A Retailer Market:
Domain Analysis & Description
A Comparison HERAKLIT/DS&E Case Study

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

January 27, 2021: 15:46

Abstract

We report an exercise in modeling a retail system such as outlined in both [1, Bjørner, 2002] and [26, Fettke & Reisig, Dec. 21, 2020]. In the present exercise we try follow [26, Fettke & Reisig] – but we do so slavishly following the *domain analysis & description* (DA&D) method of [16, Domain Science & Engineering, Chapters 3–6, Bjørner 2021][1] (DS&E).[2]

Contents

1 Two Approaches to Modeling **3**
1.1 Domain Science & Engineering: DS&E . . . 4
1.2 HERAKLIT: http://heraklit.dfki.de/ . . . 4

2 The retailer market Case Study **4**
2.1 Three Rough Sketches . . . 4
2.1.1 Identification of "Main Players" . . . 4
2.1.2 Main Transaction Sequences . . . 5
2.1.3 Detailed Sketch . . . 6
2.1.4 Transitions . . . 8

3 Endurants: External Qualities **8**
3.1 Main Decompositions . . . 8
3.1.0.1 Narrative . . . 8
3.1.0.2 Formalisation . . . 8
3.2 Aggregates as Sets . . . 9
3.2.0.1 Narrative . . . 9
3.2.0.2 Formalisation . . . 9
3.3 The Retailer . . . 9
3.3.1 The HERAKLIT View . . . 9
3.3.1.1 Narrative . . . 10
3.3.1.2 Formalisation . . . 10
3.3.2 The DS&E View . . . 10
3.4 The Market System State . . . 10
3.4.0.1 Narrative . . . 10
3.4.0.2 Formalisation . . . 11

[1] [16] is scheduled to be published by **Springer** in their *EATCS Monographs in Theoretical Computer Science* series, Winter/Spring of 2021. Till such a time you may fund an electronic copy at `www.imm.dtu.dk/~dibj/2020/mono/mono.pdf`. That electronic copy may, from time to time, be updated as I "improve" on its text.

[2] Work on this document started December 28, 2020.

4 Endurants: Internal Qualities **12**
4.1 Unique Identifiers . . . 12
4.1.0.1 Narrative . . . 12
4.1.0.2 Formalisation . . . 12
4.2 Mereology . . . 13
4.2.1 Customer Mereology . . . 13
4.2.1.1 Narrative . . . 13
4.2.1.2 Formalisation . . . 13
4.2.2 Order Management Mereology . . . 13
4.2.2.1 Narrative . . . 13
4.2.2.2 Formalisation . . . 14
4.2.3 Inventory Mereology . . . 14
4.2.3.1 Narrative . . . 14
4.2.3.2 Formalisation . . . 14
4.2.4 Warehouse Mereology . . . 14
4.2.4.1 Formalisation . . . 14
4.2.5 Supplier Mereology . . . 15
4.2.5.1 Narrative . . . 15
4.2.5.2 Formalisation . . . 15
4.2.6 Courier Service Mereology . . . 15
4.2.6.1 Narrative . . . 15
4.2.6.2 Formalisation . . . 15
4.3 Attributes . . . 15
4.3.1 Transactions . . . 15
4.3.1.1 Narrative . . . 16
4.3.1.2 Formalisation . . . 16
4.3.1.3 Narrative . . . 16
4.3.1.4 Formalisation . . . 16
4.3.2 Customer Attributes . . . 17
4.3.2.1 Narrative . . . 17
4.3.2.2 Formalisation . . . 17
4.3.2.3 Narrative . . . 17
4.3.2.4 Formalisation . . . 18
4.3.3 Order Management Attributes . . . 18
4.3.3.1 Narrative . . . 18
4.3.3.2 Formalisation . . . 18
4.3.3.3 Narrative . . . 18
4.3.3.4 Formalisation . . . 19
4.3.4 Inventory Attributes . . . 19
4.3.4.1 Narrative . . . 19
4.3.4.2 Formalisation . . . 19
4.3.4.3 Narrative . . . 19
4.3.4.4 Formalisation . . . 19
4.3.5 Warehouse Attributes . . . 20
4.3.5.1 Narrative . . . 20
4.3.5.2 Formalisation . . . 20
4.3.5.3 Narrative . . . 20
4.3.5.4 Formalisation . . . 20
4.3.6 Supplier Attributes . . . 20
4.3.6.1 Narrative . . . 20
4.3.6.2 Formalisation . . . 21
4.3.6.3 Narrative . . . 21
4.3.6.4 Formalisation . . . 21
4.3.7 Courier Attributes . . . 21
4.3.7.1 Narrative . . . 21
4.3.7.2 Formalisation . . . 21
4.3.7.3 Narrative . . . 21
4.3.7.4 Formalisation . . . 22

5 Merchandise **22**
5.1 "Unique Identity" . . . 22
5.2 "Mereology" . . . 22
5.3 "Attributes" . . . 22
5.4 Representation . . . 23

6 Perdurants **23**
6.1 Channels . . . 23
6.2 Behaviours . . . 23
6.2.1 Customer Behaviour . . . 23
6.2.1.1 Narrative . . . 23
6.2.1.2 Formalisation . . . 24
6.2.2 Order Management Behaviour . . . 25
6.2.2.1 Narrative . . . 25
6.2.2.2 Formalisation . . . 25
6.2.2.3 Narrative . . . 25
6.2.2.4 Formalisation . . . 25
6.2.2.5 Narrative . . . 26
6.2.2.6 Formalisation . . . 26
6.2.2.7 Narrative . . . 26
6.2.2.8 Formalisation . . . 26

