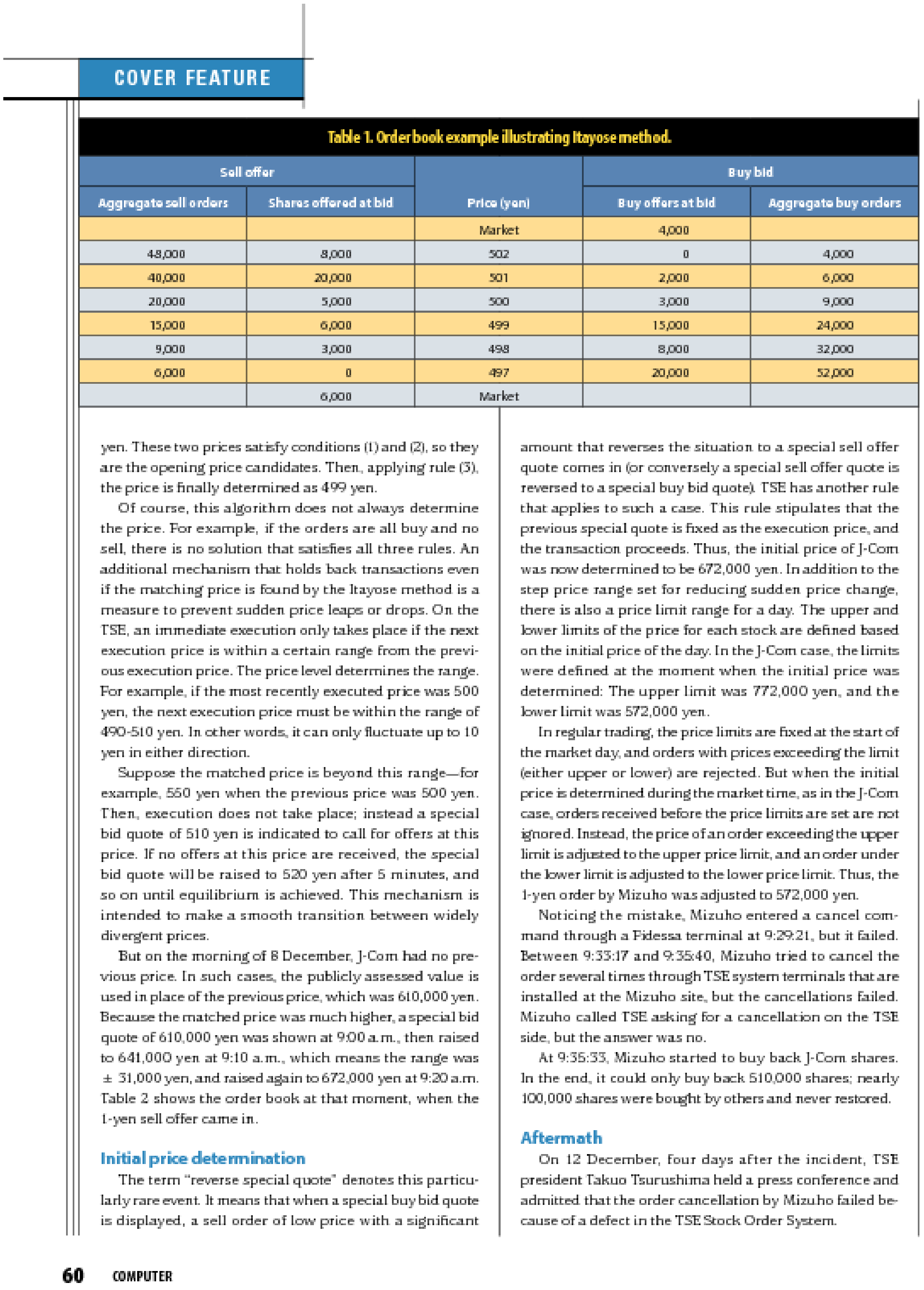


Table 1. Order book example illustrating Itayose method.

Sell offer		Price (yen)	Buy bid	
Aggregate sell orders	Shares offered at bid		Buy offers at bid	Aggregate buy orders
		Market	4,000	
48,000	8,000	502	0	4,000
40,000	20,000	501	2,000	6,000
20,000	5,000	500	3,000	9,000
15,000	6,000	499	15,000	24,000
9,000	3,000	498	8,000	32,000
6,000	0	497	20,000	52,000
	6,000	Market		

yen. These two prices satisfy conditions (1) and (2), so they are the opening price candidates. Then, applying rule (3), the price is finally determined as 499 yen.

Of course, this algorithm does not always determine the price. For example, if the orders are all buy and no sell, there is no solution that satisfies all three rules. An additional mechanism that holds back transactions even if the matching price is found by the Itayose method is a measure to prevent sudden price leaps or drops. On the TSE, an immediate execution only takes place if the next execution price is within a certain range from the previous execution price. The price level determines the range. For example, if the most recently executed price was 500 yen, the next execution price must be within the range of 490-510 yen. In other words, it can only fluctuate up to 10 yen in either direction.

Suppose the matched price is beyond this range—for example, 550 yen when the previous price was 500 yen. Then, execution does not take place; instead a special bid quote of 510 yen is indicated to call for offers at this price. If no offers at this price are received, the special bid quote will be raised to 520 yen after 5 minutes, and so on until equilibrium is achieved. This mechanism is intended to make a smooth transition between widely divergent prices.

But on the morning of 8 December, J-Com had no previous price. In such cases, the publicly assessed value is used in place of the previous price, which was 610,000 yen. Because the matched price was much higher, a special bid quote of 610,000 yen was shown at 9:00 a.m., then raised to 641,000 yen at 9:10 a.m., which means the range was ± 31,000 yen, and raised again to 672,000 yen at 9:20 a.m. Table 2 shows the order book at that moment, when the 1-yen sell offer came in.

Initial price determination

The term "reverse special quote" denotes this particularly rare event. It means that when a special buy bid quote is displayed, a sell order of low price with a significant amount that reverses the situation to a special sell offer quote comes in (or conversely a special sell offer quote is reversed to a special buy bid quote). TSE has another rule that applies to such a case. This rule stipulates that the previous special quote is fixed as the execution price, and the transaction proceeds. Thus, the initial price of J-Com was now determined to be 672,000 yen. In addition to the step price range set for reducing sudden price change, there is also a price limit range for a day. The upper and lower limits of the price for each stock are defined based on the initial price of the day. In the J-Com case, the limits were defined at the moment when the initial price was determined: The upper limit was 772,000 yen, and the lower limit was 572,000 yen.

In regular trading, the price limits are fixed at the start of the market day, and orders with prices exceeding the limit (either upper or lower) are rejected. But when the initial price is determined during the market time, as in the J-Com case, orders received before the price limits are set are not ignored. Instead, the price of an order exceeding the upper limit is adjusted to the upper price limit, and an order under the lower limit is adjusted to the lower price limit. Thus, the 1-yen order by Mizuho was adjusted to 572,000 yen.

Noticing the mistake, Mizuho entered a cancel command through a Fidessa terminal at 9:29:21, but it failed. Between 9:33:17 and 9:35:40, Mizuho tried to cancel the order several times through TSE system terminals that are installed at the Mizuho site, but the cancellations failed. Mizuho called TSE asking for a cancellation on the TSE side, but the answer was no.

At 9:35:33, Mizuho started to buy back J-Com shares. In the end, it could only buy back 510,000 shares; nearly 100,000 shares were bought by others and never restored.

Aftermath

On 12 December, four days after the incident, TSE president Takuo Tsurushima held a press conference and admitted that the order cancellation by Mizuho failed because of a defect in the TSE Stock Order System.