6.2.2.9 Narrative . . . 27
6.2.2.10 Formalisation . . . 27
6.2.2.11 Narrative . . . 27
6.2.2.12 Formalisation . . . 27
6.2.3 Inventory Behaviour . . . 27
6.2.3.1 Narrative . . . 27
6.2.3.2 Formalisation . . . 28
6.2.3.3 Narrative . . . 28
6.2.3.4 Formalisation . . . 28
6.2.3.5 Narrative . . . 29
6.2.3.6 Formalisation . . . 29
6.2.3.7 Narrative . . . 29
6.2.3.8 Formalisation . . . 30
6.2.3.9 Narrative . . . 30
6.2.3.10 Formalisation . . . 30
6.2.3.11 Narrative . . . 31
6.2.3.12 Formalisation . . . 31
6.2.4 Warehouse Behaviour . . . 31
6.2.4.1 Narrative . . . 31
6.2.4.2 Formalisation . . . 32
6.2.4.3 Narrative . . . 32
6.2.4.4 Formalisation . . . 32
6.2.4.5 Narrative . . . 32
6.2.4.6 Formalisation . . . 33
6.2.4.7 Narrative . . . 33
6.2.4.8 Formalisation . . . 33
6.2.4.9 Narrative . . . 34
6.2.4.10 Formalisation . . . 34
6.2.4.11 Narrative . . . 34
6.2.4.12 Formalisation . . . 35
6.2.5 Supplier Behaviour . . . 35
6.2.5.1 Narrative . . . 35
6.2.5.2 Formalisation . . . 35
6.2.5.3 Narrative . . . 35
6.2.5.4 Formalisation . . . 36
6.2.5.5 Narrative . . . 36
6.2.5.6 Formalisation . . . 36
6.2.6 Courier Service Behaviour . . . 36
6.2.6.1 Narrative . . . 36
6.2.6.2 Formalisation . . . 37
6.2.6.3 Narrative . . . 37
6.2.6.4 Formalisation . . . 37
6.2.6.5 Formalisation . . . 37
6.3 System Initialisation . . . 38

7 Conclusion **38**
7.1 Critique of the DA&D Model . . . 38
7.2 Proofs about Models . . . 39
7.3 Comparison of Models . . . 39
7.3.1 "Minor" Discrepancies . . . 39
7.3.2 Use of Diagrams . . . 39
7.3.3 Interleave versus "True" Concurrency . . . 40
7.4 Development Management . . . 40
7.5 What Next ? . . . 41

8 References **42**

A Indexes **44**
A.1 Sorts and Types . . . 44
A.2 "Global" Values . . . 45
A.3 Functions . . . 45
A.4 Axioms . . . 45
A.5 Channels . . . 45
A.6 Behaviours . . . 46
A.7 All Formal Identifiers . . . 46

1 Two Approaches to Modeling

In this report we present a model of a customer/retailer/supplier/... market. We do it in the more-or-less classical style which emanated from the denotational-like formal specification of programming languages and lead to VDM [17, 18] – and from there to RAISE [29]. There are other approaches to modeling discrete systems. One is by means of *symbolic* Petri nets [33, 34, 35, 36, 37].

1.1 Domain Science & Engineering: 𝔻𝕊&𝔼

At the center of 𝔻𝕊&𝔼 stands 𝔻𝔸&𝔻: a *domain analysis & description* method. 𝔻𝔸&𝔻 is first outlined in [11, 12, Bjørner, 2010]. 𝔻𝔸&𝔻 found a more final form in [14, 15, Bjørner, 2016-2019]. [15] form the core chapters, Chapters 3–6, of [16]. That forthcoming Springer monograph, [16], covers the 𝔻𝕊&𝔼 concept: *domain science & engineering*.

In this report we shall *slavishly* follow the doctrines of the 𝔻𝔸&𝔻 method. First we consider endurants[3] and "within" our analysis & description of endurants we first focus on their so-called *external qualities* ("form, but not content"), then on *internal qualities: unique identifiers*, *mereology* and *attributes*. Then, by *transcendental deduction*, we "morph" some endurants into perdurants[4], that is, behaviours. Here we first consider the *channels* and the messages sent over channels between behaviours, before we consider these latter. We do so in the style of [RSL's [28]] CSP [30, 31, 38, 39, 32].

It may seem a long beginning before we get to "process-oriented" modeling.

But a worthwhile thing is worth doing right, hence carefully!

It seems to this author that the HERAKLIT approach, in keeping with its name, from the first beginning, considers "all as flowing", that is, as [Petri net-like] processes.

1.2 HERAKLIT: http://heraklit.dfki.de/

Based on Petri net ideas [33, 34, 35, 36, 37] Wolfgang Reisig has conceived and researched HERAKLIT. In a number of reports and papers, [21, 24, 27, 26, 25, 23, 22], Peter Fettke and Wolfgang Reisig has developed the HERAKLIT theory & practice of modeling, what they call *service systems*.

The present report "mimics" [26, HERAKLIT *case study: retailer*] in providing a Domain Analysis & Description (𝔻𝔸&𝔻)-oriented model of "the same" domain!

The HERAKLIT *retailer case study* [26] straddles three concerns: presenting the HERAKLIT methodology, its mathematical foundation and the retailer case study. The 𝔻𝔸&𝔻 case study presented in this report makes use of RSL, the RAISE Specification Language [28] – and can thus concentrate on the case study. The semantics of the RSL-expressed case study is that derived from the semantics of RSL, notably its CSP [30, 31, 38, 39, 32] "subset".

2 The retailer market Case Study

The following case study is based on [26]. It does not, in the present version, follow the domain of [26] strictly. But I am quite sure that any discrepancies can be easily incorporated into the present model.

2.1 Three Rough Sketches

It is good domain modeling development practice to start a domain modeling project with one or more alternative rough sketch informal descriptions. But they are to be just such rough sketches. No formal meaning is to be attached to these rough sketches. They are meant to get the domain modeling project team "aligned".

We present three, obviously "overlapping", rough sketches.

2.1.1 Identification of "Main Players"

We **rough sketch** narrate a description of the domain.

The domain is that of a set of **customers**, a set of **retailers**, a set of **suppliers**, a set of **courier services**. **Retailers** each embody three sub-components: an **order management**, an **inventory**; and a **warehouse**.

[3] Endurants, colloquially speaking, "end up" as data in the computer.
[4] Perdurants, colloquially speaking, "end up" as processes in the computer.

See Fig. 1

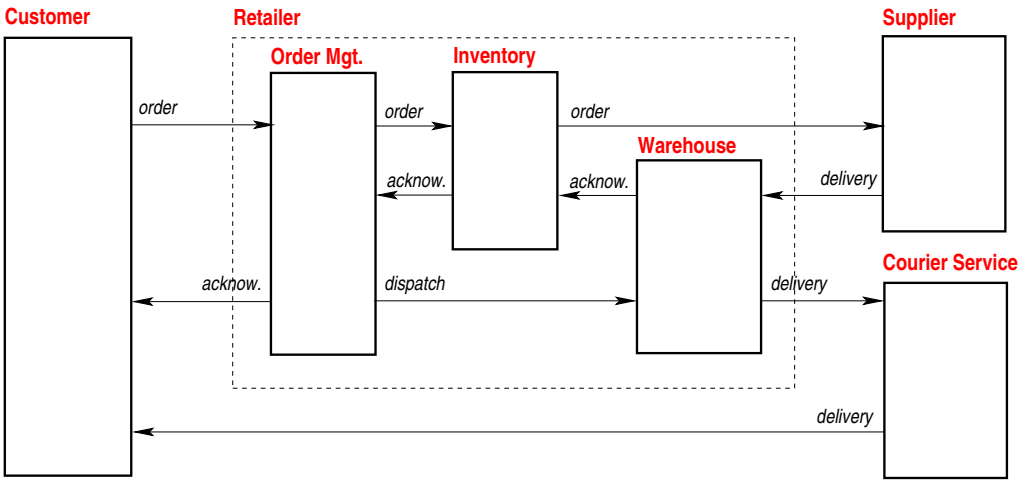

Figure 1: A *Market System*
[5]

Customers **order** merchandise from retailers' order management. They in tern **order** that merchandise from their inventory [management]. If inventory [management] judges that they have the needed quantity in their warehouse, they **acknowledge** the order management. If inventory [management] judges that they do not have the needed quantity in their warehouse, they proceed to **order** a sufficient quantity of the desired merchandise from a supplier. The supplier eventually **deliver** a quantity to the warehouse of the ordering retailer. That warehouse **acknowledges** receipt to its inventory which eventually **acknowledges** that receipt to its order management. The order management **acknowledges** the customer order and notifies its warehouse of a proper **dispatch**. The warehouse **delivers** the desired merchandise quantity to a courier service which subsequently **delivers** that desired merchandise quantity to the customer.

It is thus we see that there are essentially three four kinds of transactions between market "players": **orders**, **acknowledgments**, **dispatches** and **deliveries**.

In the formalisations to follow we shall refer to customers as c:C, order managements as om:OM, inventories as iv:IV, suppliers as s:S and courier services as cs:CS.[6]

2.1.2 Main Transaction Sequences

Customers issue **purchase orders** for **merchandise** from retailers' **order management**; receive **order acknowledgments** from retailers' **order management**; and receive **customer delivered merchandise** (via retailers' warehouses) from **courier services**.

Retailers' order management inquire with its **inventory** as to the availability of ordered **merchandise**; await **acknowledgment** of **availability** (of **merchandise**) from its **inventory**; informs **customer** of availability (**order acknowledgment**); and and **dispatch order** to **warehouse** when available.

[5] None of the figures in this report, Figs. 1, 2 on the following page, 3 on page 9, 4 on page 10 and 5 on page 11, are formal. That is, they do not add to or detract from the meaning of the formulas otherwise shown in this report. They merely "support", by graphics, the narrative text.

[6] We shall, corresponding, prefix the transaction names: C_OM_Order, OM_I_Order, IV_S_Order, IV_WH_Delivery, WH_CS_Delivery, CS_C_Delivery, WH_IV_Ack, IV_OM_Ack, OM_C_Ack, OM_WH_Dispatch, or some suitable variants thereof.

Retailers' inventory issues **acknowledgment** of **merchandise** to **order management**; issues **wholesale orders** for supply of **merchandise**, "when out-of-stock", from **suppliers**; and receive **acknowledgment** of **supplies** from **suppliers**.

Retailers' warehouse receive **merchandise deliveries** from **suppliers**; informs **inventory** management of **merchandise availability**; accepts **dispatch orders** from **order management**; and **forward merchandise** for such customer **merchandise** dispatches to **courier services**.

Etcetera.

2.1.3 Detailed Sketch

We refer to Fig. 2.

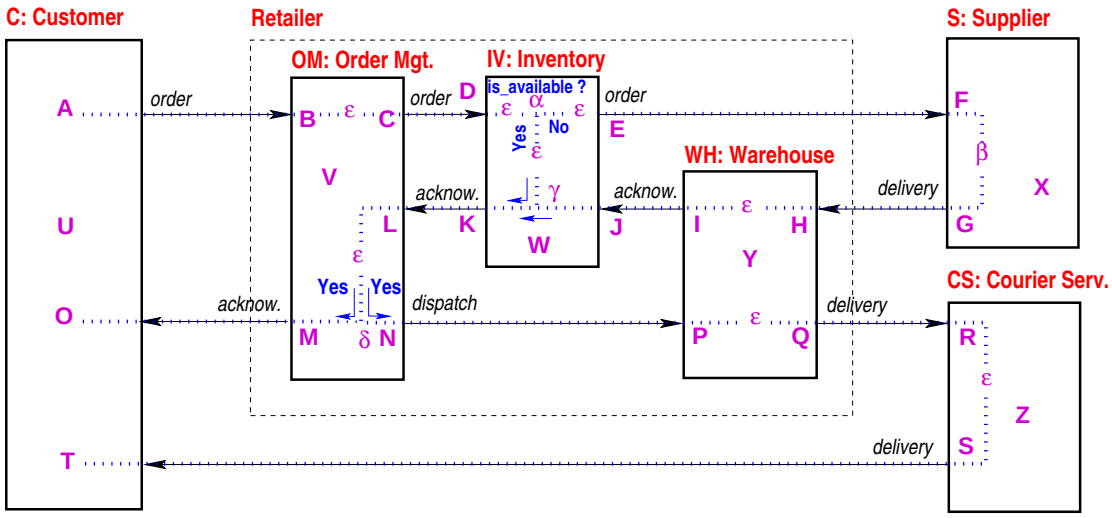

Figure 2: Transaction Sequences

- A It all starts with a customer issuing a purchase order. It is date-time stamped with the customers unique identifier.