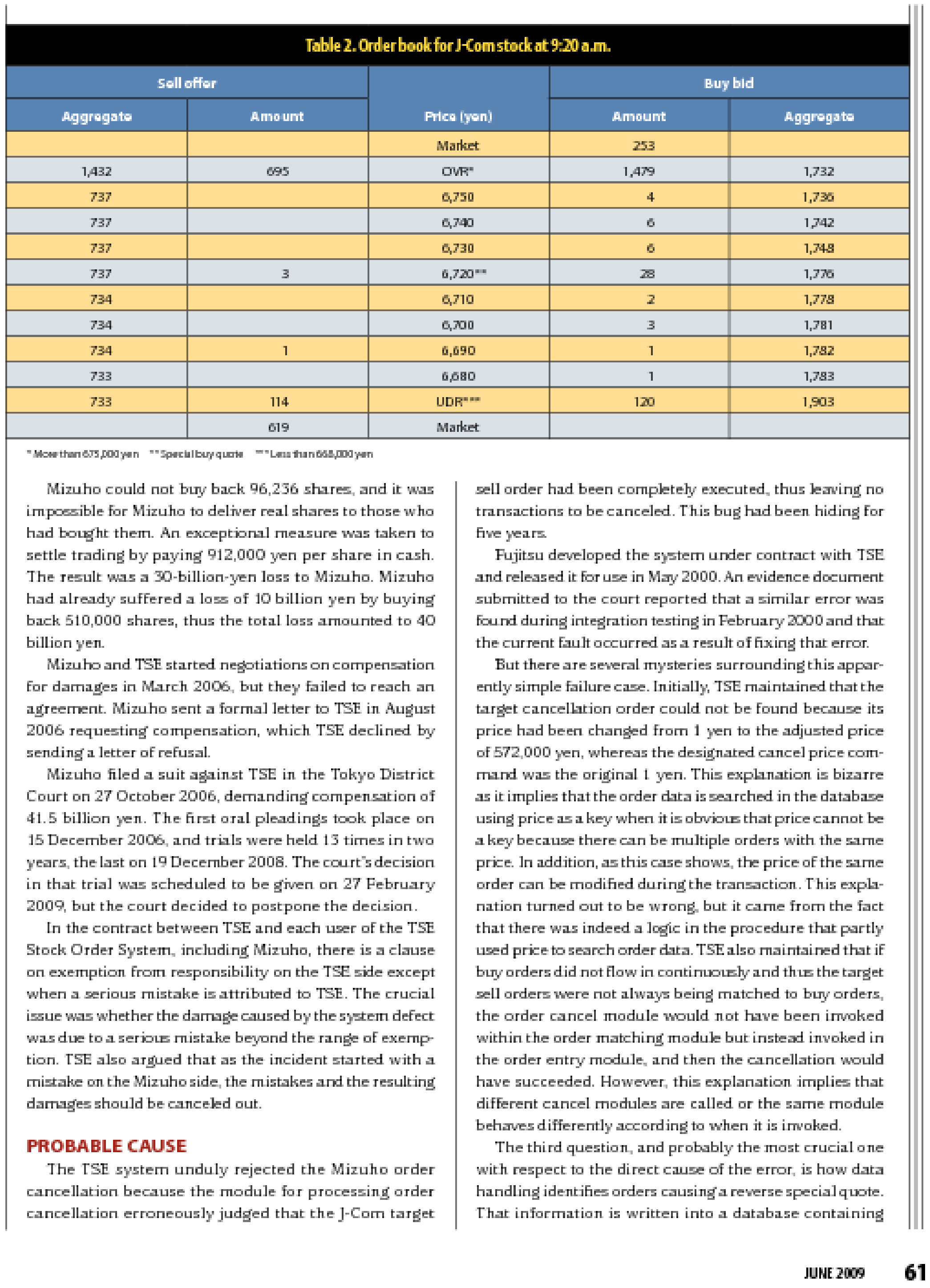


Table 2. Order book for J-Com stock at 9:20 a.m.

Sell offer			Buy bid	
Aggregate	Amount	Price (yen)	Amount	Aggregate
		Market	253	
1,432	695	OVR*	1,479	1,732
737		6,750	4	1,736
737		6,740	6	1,742
737		6,730	6	1,748
737	3	6,720**	28	1,776
734		6,710	2	1,778
734		6,700	3	1,781
734	1	6,690	1	1,782
733		6,680	1	1,783
733	114	UDR***	120	1,903
	619	Market		

* More than 675,000 yen ** Special buy quote *** Less than 668,000 yen

Mizuho could not buy back 96,236 shares, and it was impossible for Mizuho to deliver real shares to those who had bought them. An exceptional measure was taken to settle trading by paying 912,000 yen per share in cash. The result was a 30-billion-yen loss to Mizuho. Mizuho had already suffered a loss of 10 billion yen by buying back 510,000 shares, thus the total loss amounted to 40 billion yen.

Mizuho and TSE started negotiations on compensation for damages in March 2006, but they failed to reach an agreement. Mizuho sent a formal letter to TSE in August 2006 requesting compensation, which TSE declined by sending a letter of refusal.

Mizuho filed a suit against TSE in the Tokyo District Court on 27 October 2006, demanding compensation of 41.5 billion yen. The first oral pleadings took place on 15 December 2006, and trials were held 13 times in two years, the last on 19 December 2008. The court's decision in that trial was scheduled to be given on 27 February 2009, but the court decided to postpone the decision.

In the contract between TSE and each user of the TSE Stock Order System, including Mizuho, there is a clause on exemption from responsibility on the TSE side except when a serious mistake is attributed to TSE. The crucial issue was whether the damage caused by the system defect was due to a serious mistake beyond the range of exemption. TSE also argued that as the incident started with a mistake on the Mizuho side, the mistakes and the resulting damages should be canceled out.

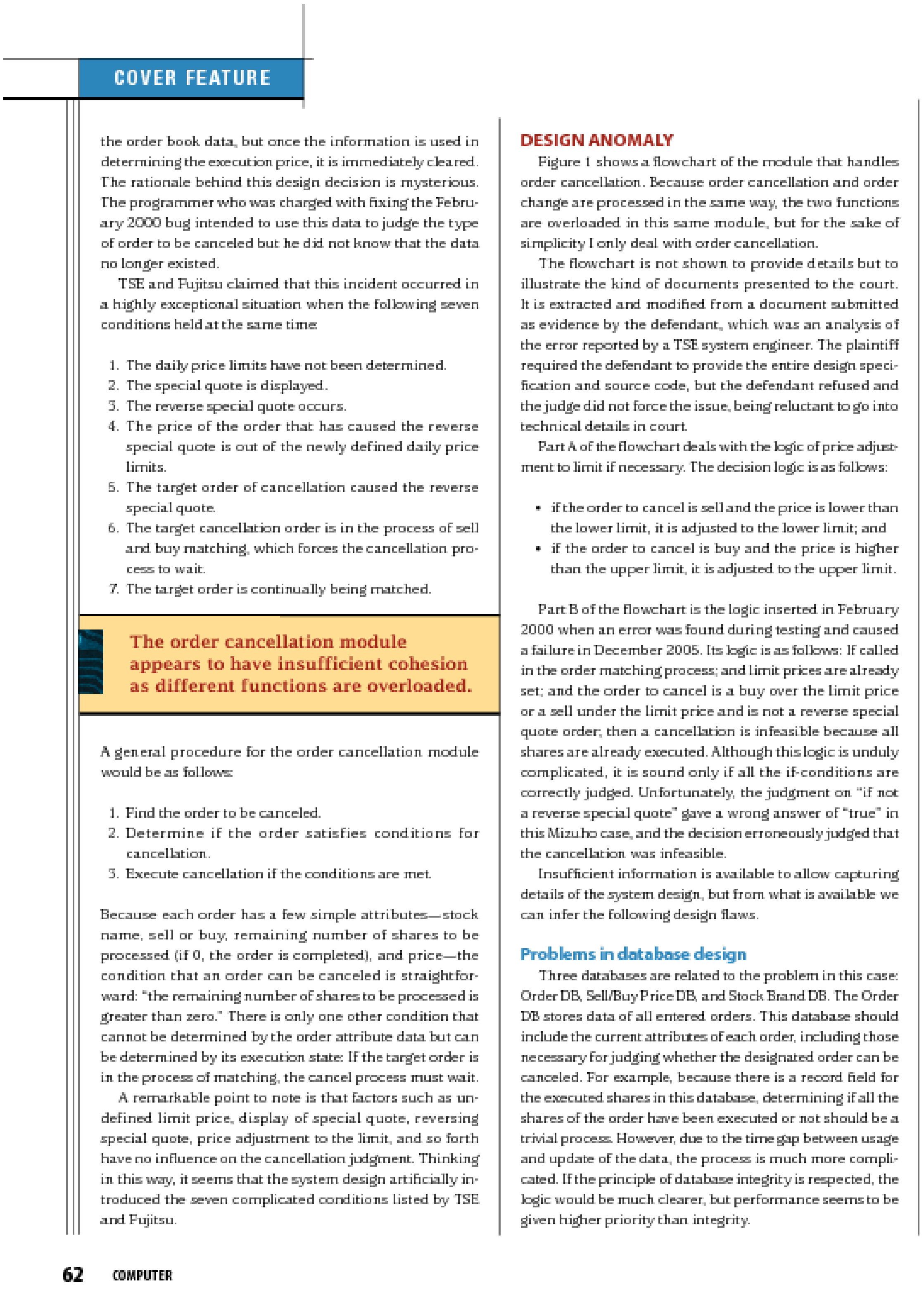
PROBABLE CAUSE

The TSE system unduly rejected the Mizuho order cancellation because the module for processing order cancellation erroneously judged that the J-Com target sell order had been completely executed, thus leaving no transactions to be canceled. This bug had been hiding for five years.

Fujitsu developed the system under contract with TSE and released it for use in May 2000. An evidence document submitted to the court reported that a similar error was found during integration testing in February 2000 and that the current fault occurred as a result of fixing that error.

But there are several mysteries surrounding this apparently simple failure case. Initially, TSE maintained that the target cancellation order could not be found because its price had been changed from 1 yen to the adjusted price of 572,000 yen, whereas the designated cancel price command was the original 1 yen. This explanation is bizarre as it implies that the order data is searched in the database using price as a key when it is obvious that price cannot be a key because there can be multiple orders with the same price. In addition, as this case shows, the price of the same order can be modified during the transaction. This explanation turned out to be wrong, but it came from the fact that there was indeed a logic in the procedure that partly used price to search order data. TSE also maintained that if buy orders did not flow in continuously and thus the target sell orders were not always being matched to buy orders, the order cancel module would not have been invoked within the order matching module but instead invoked in the order entry module, and then the cancellation would have succeeded. However, this explanation implies that different cancel modules are called or the same module behaves differently according to when it is invoked.

The third question, and probably the most crucial one with respect to the direct cause of the error, is how data handling identifies orders causing a reverse special quote. That information is written into a database containing



the order book data, but once the information is used in determining the execution price, it is immediately cleared. The rationale behind this design decision is mysterious. The programmer who was charged with fixing the February 2000 bug intended to use this data to judge the type of order to be canceled but he did not know that the data no longer existed.

TSE and Fujitsu claimed that this incident occurred in a highly exceptional situation when the following seven conditions held at the same time:

1. The daily price limits have not been determined.
2. The special quote is displayed.
3. The reverse special quote occurs.
4. The price of the order that has caused the reverse special quote is out of the newly defined daily price limits.
5. The target order of cancellation caused the reverse special quote.
6. The target cancellation order is in the process of sell and buy matching, which forces the cancellation process to wait.
7. The target order is continually being matched.

The order cancellation module appears to have insufficient cohesion as different functions are overloaded.

A general procedure for the order cancellation module would be as follows:

1. Find the order to be canceled.
2. Determine if the order satisfies conditions for cancellation.
3. Execute cancellation if the conditions are met.

Because each order has a few simple attributes—stock name, sell or buy, remaining number of shares to be processed (if 0, the order is completed), and price—the condition that an order can be canceled is straightforward: "the remaining number of shares to be processed is greater than zero." There is only one other condition that cannot be determined by the order attribute data but can be determined by its execution state: If the target order is in the process of matching, the cancel process must wait.

A remarkable point to note is that factors such as undefined limit price, display of special quote, reversing special quote, price adjustment to the limit, and so forth have no influence on the cancellation judgment. Thinking in this way, it seems that the system design artificially introduced the seven complicated conditions listed by TSE and Fujitsu.

DESIGN ANOMALY

Figure 1 shows a flowchart of the module that handles order cancellation. Because order cancellation and order change are processed in the same way, the two functions are overloaded in this same module, but for the sake of simplicity I only deal with order cancellation.

The flowchart is not shown to provide details but to illustrate the kind of documents presented to the court. It is extracted and modified from a document submitted as evidence by the defendant, which was an analysis of the error reported by a TSE system engineer. The plaintiff required the defendant to provide the entire design specification and source code, but the defendant refused and the judge did not force the issue, being reluctant to go into technical details in court.

Part A of the flowchart deals with the logic of price adjustment to limit if necessary. The decision logic is as follows:

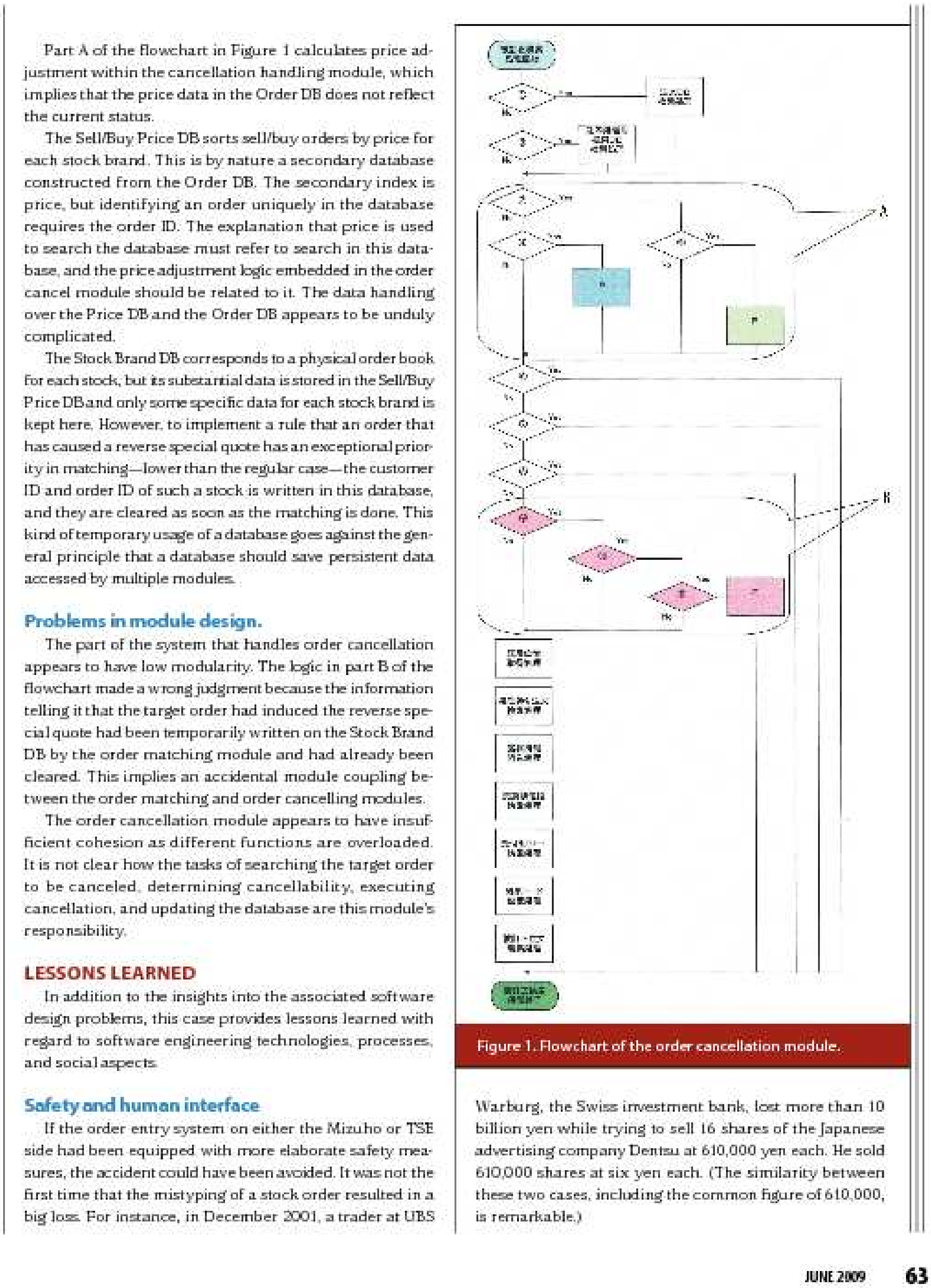
- if the order to cancel is sell and the price is lower than the lower limit, it is adjusted to the lower limit; and
- if the order to cancel is buy and the price is higher than the upper limit, it is adjusted to the upper limit.

Part B of the flowchart is the logic inserted in February 2000 when an error was found during testing and caused a failure in December 2005. Its logic is as follows: If called in the order matching process; and limit prices are already set; and the order to cancel is a buy over the limit price or a sell under the limit price and is not a reverse special quote order; then a cancellation is infeasible because all shares are already executed. Although this logic is unduly complicated, it is sound only if all the if-conditions are correctly judged. Unfortunately, the judgment on "if not a reverse special quote" gave a wrong answer of "true" in this Mizuho case, and the decision erroneously judged that the cancellation was infeasible.

Insufficient information is available to allow capturing details of the system design, but from what is available we can infer the following design flaws.

Problems in database design

Three databases are related to the problem in this case: Order DB, Sell/Buy Price DB, and Stock Brand DB. The Order DB stores data of all entered orders. This database should include the current attributes of each order, including those necessary for judging whether the designated order can be canceled. For example, because there is a record field for the executed shares in this database, determining if all the shares of the order have been executed or not should be a trivial process. However, due to the time gap between usage and update of the data, the process is much more complicated. If the principle of database integrity is respected, the logic would be much clearer, but performance seems to be given higher priority than integrity.



Part A of the flowchart in Figure 1 calculates price adjustment within the cancellation handling module, which implies that the price data in the Order DB does not reflect the current status.