  Since no customer can issue more than one such order at a time, such date-time-customer identification is unique and can serve as the unique customer order identification across the market.

  Once the customer has issued the order request it either O awaits replies from some retailer's order management or T some courier service's delivery (of otherwise ordered products) or U resumes other business!

- B The customer order is received by some retailer's order management. That order management makes a note of the incoming order and posits that note in a 'work-to-do' dossier.

- ϵ_B At some time the order management selects an arbitrary "what-to-do-next" note from its dossier. If it is that if an customer order – arising from B – then it

- C issues an inquiry to its inventory as to the availability of the quantity of the named product of the customer order.

- D The inventory receives an inventory inquiry. The inventory makes a note of the incoming order and deposits that note in its 'work-to-do' dossier.
- ϵ_D At some time the inventory selects an arbitrary "what-to-do-next" note from its dossier. If it is that if an inventory inquiry – arising from D – then it
- α examines whether the quantity of the named product of the customer order is "on-hand" (in the retailer's warehouse, as recorded in the inventory).
- E If not the inventory issues a wholesale order to a supplier.
- F The supplier receives a wholesale order. The supplier makes a note of the wholesale order and deposits that note in its 'work-to-do' dossier.
- β It may take same time to respond to the wholesale request. For example, if the supplier first has to manufacture or otherwise get hold of the requested supply.
- G Eventually the supplier transfers the requested quantity of named merchandise to the requesting retailer's warehouse.
- H The warehouse receives this delivery and – eventually - stores it –
- I while notifying its inventory (management) of availability of the [previously] requested merchandise.
- J The inventory receives this notification. It make a note thereof and deposits it in its 'work-to-do' dossier.
- γ Either inquiry α lead to a positive result, or, as now (M) such an inquiry would be positive.
- K Eventually the inventory can inform order management of order availability.
- L Order management receives positive acknowledgment and deposits notes in its 'work-to-do' dossier as to acknowledging the customer of its order and informing the warehouse of its delivery.
- M Eventually order management gets around to service this note:
  - (D,α,Yes) and (D,α,No,E,F,G,J,I,J) order management informs the customer of upcoming order delivery
  - N while also, "at the same time", issuing an order dispatch to its warehouse.
- O The customer receives this information.
- P The warehouse receives this dispatch and makes a note thereof in its 'work-to-do' dossier.
- Q The warehouse eventually issues a delivery order, with ordered merchandise, to a courier service.
- R The courier service receives this delivery and makes a note thereof in its 'work-to-do' dossier.
- S The courier service eventually dispatches the delivery to the customer.
- T The customer, finally, receives the ordered quantity of merchandise.

2.1.4 Transitions

In the technical terms of `Petri nets`, the ten (10) horisontal arrows of Fig. 2 on page 6 represent transitions as in **Place-Transition** `nets`. They are labeled by pairs of upper case alphabetic characters: **A–B, C–D, E–F, G–H, I–J, K–L, M–N, O–P, Q–R,** and **S–T**. In the technical terms of `CS`, these ten transitions correspond to pairs of `CSP` input [*ch[...]?*] and output [*ch[...]!msg*] clauses. You will find these clauses **highlighted in blue** in Sect. 6.2:

- **A–B**: Items 98 Page 24 and 113 Page 25
- **C–D**: Items 119 Page 26 and 141 Page 28
- **E–F**: Items 162 Page 30 and 208 Page 36
- **G–H**: Items 177 Page 32 and 216 Page 36
- **I–J**: Items 196 Page 34 and 144 Page 29
- **K–L**: Items 168 Page 31 and 122 Page 26
- **M–O**: Items 129 Page 27 and 100 Page 24
- **N–P**: Items 130 Page 27 and 180 Page 33
- **Q–R**: Items 202 Page 35 and 222 Page 37 and
- **S–T**: Items 227 Page 37 and 102 Page 24.

The pairs of formulas listed in each • above represents the **transition**. The formula text from the behaviour definition parameter line up up to the "transition" line defines the **place**. Thus the `RSL/CSP` definition that we shall present, in a sense, corresponds to place-transition nets where each transition has exactly two inputs and two outputs. The other way around: Place-transition nets where transitions have different numbers of inputs, respectively outputs, can be likewise "mimicked" by appropriate `RSL/CSP` definitions.

3 Endurants: External Qualities

We now begin the proper, methodical description of the retailer, i.e., the market system. That description is presented in Sects. 3–6.

We refer to [16, *Chapter 3*].

3.1 Main Decompositions

3.1.0.1 Narrative

1. Our market system comprises
2. a customer aggregate,
3. a retailer aggregate,
4. a supplier aggregate and
5. a courier service aggregate.

We consider all these aggregates to be *structures* in the sense of [16, Sect. 4.10].

3.1.0.2 Formalisation

type
1. MKT
2. CSTa
3. RETa
4. SUPa
5. CSa

value
2. obs_CSTa: MKT → CSTa
3. obs_RETa: MKT → RETa
4. obs_SUPa: MKT → SUPa
5. obs_CSa: MKT → CSa

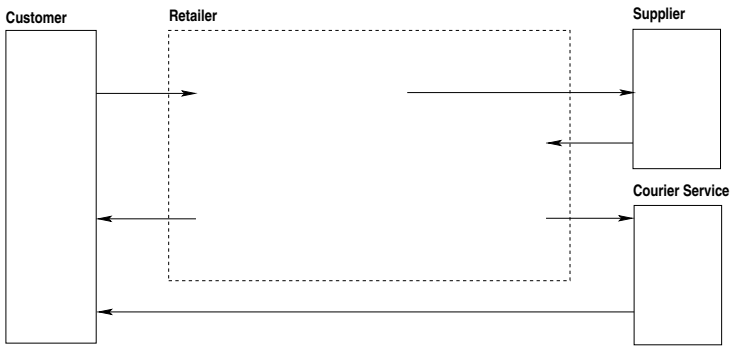

Figure 3: A Simplified Market System

3.2 Aggregates as Sets

3.2.0.1 Narrative

6. The customer aggregate form a set of one or more customers.
7. The retailer aggregate form a set of one or more retailers.
8. The supplier aggregate form a set of one or more suppliers.
9. The courier service aggregate form a set of one or more courier services.

We consider all these sets to be *structures* and the customers, suppliers and courier services to be *atoms* in the sense of [16, Sects. 4.10 and 4.13].

3.2.0.2 Formalisation

type
6. CSTs = C-**set**, **axiom** ∀ csts:CSTs • csts≠{}
7. RETs = R-**set**, **axiom** ∀ rets:CSTs • rets≠{}
8. SUPs = S-**set**, **axiom** ∀ sups:CSTs • sups≠{}
9. CSs = CS-**set**, **axiom** ∀ cts:CSs • trss≠{}

value
6. obs_CSTs: CSTa → CSTs
7. obs_RETs: RETa → RETs
8. obs_SUPs: SUPa → SUPs
9. obs_CSs: CSa → CSs

3.3 The Retailer

3.3.1 The HERAKLIT View

We focus on retailers. We treat retailers as *structures*[7,8] of three separately observable parts:

[7] We refer to [26, Sect. 3.10].
[8] We dash the retailer boxes to indicate their "structure"-ness.

Figure 4: Aggregates as Sets

3.3.1.1 Narrative

10. an *order management*,
11. an *inventory*[9] and
12. a *warehouse*.

We consider order managements, inventory managements and warehouses to be *atoms* in the sense of [16, Sects. 4.13].

3.3.1.2 Formalisation

type
10. OM
11. IV
12. WH

value
10. obs_OM: R → OM
11. obs_IV: R → IV
12. obs_WH: R → WH

3.3.2 The DS&E View

Following the DS&E "approach", i.e., "dogma", retailers might normally have been decomposed into just two components: The order management and the warehouse. Inventory would then become a programmable attribute of order management.

3.4 The Market System State

We refer to [16, *Sect. 3.18*]. We postulate some market system mkt. It consists of

3.4.0.1 Narrative

13. the market, mkt;
14. all customers *cs*;

[9] We might have modeled a retailer inventory as an attribute of the composite part retailer.

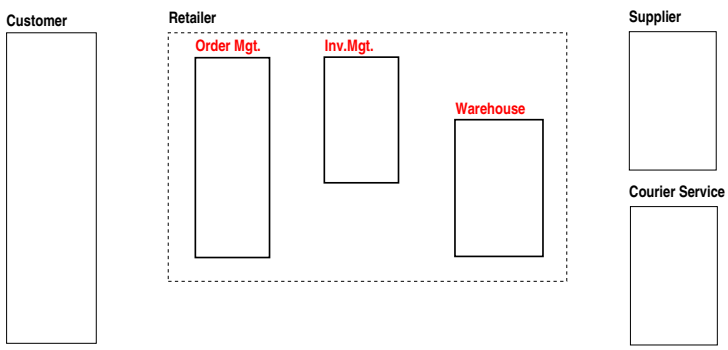

Figure 5: The Retailer

15. all retailer order managements oms;
16. all retailer inventories ivs; and
17. all retailer warehouses whs;
18. all suppliers ss; and
19. all courier services css.

To obtain these we define respective extraction functions.

3.4.0.2 Formalisation

value

13. mkt:MKT

14. xtr_Cs: MKT → C-**set**
14. xtr_Cs(mkt) ≡ obs_CSTs(obs_CSTa(mkt))
14. cs:C-**set** := xtr_Cs(mkt)

15. xtr_OMs: MKT → OM-**sett**
15. xtr_OMs(mkt) ≡ {om|r:RET,om:OM•r ∈ obs_RETs(obs_RETa(mkt))∧om=obs_OM(r)}
15. oms:OM-**set** := xtr_OMs(mkt)

16. xtr_IVS: MKT → IV-**set**
16. xtr_IVS(mkt) ≡ {iv|r:RET,iv:IV•r ∈ obs_RETs(obs_RETa(mkt))∧iv=obs_IV(r)}
16. ivs:IV-**set** := xtr_IVS(mkt)

17. xtr_WHs: MKT → WH-**set**
17. xtr_WHs(mkt) ≡ {wh|r:RET,wh:WH•r ∈ obs_RETs(obs_RETa(mkt))∧wh=obs_WH(r)}
17. whs:WH-**set** := xtr_WHs(mkt)

18. xtr_Ss: MKT → S-**set**
18. xtr_Ss(mkt) ≡ obs_SUPs(obs_SUPa(mkt))
18. ss:S-**set** := xtr_Ss(mkt)

19. xtr_CSs: MKT → CS-**set**
19. xtr_CSs(mkt) ≡ obs_CSs(obs_CSa(mkt))
19. css:CS-**set** := xtr_CSs(mkt)

4 Endurants: Internal Qualities

4.1 Unique Identifiers

We refer to [16, *Sect. 5.2*].

The concept of parts having unique identifiability, that is, that two parts, if they are the same, have the same unique identifier, and if they are not the same, then they have distinct identifiers, that concept is fundamental to our being able to analyse and describe internal qualities of endurants. So we are left with the issue of "sameness" !

4.1.0.1 Narrative

20. Customers, retailer order managements, retailer inventories, retailer warehouses, suppliers and courier services all have distinct unique identifiers.
21. By UI we designate the sort of all unique identifiers.
22. We define auxiliary functions which observe the unique identifiers of all customers, retailers, suppliers and courier services of a market system.
23. *uis* name the set of all unique identifiers.