The Sell/Buy Price DB sorts sell/buy orders by price for each stock brand. This is by nature a secondary database constructed from the Order DB. The secondary index is price, but identifying an order uniquely in the database requires the order ID. The explanation that price is used to search the database must refer to search in this database, and the price adjustment logic embedded in the order cancel module should be related to it. The data handling over the Price DB and the Order DB appears to be unduly complicated.

The Stock Brand DB corresponds to a physical order book for each stock, but its substantial data is stored in the Sell/Buy Price DBand only some specific data for each stock brand is kept here. However, to implement a rule that an order that has caused a reverse special quote has an exceptional priority in matching—lower than the regular case—the customer ID and order ID of such a stock is written in this database, and they are cleared as soon as the matching is done. This kind of temporary usage of a database goes against the general principle that a database should save persistent data accessed by multiple modules.

Problems in module design.

The part of the system that handles order cancellation appears to have low modularity. The logic in part B of the flowchart made a wrong judgment because the information telling it that the target order had induced the reverse special quote had been temporarily written on the Stock Brand DB by the order matching module and had already been cleared. This implies an accidental module coupling between the order matching and order cancelling modules.

The order cancellation module appears to have insufficient cohesion as different functions are overloaded. It is not clear how the tasks of searching the target order to be canceled, determining cancellability, executing cancellation, and updating the database are this module's responsibility.

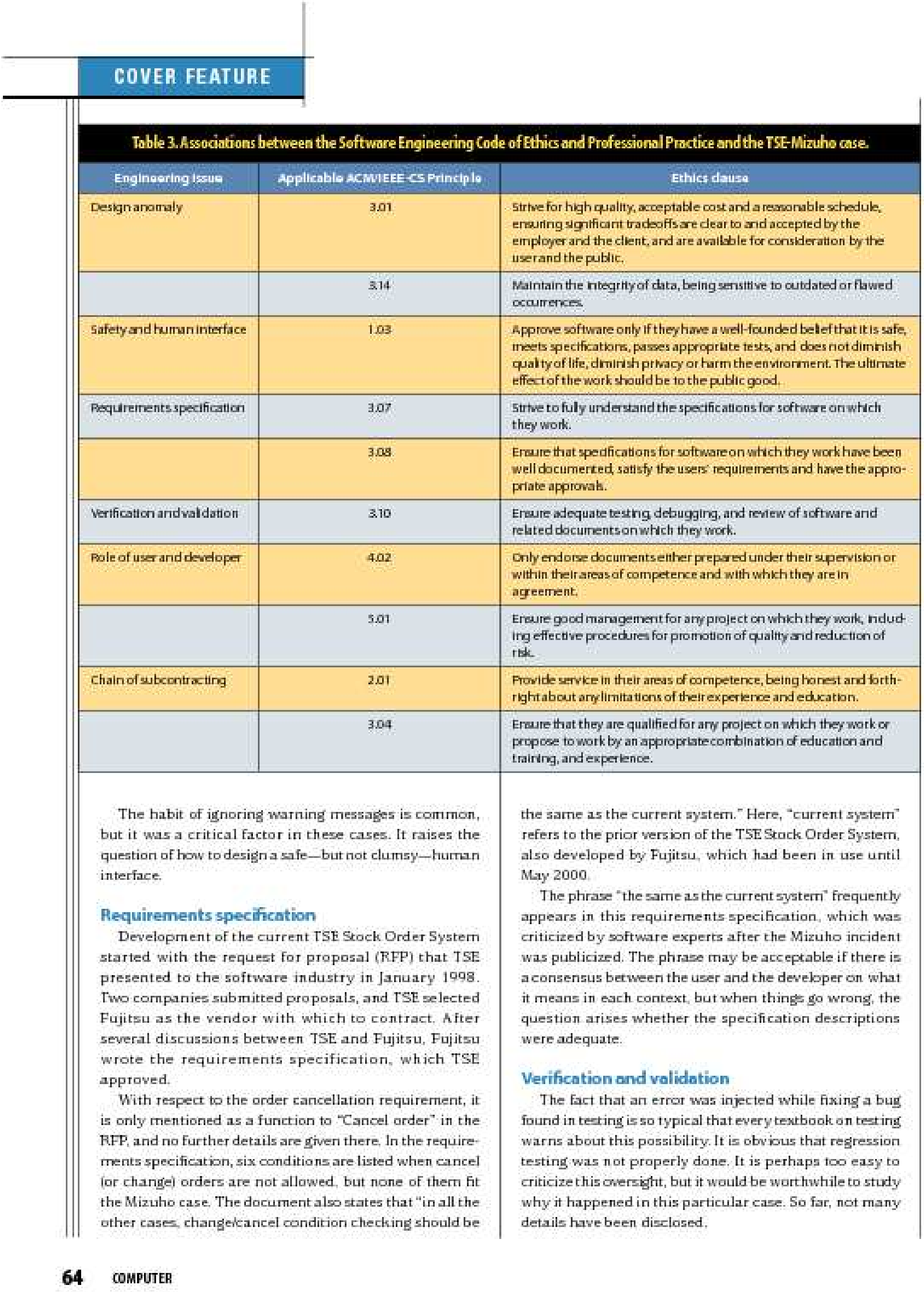
LESSONS LEARNED

In addition to the insights into the associated software design problems, this case provides lessons learned with regard to software engineering technologies, processes, and social aspects.

Safety and human interface

If the order entry system on either the Mizuho or TSE side had been equipped with more elaborate safety measures, the accident could have been avoided. It was not the first time that the mistyping of a stock order resulted in a big loss. For instance, in December 2001, a trader at UBS Warburg, the Swiss investment bank, lost more than 10 billion yen while trying to sell 16 shares of the Japanese advertising company Dentsu at 610,000 yen each. He sold 610,000 shares at six yen each. (The similarity between these two cases, including the common figure of 610,000, is remarkable.)

Figure 1. Flowchart of the order cancellation module.



COVER FEATURE

Table 3. Associations between the Software Engineering Code of Ethics and Professional Practice and the TSE-Mizuho case.

Engineering issue	Applicable ACM/IEEE-CS Principle	Ethics clause
Design anomaly	3.01	Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.
	3.14	Maintain the integrity of data, being sensitive to outdated or flawed occurrences.
Safety and human interface	1.03	Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.
Requirements specification	3.07	Strive to fully understand the specifications for software on which they work.
	3.08	Ensure that specifications for software on which they work have been well documented, satisfy the users' requirements and have the appropriate approvals.
Verification and validation	3.10	Ensure adequate testing, debugging, and review of software and related documents on which they work.
Role of user and developer	4.02	Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.
	5.01	Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.
Chain of subcontracting	2.01	Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education.
	3.04	Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.

The habit of ignoring warning messages is common, but it was a critical factor in these cases. It raises the question of how to design a safe—but not clumsy—human interface.

Requirements specification

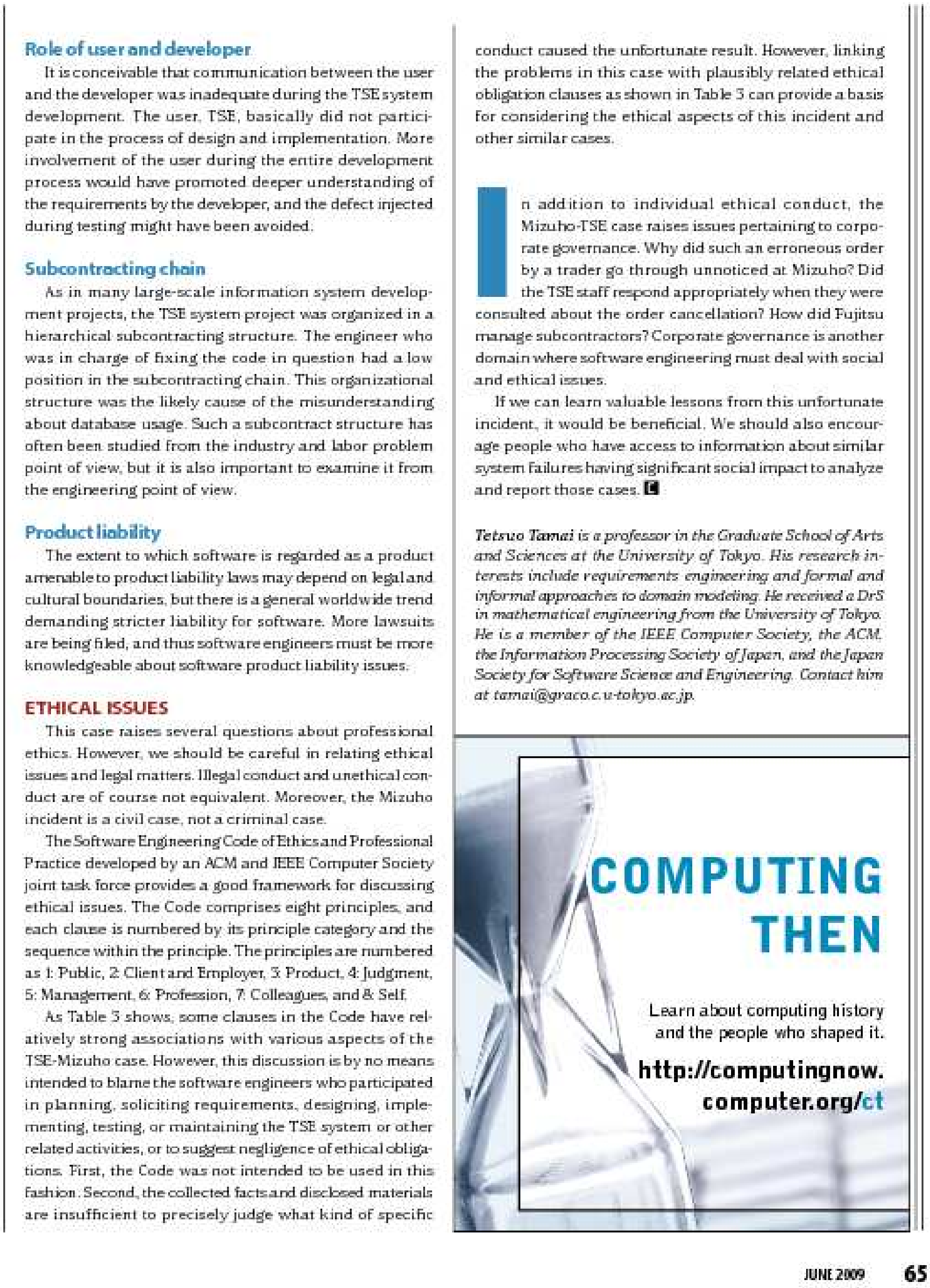
Development of the current TSE Stock Order System started with the request for proposal (RFP) that TSE presented to the software industry in January 1998. Two companies submitted proposals, and TSE selected Fujitsu as the vendor with which to contract. After several discussions between TSE and Fujitsu, Fujitsu wrote the requirements specification, which TSE approved.

With respect to the order cancellation requirement, it is only mentioned as a function to "Cancel order" in the RFP, and no further details are given there. In the requirements specification, six conditions are listed when cancel (or change) orders are not allowed, but none of them fit the Mizuho case. The document also states that "in all the other cases, change/cancel condition checking should be the same as the current system." Here, "current system" refers to the prior version of the TSE Stock Order System, also developed by Fujitsu, which had been in use until May 2000.

The phrase "the same as the current system" frequently appears in this requirements specification, which was criticized by software experts after the Mizuho incident was publicized. The phrase may be acceptable if there is a consensus between the user and the developer on what it means in each context, but when things go wrong, the question arises whether the specification descriptions were adequate.

Verification and validation

The fact that an error was injected while fixing a bug found in testing is so typical that every textbook on testing warns about this possibility. It is obvious that regression testing was not properly done. It is perhaps too easy to criticize this oversight, but it would be worthwhile to study why it happened in this particular case. So far, not many details have been disclosed.



Role of user and developer

It is conceivable that communication between the user and the developer was inadequate during the TSE system development. The user, TSE, basically did not participate in the process of design and implementation. More involvement of the user during the entire development process would have promoted deeper understanding of the requirements by the developer, and the defect injected during testing might have been avoided.

Subcontracting chain

As in many large-scale information system development projects, the TSE system project was organized in a hierarchical subcontracting structure. The engineer who was in charge of fixing the code in question had a low position in the subcontracting chain. This organizational structure was the likely cause of the misunderstanding about database usage. Such a subcontract structure has often been studied from the industry and labor problem point of view, but it is also important to examine it from the engineering point of view.

Product liability

The extent to which software is regarded as a product amenable to product liability laws may depend on legal and cultural boundaries, but there is a general worldwide trend demanding stricter liability for software. More lawsuits are being filed, and thus software engineers must be more knowledgeable about software product liability issues.

ETHICAL ISSUES

This case raises several questions about professional ethics. However, we should be careful in relating ethical issues and legal matters. Illegal conduct and unethical conduct are of course not equivalent. Moreover, the Mizuho incident is a civil case, not a criminal case.

The Software Engineering Code of Ethics and Professional Practice developed by an ACM and IEEE Computer Society joint task force provides a good framework for discussing ethical issues. The Code comprises eight principles, and each clause is numbered by its principle category and the sequence within the principle. The principles are numbered as 1: Public, 2: Client and Employer, 3: Product, 4: Judgment, 5: Management, 6: Profession, 7: Colleagues, and 8: Self.

As Table 3 shows, some clauses in the Code have relatively strong associations with various aspects of the TSE-Mizuho case. However, this discussion is by no means intended to blame the software engineers who participated in planning, soliciting requirements, designing, implementing, testing, or maintaining the TSE system or other related activities, or to suggest negligence of ethical obligations. First, the Code was not intended to be used in this fashion. Second, the collected facts and disclosed materials are insufficient to precisely judge what kind of specific conduct caused the unfortunate result. However, linking the problems in this case with plausibly related ethical obligation clauses as shown in Table 3 can provide a basis for considering the ethical aspects of this incident and other similar cases.

In addition to individual ethical conduct, the Mizuho-TSE case raises issues pertaining to corporate governance. Why did such an erroneous order by a trader go through unnoticed at Mizuho? Did the TSE staff respond appropriately when they were consulted about the order cancellation? How did Fujitsu manage subcontractors? Corporate governance is another domain where software engineering must deal with social and ethical issues.

If we can learn valuable lessons from this unfortunate incident, it would be beneficial. We should also encourage people who have access to information about similar system failures having significant social impact to analyze and report those cases. C

Tetsuo Tamai is a professor in the Graduate School of Arts and Sciences at the University of Tokyo. His research interests include requirements engineering and formal and informal approaches to domain modeling. He received a DrS in mathematical engineering from the University of Tokyo. He is a member of the IEEE Computer Society, the ACM, the Information Processing Society of Japan, and the Japan Society for Software Science and Engineering. Contact him at tamai@graco.c.u-tokyo.ac.jp.

Appendix D

Indexes

D.1 RSL Index

Literals , 162–174
η, 176
false, 150
true, 150
RSL-Text, 176
⌢, 176
$=$, 176
Unit, 174
chaos, 162, 164, 165
false, 155
true, 155

Arithmetic Constructs, 157
a_i*a_j, 157
a_i+a_j, 157
a_i/a_j, 157
$a_i=a_j$, 157
$a_i \geq a_j$, 157
$a_i > a_j$, 157
$a_i \leq a_j$, 157
$a_i < a_j$, 157
$a_i \neq a_j$, 157
$a_i - a_j$, 157
□, 155
$\Rightarrow$, 155
$=$, 155
$\neq$, 155
$\sim$, 155
$\vee$, 155
$\wedge$, 155

Cartesian Constructs, 158, 162–163
$(e_1,e_2,...,e_n)$, 158

Combinators, 170–173
... **elsif** ... , 172
case b_e **of** $pa_1 \to c_1$, ... $pa_n \to c_n$ **end** , 172
if b_e **then** c_c **else** c_a **end** , 172
let a:A • P(a) **in** c **end** , 171
let pa = e **in** c **end** , 170

Function Constructs, 169–170
post P(args,result), 170
pre P(args), 170
f(args) **as** result, 170
f(a), 168
f(args) $\equiv$ expr, 170

List Constructs, 158–159, 163–165
<Q(l(i))|i **in**<1..**len**l> •P(a)> , 159
<> , 158
ℓ(i) , 163
$\ell' = \ell''$, 163
$\ell' \neq \ell''$, 163
$\ell'^\frown\ell''$, 163
elems ℓ , 163
hd ℓ , 163
inds ℓ , 163
len ℓ , 163
tl ℓ , 163
e_1 $<e_2,e_2,...,e_n>$, 158

Logic Constructs, 154–157

$b_i \vee \mathrm{b}_j$, 155
$\forall$ a:A • P(a) , 156
$\exists$! a:A • P(a) , 156
$\exists$ a:A • P(a) , 156
$\sim$ b , 155
false, 150
true, 150
false, 155
true, 155
$\mathrm{b}_i \Rightarrow \mathrm{b}_j$, 155
$\mathrm{b}_i \wedge \mathrm{b}_j$, 155