4.1.0.2 Formalisation

type
20. C_UI, OM_UI, IV_UI, WH_UI, S_UI, CS_UI
21. UI = C_UI | OM_UI | IV_UI | WH_UI | S_UI | CS_UI

value
20. uid_C: C → C_UI
20. uid_OM: OM → OM_UI
20. uid_IV: IN → IV_UI
20. uid_WH: WH → WH_UI
20. uid_S: S → S_UI
20. uid_CS: CS → CS_UI

axiom
20. ∀ c,c′:C•{c,c′}⊆*cs* ∧ c≠c′ ⇒ uid_C(c)≠uid_C(c′),
20. ∀ om,om′:OM•{om,om′}⊆*oms* ∧ om≠om′ ⇒ uid_OM(om)≠uid_OM(om′),
20. ∀ iv,iv′:IV•{iv,iv′}⊆*ivs* ∧ iv≠iv′ ⇒ uid_IV(iv)≠uid_IV(iv′),
20. ∀ wh,wh′:WH•{wh,wh′}⊆*whs* ∧ wh≠wh′ ⇒ uid_WH(wh)≠uid_WH(wh′),
20. ∀ s,s′:S•{s,s′}⊆*ss* ∧ s≠s′ ⇒ uid_S(s)≠uid_S(s′),
20. ∀ cs,cs′:CS•{cs,cs′}⊆*css* ∧ cs≠cs′ ⇒ uid_CS(cs)≠uid_CS(cs′).

value
22. xtr_C_UIs: MKT → CI-**set**
22. xtr_C_UIs(mkt) ≡ {uid_C(c)|c:C•c ∈ *cs*}
22. xtr_OM_UIs: MKT → OMI-**set**
22. xtr_OM_UIs(mkt) ≡ {uid_OM(om)|om:OM•om ∈ *oms*}
22. xtr_IV_UIs: MKT → IVI-**set**
22. xtr_IV_UIs(mkt) ≡ {uid_IV(iv)|iv:IV•iv ∈ *ivs*}
22. xtr_WH_UIs: MKT → WHI-**set**
22. xtr_WH_UIs(mkt) ≡ {uid_WH(wh)|wh:WH•wh ∈ *whs*}
22. xtr_S_UIs: MKT → SI-**set**
22. xtr_S_UIs(mkt) ≡ {uid_S(s)|s:S•s ∈ *ss*}
22. xtr_CS_UIs: MKT → CSI-**set**
22. xtr_CS_UIs(mkt) ≡ {uid_CS(cs)|cs:CS•cs ∈ *css*}

22. *cuis*:CUI-**set** = xtr_C_UIs(mkt)

22. *omuis*:OMUI-**set** = xtr_OM_UIs(mkt)
22. *ivuis*:IVUI-**set** = xtr_IV_UIs(mkt)
22. *whuis*:WHUI-**set** = xtr_WH_UIs(mkt)
22. *suis*:SUI-**set** = xtr_S_UIs(mkt)
22. *csuis*:CSUI-**set** = xtr_CS_UIs(mkt)
23. *uis*:UI-**set** = *cuis* ∪ *omuis* ∪ *ivuis* ∪ *whuis* ∪ *suis* ∪ *csuis*

axiom

22. **card** *cuis* + **card** *omuis* + **card** *ivuis* + **card** *whuis* + **card** *suis* + **card** *csuis* = **card** *uis*

4.2 Mereology

We refer to [16, *Sect. 5.3*].

Mereology, as a logical/philosophical discipline, can perhaps best be attributed to the Polish mathematician/logician Stanisław Leśniewski [20, 13].

Which are the relations that can be relevant for "endurant-hood" ? There are basically two relations: (i) physical ones, and (ii) conceptual ones.

(i) Physically two or more endurants may be topologically either adjacent to one another, like rails of a line, or within an endurant, like links and hubs of a road net, or an atomic part is conjoined to one or more materials, or a material is conjoined to one or more parts. The latter two could also be considered conceptual "adjacencies".

(ii) Conceptually some parts, like automobiles, "belong" to an embedding endurant, like to an automobile club, or are registered in the local department of vehicles, or are 'intended' to drive on roads

4.2.1 Customer Mereology

4.2.1.1 Narrative

24. The mereology of a customer is a pair:
    - the set of all retail order management identifiers and
    - the set of all courier service identifiers.

4.2.1.2 Formalisation

type

24. C_Mer = OM_UI-**set** × CSU_I-**set**

value

24. mereo_C: C → C_Mer
24. mereo_C(c) ≡ (*omuis*,*csuis*)

4.2.2 Order Management Mereology

4.2.2.1 Narrative

25. The mereology of an order management is the triplet of
    - the set of all customer identifiers,
    - the unique identifier of the retailer's inventory and
    - the unique identifier of the retailer's warehouse.

4.2.2.2 Formalisation

```
type
25. OM_Mer = C_UI-set × IV_UI × WH_UI
value
25. mereo_OM: OM → OM_Mer
25. mereo_OM(om) ≡
25.     let r:R • r ∈ obs_RETs(obs_RETa(mkt))∧om=obs_OM(r) in
25.     (cis,uid_IV(obs_IV(r)),uid_WH(obs_WH(r))) end
25.     pre: ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧om=obs_OM(r)
```

4.2.3 Inventory Mereology

4.2.3.1 Narrative

26. The mereology of an inventory is a triplet of
    - the unique identifier of that inventory's order management,
    - the unique identifier of that inventory's warehouse and
    - the set of all supplier identifiers.