Map Constructs, 159–160, 165–167
$\mathrm{m}_i \setminus \mathrm{m}_j$, 165, 166
$\mathrm{m}_i \circ \mathrm{m}_j$, 165, 166
$\mathrm{m}_i / \mathrm{m}_j$, 165, 166
dom m , 165, 166
rng m , 165, 166
$\mathrm{m}_i \dagger \mathrm{m}_j$, 165, 166
$\mathrm{m}_i = \mathrm{m}j$, 165
$\mathrm{m}_i \cup \mathrm{m}_j$, 165, 166
$\mathrm{m}_i \neq \mathrm{m}j$, 165
m(e) , 165, 166
[ ] , 159
$[\mathrm{u}_1 \mapsto \mathrm{v}_1,\mathrm{u}_2 \mapsto \mathrm{v}_2,...,\mathrm{u}_n \mapsto \mathrm{v}_n]$, 159
[F(e)$\mapsto$G(m(e))|e:E•e$\in$**dom** m$\wedge$P(e)] , 160

Process Constructs, 173–174
channel c:T , 173
channel {k[i]:T•i:Idx} , 173
c ! e , 174
c ? , 174
k[i] ! e , 174
k[i] ? , 174
$\mathrm{p}_i \,\square\, p_j$, 173
$\mathrm{p}_i \,\sqcap\, p_j$, 173
$\mathrm{p}_i \| p_j$, 173
$\mathrm{p}_i \dashv\!\vert p_j$, 174
P: **Unit** → **in** c **out** k[i] **Unit** , 174
Q: i:KIdx → **out** c **in** k[i] **Unit**, 174

Set Constructs, 157–158, 160–162
$\cap\{\mathrm{s}_1,\mathrm{s}_2,...,\mathrm{s}_n\}$, 160
$\cup\{\mathrm{s}_1,\mathrm{s}_2,...,\mathrm{s}_n\}$, 160
card s , 160
e$\in$s , 160
e$\notin$s , 160
$\mathrm{s}_i = \mathrm{s}_j$, 160
$\mathrm{s}_i \cap \mathrm{s}_j$, 160
$\mathrm{s}_i \cup \mathrm{s}_j$, 160
$\mathrm{s}_i \subset \mathrm{s}_j$, 160
$\mathrm{s}_i \subseteq \mathrm{s}_j$, 160
$\mathrm{s}_i \neq \mathrm{s}_j$, 160
$\mathrm{s}_i \setminus \mathrm{s}_j$, 160
{} , 157
$\{e_1, e_2, ..., e_n\}$, 157
{Q(a)|a:A•a$\in$s$\wedge$P(a)} , 158

Type Expressions, 150, 152
$(\mathrm{T}_1 \times \mathrm{T}_2 \times ... \times \mathrm{T}_n)$, 151
Bool, 150
Char, 150
Int, 150
Nat, 150
Real, 150
Text, 150
mk_ id(s_1:T_1,s_2:T_2,...,s_n:T_n) , 151
s_1:T_1 s_2:T_2 ... s_n:T_n , 151
T^* , 151
T^ω , 151
$\mathrm{T}_1 \times \mathrm{T}_2 \times ... \times \mathrm{T}_n$, 151
$\mathrm{T}_1 \mid \mathrm{T}_2 \mid ... \mid \mathrm{T}_1 \mid \mathrm{T}_n$, 151
$\mathrm{T}_i \,\underset{m}{\rightarrow}\, \mathrm{T}_j$, 151
$\mathrm{T}_i \overset{\sim}{\rightarrow} \mathrm{T}_j$, 151
$\mathrm{T}_i \rightarrow \mathrm{T}_j$, 151
T-infset, 151
T-set, 151

Type Definitions, 152–154
T = Type_ Expr, 152
T={| v:T′• P(v)|} , 153, 154
T==TE_1 | TE_2 | ... | TE_n , 153

Space & Time, 177–178
CURVE, 178
LOCATION, 177
POINT, 177, 178
SPACE, 177
SURFACE, 178
TIME, 178
TI, 178
VOLUME, 178
AREA $\mathbf{m}^2$, 178

LENGTH **m**, 178
VOLUME $\mathbf{m}^3$, 178
*, 178
+, 178
-, 178
record_$\mathbb{LOCATION}$, 177
record_$\mathbb{TIME}$, 178

D.2 A Domain Modeling Index

General

Domain Modeling Method: By a systematic domain analysis & description method we mean a set of *principles*, *procedures*, *techniques* and *tools*, for efficiently *analyzing & describing* domains., 185

Domain: By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants*., 187

Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster], 190

Entity: By an entity By an *entity* we shall understand a more-or-less rationally describable phenomenon., 194

Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time*, 191

Phenomena: By a *phenomenon* we shall understand a fact that is observed to exist or happen., 194

Prompt: By a prompt we shall understand an informal "advice" to the domain analyzer to "perform" a mental inquiry wrt. the real-life domain being studied., 194

Rationality: The rational, analytic philosophy issues of the inevitability of these external and internal qualities is this: (i) can they be justified as inevitable, and (ii) can they be suitably "separated", i.e., both disjoint and exhaustive ? Or are they merely of empirical nature ? The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not 'either or'], 196

Transcendence: By transcendence we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience, 212

Transcendental Deduction: By a transcendental deduction we shall understand the notion: a "conversion" of one kind of knowledge into a seemingly different kind of knowledge, 212

$\mathbb{T}$: the name of the type of all type names., 200

Endurants

External Qualities

Atomic Part: y an *atomic part* we shall understand a part which the domain analyzer considers to be indivisible in the sense of not meaningfully consist of sub-parts., 199

Cartesian and Part Sets: A description prompt, 202

Cartesians: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids). , 200

Compound Part: Compound parts are those which are observed to [potentially] consist of several parts, 199

Domain Description Schema. *Cartesian and Part Sets: ...*, 202

Domain Description Schema. *Describe Attributes: ...*, 208

Domain Description Schema. *Describe Unique Identity: ...*, 205

Domain Description Schema: *Describe Mereology: ...*, 207

External Quality: External qualities of endurants of a manifest domain are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form. , 196

Fluid Endurant: By a *fluid endurant* we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves; , 198

Part Sets: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts, 201

Part: Non-living solid species are what we shall call parts., 198

Solid Endurant: By a *solid* cum *discrete* endurant we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing, have body (or magnitude) of three-dimensions: length (or height), breadth and depth, 197

State: By a *state* we shall mean any subset of the parts of a domain., 203

is_ Cartesian: An analysis prompt, 200

is_ atomic: An analysis prompt, 199

is_ compound: An analysis prompt, 199

is_ fluid: An analysis prompt, 198

is_ part: An analysis prompt, 198

is_ part_ set: An analysis prompt, 201

is_ solid: An analysis prompt, 197

record_ Cartesian_ part_ type_ names: An analysis function, 201

record_ part_ set_ part_ type_ names: An analysis function, 201

Internal Qualities

Internal Quality: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about. , 196

Internal Qualities: **Unique Identification**

Domain Description Schema. *Describe Unique Identity: ...*, 205

Unique Identity: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids. , 205

uid_ : unique identifier observer., 205

Internal Qualities: **Mereology**

Domain Description Schema: *Describe Mereology: ...*, 207

Mereology: Mereology is a theory of [endurant] part-hood relations: of the relations of an [endurant] parts to a whole and the relations of [endurant] parts to [endurant] parts within that whole. , 206

mereo_ : mereology observer., 207

Internal Qualities: **Attributes**
- **Attribute:** Attributes are properties of endurants that can be measured either physically (by means of length (ruler) and spatial quantity measuring equipment, electronically, chemically, or otherwise) or can be objectively spoken about. , 207
- **Domain Description Schema.** *Describe Attributes:* ..., 208
- **is_ active:** An attribute category observer, 209
- **is_ autonomous:** An attribute category observer, 209
- **is_ biddable:** An attribute category observer, 209
- **is_ dynamic:** An attribute category observer, 209
- **is_ inert:** An attribute category observer, 209
- **is_ monitorable_ attribute:** An attribute category observer, 210
- **is_ programmable:** An attribute category observer, 209
- **is_ programmable_ attribute:** An attribute category observer, 210
- **is_ static:** An attribute category observer, 209
- **is_ static_ attribute:** An attribute category observer, 210
- **record_ attribute_ type_ names:** An analysis function, 208
- **attr_** : attribute observer, 208

Perdurants
- **Action:** An action is a function that can purposefully change a state, 213
- **Behaviour:** Behaviours are sets of sequences of actions, events and behaviours, 213
- **Channel:** A channel is anything that allows synchronization and communication of values between behaviours, 213
- **Description Schema: Behaviour Invocation:** ..., 219
- **Description Schema: Behaviour Signatures:** ..., 214
- **Description Schema: Channels: channel** ... , 214
- **Events:** An event is a function that surreptitiously changes a state, 213

D.3 Banking Domain Concepts

This index is far rom satifactory. I will, at some time, make sure that it eventually becomes one !

bank
- branch office, 14
- central, 14
- credit/debit companies, 14
- head-quarter, 14
- HQ, 14
- “mortar & brick’, 31
- mortgage, savings & loan, 14
- national, 14
- regional, 14

banking, 13
- structure, 14

branch office, 14

central bank, 14
command, 37
credit/debit companies, 14
Currency, 14
currency, 14
customer, 14, 31

data vetting, 23
domain
- facet, 37

entity
- syntactic, 37

facet
  of domain, 37
  script, 37

head-quarter, 14

IMF, 14
infrastructure
  component, 5
International Monetary Fund, 14

“mortar & brick” bank, 31
mortgage, savings & loan banks, 14

national bank, 14

regional bank, 14

script facet, 37
state
  change, 37
stock
  brokers, 14
  exchanges, 14
stocks, 14
syntactic entity, 37

tellers, 14

D.4 Formal Entities

The formal entries first lists formula entries by ordinological category, then all:

Endurants

  External Qualities

  * **Parts: Sorts ad Observers**
  * **A Part State Concept**

  Internal Qualities

    Unique Identification

    * **Unique Identifiers: Sorts and Observers**
    * **A Unique Identifier State Concept**
    * **A Wellformedness Axiom**

    Mereology

    * **Mereology: Sorts and Observers**
    * **A Wellformedness Axiom**

    Attributes

    * **Attributes: Sorts and Observers**
    * **Wellformedness Axioms**

  * **Intentional Pull**

* **Commands**

Perdurants

* **Communication**
* **Messages**
* **Behaviour Signatures**
* **Behaviour Definitions**
* **Initialization**

* **Values**

- * **Auxiliary Types**
- * **Auxiliary Functions**
- * **Theorems**

Only the *'ed entries are listed.

Endurant
sorts
ι39, 30
B ι37 π30, 30
B ι5 π16, 16
BA ι4 π16, 16
BOA ι7 π16, 16
BOS ι8 π16, 16
BS ι5 π16, 16
C ι11 π16, 16
C ι41 π30, 30
CA ι9 π16, 16
CS ι10 π16, 16
HQ ι6 π16, 16
WBS ι3 π16, 16
WS ι34 π29, 30
observers
obs_BA ι4 π16, 16
obs_BOA ι7 π16, 16
obs_BOS ι8 π16, 16
obs_BS ι5 π16, 16
obs_CA ι38, 30
obs_CA ι9 π16, 16
obs_CS ι10 π16, 16
obs_CS ι39, 30
obs_HQ ι6 π16, 16

Unique Identification
sorts
BAI ι20b π19, 19
BI ι20d π19, 19
BOA ι20f π19, 19
BOI ι20h π19, 19
BOSI ι20g π19, 19
BSI ι20c π19, 19
CAI ι20i π19, 19
CI ι20k π19, 19
CI ι43 π30, 30
CSI ι20j π19, 19
HQI ι20e π19, 19
WBSI ι20a π19, 19
observers
uid_ BA ι20b π19, 19
uid_ BOA ι20f π19, 19
uid_ BOS ι20g π19, 19
uid_ BO ι20h π19, 19
uid_ BS ι20c π19, 19
uid_ B ι20d π19, 19
uid_ CA ι20i π19, 19
uid_ CS ι20j π19, 19
uid_ C ι20k π19, 19
uid_ C ι43 π30, 30
uid_ HQ ι20e π19, 19
uid_ WBS ι20a π19, 19

Axioms
Consistent Bank Identification ι25 π23, 23

Mereology
types
BM ι28 π24, 24
C ι44 π32, 32
DOM ι30 π24, 24
HQM ι29 π24, 24
observers
mereo_ BO ι30 π24, 24
mereo_ B ι28 π24, 24
mereo_ C ι44 π32, 32
mereo_ HQ ι29 π24, 24

Attributes
Accts ι152 π101, 101
BHist ι118 π72, 72
BOHist ι153 π101, 101
CashCat ι117a π72, 72
Cashs ι117 π72, 72
CustAcct ι113a π72, 72
CustAccts ι113 π72, 72
CustCtlg ι150 π101, 101
RegNu ι149 π101, 101

Attribute
types:
AccId ι150 π101, 101
AccId ι152a π101, 101
Accounts ι55 π33, 34
AcctInfo ι152b π101, 101
AdminInfo ι45 π33, 34
Assets ι47 π33, 34
Balance ι152c π101, 101
Cards ι56 π34, 34
Displays ι57 π34, 34
PosR ι152d π101, 101
Trans ι152f π101, 101
Transl ι152e π101, 101
observers:
attr_ Accounts ι114 π72, 72
attr_ Accounts ι55 π33, 34
attr_ AcctHist ι115 π72, 72
attr_ Accts ι152 π101, 101
attr_ AdmInfo ι112 π72, 72
attr_ AdminInfo ι45 π33, 34
attr_ Assets ι47 π33, 34
attr_ BOHist ι153 π101, 101
attr_ BankHist ι118 π72, 73
attr_ Banks ι54 π33, 34
attr_ Cards ι56 π34, 34
attr_ Cashs ι117 π72, 72
attr_ CustAccts ι113 π72, 72
attr_ CustCtlg ι150 π101, 101
attr_ Displays ι57 π34, 34
attr_ Income ι46 π33, 34
attr_ Liabilities ι52 π33, 34
attr_ RegNu ι149 π101, 101
attr_ Taxes ι53 π33, 34
auxiliary types:
Accounts ι114 π72, 72
AcctHist ι115 π72, 72
AcctInfo ι58 π34, 34
AcctNu ι113b π72, 72
AcctNu ι55 π33, 34
Addresses ι45 π33, 34
Balance ι114a π72, 72
BankID ι55 π33, 34
Banks ι54 π33, 34
Birthdate ι45 π33, 34
Bonds ι47 π33, 34
Cash ι47 π33, 34
CDId ι56 π34, 34
Commodities ι47 π33, 34
Creditors ι50 π33, 34
Debitors ι51 π33, 34
Dep ι116 π72, 72
DI ι51 π33, 34
Income ι46 π33, 34
KI ι50 π33, 34
Liabilities ι52 π33, 34
MaritaStatus ι45 π33, 34
Name ι45 π33, 34
Nationality ι45 π33, 34
RealEstate ι47 π33, 34
Stocks ι47 π33, 34
Taxes ι53 π33, 34
With ι116 π72, 72
Work ι45 π33, 34

Commands
syntax
AbortDisp ι122f π75, 76
AccChgAcctSta ι155g π103, 103
BCmd ι122, 76
BOCmd ι155 π103, 103
CloCDCard ι155c π103, 103
ClosAcct ι155i π103, 103
Cmd ι60, 37
DlvWith ι122c π75, 76
EffAmort ι122h π75, 76
EffClose ι122i π75, 76
EffDebCred ι122g π75, 76
EffectXfer ι122d π75, 76
InfoFromBank ι155h π103, 103
LoanAccDen ι155d π103, 103
LoanClos ι155f π103, 103
LoanIncrOK ι155e π103, 103
NewAcct ι155a π103, 103
NewAcctNu ι122a π75, 76
OpnCDCard ι155b π103, 103
PresDisp ι122e π75, 76
RcvDe ι122b π75, 76

Behaviour
Signatures
bank ι61 π37, 76

customer ι62 π38, 37

Definitions

apply_ loan_ bo ι175 π115, 115
bank_ amort ι147, 95
bank_ close_ disp ι142, 93
bank_ del_ with ι130, 83
bank_ eff_ xfer ι132, 85
bank_ open_ disp ι140, 92
bank_ rcv_ dep ι129, 81
bank ι61 π37, 76
buy_ stock_ bo ι177 π116, 118
change_ account_ bo ι187 π121, 121
close_ account_ bo ι173 π114, 114
close_ account_ bo ι189 π122, 122
close_ CD_ bo ι163 π109, 109
close_ deposit_ bo ι171 π113, 113
close_ display ι78, 47
close_ loan_ bo ι179 π117, 117
close_ paym_ bo ι167 π111, 111
credit_ debit ι82, 49
customer ι62 π38, 37
deposit ι67, 41
diff_ custs_ diff_ banks_ xfer ι138 π90, 90
diff_ custs_ same_ bank_ xfer ι136 π88, 88
exchange ι74, 45
handle_ cust_ reacts ι111, 64
incl_ excl_ deb_ cre_ bo ι185 π120, 120
increase_ loan_ bo ι177 π116, 116
info_ from_ bank ι192 π124, 124
nil_ xfer ι134 π86, 86
open_ account_ bo ι159 π106, 106
open_ account ι65, 40
open_ CD_ bo ι161 π108, 108
open_ deposit_ bo ι169 π112, 112
open_ display ι76, 46
open_ paym_ bo ι165 π110, 110
pro_ active_ bank ι148, 96
pro_ active_ customer ι62 π38, 38
re_ active_ bank ι125 π77, 77
re_ active_ branch_ office ι157 π105, 105
re_ active_ customer ι63 π38, 38
same_ cust_ diff_ banks_ xfer ι137 π89, 89
same_ cust_ same_ bank_ xfer ι135 π87, 87
sell_ stock_ bo ι177 π116, 119
transfer ι72, 44
withdraw ι69, 42