4.2.3.2 Formalisation

```
type
26. IV_Mer = OM_UI × WH_UI × S_UI-set
value
26. mereo_IV: IV → IV_Mer
26. mereo_IV(iv) ≡
26.     let r:R • r ∈ obs_RETs(obs_RETa(mkt))∧iv=obs_IV(r) in
26.     (uid_OM(obs_OM(r)),uid_WH(obs_WH(r)),suis) end
26.     pre: ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧iv=obs_IV(r)
```

4.2.4 Warehouse Mereology

narrative

27. The mereology of a warehouse is a quadruplet of
    - the warehouse retailer's order management identifier,
    - the warehouse retailer's inventory identifier,
    - the set of all supplier identifiers, and
    - the set of all courier service identifiers,

4.2.4.1 Formalisation

```
type
27. WH_Mer = OM_UI × IV_UI × _SUI-set × CS_UI-set
value
27. mereo_WH: WH → WH_Mer
27. mereo_WH(wh) ≡
27.     let r:R • r ∈ obs_RETs(obs_RETa(mkt))∧iv=obs_WH(wh) in
27.     (uid_OM(obs_OM(r)),uid_IV(obs_IV(r)),suis,csuis) end
27.     pre: ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh=obs_WH(r)
```

4.2.5 Supplier Mereology

4.2.5.1 Narrative

28. The mereology of a supplier is a pair:
    - the set of all inventory identifiers and
    - the set of all warehouse identifiers.

4.2.5.2 Formalisation

type
28. S_Mer = IV_UI-**set** × WH_UI-**set**
value
28. mereo_S: S → S_Mer
28. mereo_S(s) ≡ ({uid_IV(iv)|iv:IV • iv ∈ $ivuis$},{uid_WI(wh)|wh:WH • wh ∈ whs})

4.2.6 Courier Service Mereology

4.2.6.1 Narrative

29. The mereology of a courier service is a pair
    - the set of all warehouse identifiers and
    - the set of all customer identifiers.

4.2.6.2 Formalisation

type
29. CS_Mer = WH_UI-**set** × CS_UI-**set**
value
29. mereo_CS: CS → CSMer
29. mereo_CS(t) ≡ ({uid_WI(wh)|wh:WH • wh ∈ whs},$cuis$)

4.3 Attributes

We refer to [16, *Sects. 5.4–5.5*].

To recall: there are three sets of **internal qualities**: unique identifiers, part mereology and attributes. Unique identifiers and mereology are rather definite kinds of internal endurant qualities; attributes form more "free-wheeling" sets of internal qualities.

Since one can talk about transaction events between the six "players", i.e., the customers, order managements, inventories, warehouses, suppliers and courier services of the 'market' we must, really, consider their transaction histories as [programmable] attributes.

In order to deal with the attributes of these six "players" we really need first consider what they are all focused on: namely the merchandise, i.e., products, they order, store, supply and deliver. For this we refer to Sect. 5 on page 22.

4.3.1 Transactions

The 'market' is a typical *transaction-oriented* system. By a *transaction* we shall mean an *event* involving two or more "exchanges" of messages between two behaviours. Behaviours will be defined as the result of *transcendental deductions* of part endurants. With part endurants we associate attributes.

Since we can "talk" about events that "occur to parts", that is, as behaviour properties, we shall attribute some of these events to parts. So parts are attributed the transactions in which their behaviours engage (with other behaviours).

Since we can "talk" about "such-and-such" a transaction having been initiated by a behaviour at such-and-such a time ,we shall provide, with each transaction, a prefix of one or more time-stamped unique identifiers of the part/behaviour issuing the transaction.

4.3.1.1 Narrative

30. DaTi refers to 𝕋𝕀𝕄𝔼. We refer to [16, Sect. 2.5]. The expression record_𝕋𝕀𝕄𝔼 yields a 𝕋𝕀𝕄𝔼. You should think of 𝕋𝕀𝕄𝔼s, for example, as of the form `January 27, 2021: 15:46` and 32 seconds (day, month, year, hour, minute, second).
31. A transaction prefix is either a pair of a customer identifier and a date-time, or is a pair of a pair of order management, inventory, warehouse, supplier or courier service identifier and a date-time, and a transation prefix.
    The specific details of the pairings of unique identifiers and data-times is given in Items 32–41.

4.3.1.2 Formalisation

type
30. DaTi = 𝕋𝕀𝕄𝔼
31. UI_Pref = ...

axiom
31. [...]

4.3.1.3 Narrative

31. More specifically the prefixes are the:
32. purchase order,
33. order inquiry,
34. wholesale order,
35. merchandise delivery,
36. merchandise availability,
37. acknowledge availability,
38. order acknowledgment,
39. dispatch order,
40. forward merchandise and the
41. customer delivery prefixes.

4.3.1.4 Formalisation

type
31. UI_Pref = C_OM_Pref | OM_IV_Pref | IV_S_Pref | S_WH_Pref | WH_IV_Pref | IV_OM_Pref
31. | OM_C_Pref | OM_WH_Pref | WH_CS_Pref | CS_C_Pref
32. C_OM_Pref = (CUI×DaTi)
33. OM_IV_Pref = (OMUI×DaTi)×C_OM_Pref
34. IV_S_Pref = (IVUI×DaTi)×OM_IV_Pref
35. S_WH_Pref = (SUI×DaTi)×IV_S_Pref
36. WH_IV_Pref = (WHUI×DaTi)×S_WH_Pref
37. IV_OM_Pref = (IVUI×DaTi)×(OM_IV_Pref|WH_IV_Pref)
38. OM_C_Pref = (OMUI×DaTi)×IV_OM_Pref
39. OM_WH_Pref = (OMUI×DaTi)×OM_C_Pref
40. WH_CS_Pref = (WHUI×DaTi)×OM_WH_Pref
41. CS_C_Pref = (CSUI×DaTi)×WH_CS_Pref

Two customer to order management to inventory etc. transaction prefixes might then schematically be:

4.3.2 Customer Attributes

In order to go about their business of being customers, customers maintain, somehow or other, in their mind, on paper, or otherwise, a number of notes – which we shall refer to as attributes.

To express some of these attributes we need first introduce some auxiliary types.

4.3.2.1 Narrative

42. Customers, besides unique identity, have further information: customer names, addresses, telephone nos., e-mail addresses, etc.
43. Customers have bank/credit card, i.e., payment refs.
44. An order comprises a product name, a quantity, the total price, and a payment reference.

    For simplicity we shall carry this *'order'* information forward in all market transactions.
45. Customers transact with retailer order managements and courier Services:
46. send purchase order to retailers;
47. receive positive acknowledgment on these orders; and
81. accept customer deliveries: a set of merchandise.