Behaviours

new_ acct ι127 π79, 79

Auxiliary

Types

AcctInfo ι113c π72, 72

Functions

xtr_ BI ι23b π22, 22
xtr_ BI ι24a π22, 22
xtr_ BOI_ HQI ι150 π101, 101
xtr_ BOIS ι22c π22, 22
xtr_ BOIS ι23c π22, 22
xtr_ HQI ι22b π22, 22
xtr_ HQI ι24b π22, 22
xtr_ NBI ι22a π22, 22
xtr_ NBI ι23a π22, 22
xtr_ UIs ι55 π33, 34

All

attr_ Accounts ι114 π72, 72
attr_ Accounts ι55 π33, 34
attr_ AcctHist ι115 π72, 72
attr_ Accts ι152 π101, 101
attr_ AdmInfo ι112 π72, 72
attr_ AdminInfo ι45 π33, 34
attr_ Assets ι47 π33, 34
attr_ BOHist ι153 π101, 101
attr_ BankHist ι118 π72, 73
attr_ Banks ι54 π33, 34
attr_ Cards ι56 π34, 34
attr_ Cashs ι117 π72, 72
attr_ CustAccts ι113 π72, 72
attr_ CustCtlg ι150 π101, 101
attr_ Displays ι57 π34, 34
attr_ Income ι46 π33, 34
attr_ Liabilities ι52 π33, 34
attr_ RegNu ι149 π101, 101
attr_ Taxes ι53 π33, 34
AccId ι150 π101, 101

AccId ι152a π101, 101
Accounts ι114 π72, 72
Accounts ι55 π33, 34
AcctHist ι115 π72, 72
AcctInfo ι113c π72, 72
AcctInfo ι152b π101, 101
AcctInfo ι58 π34, 34
AcctNu ι113b π72, 72
AcctNu ι55 π33, 34
Accts ι152 π101, 101
Addresses ι45 π33, 34
AdminInfo ι45 π33, 34
Assets ι47 π33, 34
BAI ι20b π19, 19
BA ι4 π16, 16
BHist ι118 π72, 72
BI ι20d π19, 19
BM ι28 π24, 24
BOA ι20f π19, 19
BOA ι7 π16, 16
BOHist ι153 π101, 101
BOI ι20h π19, 19
BOSI ι20g π19, 19
BOS ι8 π16, 16
BO ι30 π24, 24
BSI ι20c π19, 19
BS ι5 π16, 16
B ι28 π24, 24
B ι37 π30, 30
B ι5 π16, 16
Balance ι114a π72, 72
Balance ι152c π101, 101
BankID ι55 π33, 34
Banks ι54 π33, 34
Birthdate ι45 π33, 34
Bonds ι47 π33, 34
CAI ι20i π19, 19
CA ι9 π16, 16
CDId ι56 π34, 34
CI ι20k π19, 19
CI ι43 π30, 30
CSI ι20j π19, 19
CS ι10 π16, 16
C ι11 π16, 16
C ι41 π30, 30
C ι44 π32, 32
Cards ι56 π34, 34
CashCat ι117a π72, 72
Cash ι47 π33, 34
Cashs ι117 π72, 72
Commodities ι47 π33, 34
Consistent Bank Identification ι25 π23, 23
Creditors ι50 π33, 34
CustAcct ι113a π72, 72
CustAccts ι113 π72, 72
CustCtlg ι150 π101, 101
DI ι51 π33, 34
DOM ι30 π24, 24
Debitors ι51 π33, 34
Dep ι116 π72, 72
Displays ι57 π34, 34
HQI ι20e π19, 19
HQM ι29 π24, 24
HQ ι29 π24, 24
HQ ι6 π16, 16
Income ι46 π33, 34
KI ι50 π33, 34
Liabilities ι52 π33, 34
MaritaStatus ι45 π33, 34
Name ι45 π33, 34
Nationality ι45 π33, 34
PosR ι152d π101, 101
RealEstate ι47 π33, 34
RegNu ι149 π101, 101
Stocks ι47 π33, 34
Taxes ι53 π33, 34
Trans ι152f π101, 101
Transl ι152e π101, 101
WBSI ι20a π19, 19
WBS ι3 π16, 16
WS ι34 π29, 30
With ι116 π72, 72
Work ι45 π33, 34
ι39, 30
apply_ loan_ bo ι175 π115, 115
bank_ amort ι147, 95
bank_ close_ disp ι142, 93
bank_ del_ with ι130, 83
bank_ eff_ xfer ι132, 85
bank_ open_ disp ι140, 92
bank_ rcv_ dep ι129, 81

bank ι61 π37, 76
buy_ stock_ bo ι177 π116, 118
change_ account_ bo ι187 π121, 121
close_ CD_ bo ι163 π109, 109
close_ account_ bo ι173 π114, 114
close_ account_ bo ι189 π122, 122
close_ deposit_ bo ι171 π113, 113
close_ display ι78, 47
close_ loan_ bo ι179 π117, 117
close_ paym_ bo ι167 π111, 111
credit_ debit ι82, 49
customer ι62 π38, 37
deposit ι67, 41
diff_ custs_ diff_ banks_ xfer ι138 π90, 90
diff_ custs_ same_ bank_ xfer ι136 π88, 88
exchange ι74, 45
handle_ cust_ reacts ι111, 64
incl_ excl_ deb_ cre_ bo ι185 π120, 120
increase_ loan_ bo ι177 π116, 116
info_ from_ bank ι192 π124, 124
new_ acct ι127 π79, 79
nil_ xfer ι134 π86, 86
open_ CD_ bo ι161 π108, 108
open_ account_ bo ι159 π106, 106
open_ account ι65, 40
open_ deposit_ bo ι169 π112, 112
open_ display ι76, 46
open_ paym_ bo ι165 π110, 110
pro_ active_ bank ι148, 96
pro_ active_ customer ι62 π38, 38
re_ active_ bank ι125 π77, 77
re_ active_ branch_ office ι157 π105, 105
re_ active_ customer ι63 π38, 38
same_ cust_ diff_ banks_ xfer ι137 π89, 89
same_ cust_ same_ bank_ xfer ι135 π87, 87
sell_ stock_ bo ι177 π116, 119
transfer ι72, 44
withdraw ι69, 42
xtr_ BI ι23b π22, 22
xtr_ BI ι24a π22, 22
xtr_ BOIS ι22c π22, 22
xtr_ BOIS ι23c π22, 22
xtr_ BOI_ HQI ι150 π101, 101
xtr_ HQI ι22b π22, 22
xtr_ HQI ι24b π22, 22
xtr_ NBI ι22a π22, 22
xtr_ NBI ι23a π22, 22
xtr_ UIs ι55 π33, 34
obs_ BA ι4 π16, 16
obs_ BOA ι7 π16, 16
obs_ BOS ι8 π16, 16
obs_ BS ι5 π16, 16
obs_ CA ι38, 30
obs_ CA ι9 π16, 16
obs_ CS ι10 π16, 16
obs_ CS ι39, 30
obs_ HQ ι6 π16, 16
uid_ BA ι20b π19, 19
uid_ BOA ι20f π19, 19
uid_ BOS ι20g π19, 19
uid_ BO ι20h π19, 19
uid_ BS ι20c π19, 19
uid_ B ι20d π19, 19
uid_ CA ι20i π19, 19
uid_ CS ι20j π19, 19
uid_ C ι20k π19, 19
uid_ C ι43 π30, 30
uid_ HQ ι20e π19, 19
uid_ WBS ι20a π19, 19
AbortDisp ι122f π75, 76
AccChgAcctSta ι155g π103, 103
BCmd ι122, 76
BOCmd ι155 π103, 103
CloCDCard ι155c π103, 103
ClosAcct ι155i π103, 103
Cmd ι60, 37
DlvWith ι122c π75, 76
EffAmort ι122h π75, 76
EffClose ι122i π75, 76
EffDebCred ι122g π75, 76
EffectXfer ι122d π75, 76
InfoFromBank ι155h π103, 103
LoanAccDen ι155d π103, 103
LoanClos ι155f π103, 103
LoanIncrOK ι155e π103, 103
NewAcct ι155a π103, 103
NewAcctNu ι122a π75, 76

OpnCDCard ι155b π103, 103
PresDisp ι122e π75, 76
RcvDe ι122b π75, 76

There are 208 formal `RSL` entities, and there are 208 `RSL` definitions – the former counted among the latter.