Transactions sent by customers are time-stamped with customers identity. Transactions received by customers are time-stamped [with a time-ordered, latest transaction first] grouping of handler identifications (ui:UI) – where order managements, inventories, suppliers, warehouses and courier services are the handlers.

4.3.2.2 Formalisation

type
42. CustName, CustAddr, CustPhon, CustEmail, ...
42. CustInfo = CustName × CustAddr × CustPhon × CustEmail × ...
43. PayRef
44. Order = (ProdNm × Quant × Price × PayRef)
45. C−Trans = C_OM_Order | OM_C_Ack | CS_C_Del
46. C_OM_Order :: C_OM_Pref × Order
47. OM_C_Ack :: OM_C_Pref × Order
81. CS_C_Del :: CS_C_Pref × Order× (M-**set**|MI-**set**)

Now the attributes.

4.3.2.3 Narrative

48. Customers keep a catalog of merchandise: from whom to order, price, etc. [Simplifying we consider this a static attribute.]
49. Customers keep all the merchandise they have acquired. [A programmable attribute.]
50. Customers can recall [a programmable attribute] the time-stamped transactions it has taken part in wrt. retailer order managements and courier services.

4.3.2.4 Formalisation

type
48. C−Catalog = ...
49. C−Merchandise = M-**set**
50. C_TransHist = C_Trans*

axiom
50. ∀ cth:C_TransHist • [list is time-ordered]

value
48. attr_C_Catalog: C → C_Catalog
49. attr_C_Merchandise: C → C_Merchandise
50. attr_C_TransHist: C → CustTransHist

4.3.3 Order Management Attributes

4.3.3.1 Narrative

51. Order management partakes in several transactions:

46. accepting customer purchase orders;

52. passing on that order to its inventory;

61. accepting product availability acknowledgment from the inventory;

47. informing customer of product availability; and,

53. when available, directing a dispatch order to its warehouse.

54. Order management makes note of accepted, i.e., incoming messages, (B and L) by keeping a [programmable attribute] 'Work-to-do' "notice board" [a "basket", a "dossier"].

4.3.3.2 Formalisation

51. OM_Trans = C_OM_Order | OM_IV_Order | IV_OM_Ack | OM_C_Ack | OM_WH_Dispatch
46. C_OM_Order :: C_OM_Pref × Order
52. OM_IV_Order :: OM_IV_Pref × Order
61. IV_OM_Ack :: IV_OM_Pref × Order
47. OM_C_Ack :: OM_C_Pref × Order
53. OM_WH_Dispatch :: OM_WH_Pref × Order

Now the attributes.

4.3.3.3 Narrative

55. An order management 'work-to-do' dossier keeps a set of zero or more notes: customer orders and inventory acknowledgements.

56. Order management records [a static attribute] which suppliers supply which products.

57. Order management also records the programmable order management transaction history OMTransHist attribute records a time-stamped list of all order management transactions, be they vis-a-vis customers, and its retailer's inventory.

4.3.3.4 Formalisation

type
55. OM_WorkToDo = (C_OM_Order|IV_OM_Ack)-**set**
56. OM_ProdSupp = ProdNm $\overrightarrow{m}$ SUI-**set**
57. OM_TransHist = OM_Trans*

value
55. attr_OM_WorkToDo: OM → OrdrMgtWorkToDo
56. attr_OM_ProdSupp: OM → ProdSupp
57. attr_OM_TransHist: OM → OM_TransHist

4.3.4 Inventory Attributes

4.3.4.1 Narrative

58. Inventories partakes in several transactions:
52. accepting merchandise orders from their order management,
59. issuing wholesale order requests to a designated supplier,
60. accepting order acknowledgments from their warehouse, and
61. issuing merchandise availability messages to their order management.

4.3.4.2 Formalisation

58. IV_Trans = OM_IV_Order | IV_OM_Ack | IV_S_Order | WH_IV_Ack
59. IV_S_Order :: IV_S_Pref × Order × S_UI
60. WH_IV_Ack :: WH_IV_Pref × Order × WH_UI
61. IV_OM_Ack :: IV_OM_Pref × Order

4.3.4.3 Narrative

62. An inventory 'work-to-do' dossier (a programmable attribute) keeps a set of zero or more notes: inventory (merchandise availability) inquiry and merchandise availability.
63. The inventory (a programmable attribute) records, for every product name, its information (as listed in Items 87–92 Page 22), the name of the supplier, and the stock-in-hand.
64. The inventory also records the programmable inventory transaction history IVTransHist attribute records a time-stamped list of all inventory transactions, be they vis-a-vis order management, its retailer's warehouse or a supplier.

4.3.4.4 Formalisation

type
62. IV_WorkToDo = (IV_S_Order|IV_OM_Ack)-**set**
63. IV_Inventory = ProdNm $\overrightarrow{m}$ (WhoSalPrice×SugRetPrice×SalPrice×MInfo×SupNm×IV_Stock)
62. IV_Stock = **Nat**
64. IV_TransHist = IVTrans*

value
62. attr_IV_WorkToDo: IV → IV_WorkToDo
63. attr_IV_Inventory: IV → Inventory
64. attr_IV_TransHist: IV → IV_TransHist

4.3.5 Warehouse Attributes

4.3.5.1 Narrative

65. Warehouses partake in four kinds of transactions:
66. being delivered sets of a product named merchandise from suppliers,
67. informing its inventory of (wholesale) supplier delivery,
68. being ordered by its order management, to dispatch merchandise to customers and
69. delivering merchandise to couriers (for them to deliver to customers).

4.3.5.2 Formalisation

65. WH_Trans = S_WH_Del | WH_IV_Ack | OM_WH_Del | WH_CS_Del
66. S_WH_Del = S_WH_Pref × Order × M-**set**
67. WH_IV_Ack = WH_IV_Pref × Order
68. OM_WH_Dispatch = OM_WH_Pref × Order
69. WH_CS_Del = WH_CS_Pref × Order × M-**set**

4.3.5.3 Narrative

70.
71. The programmable warehouse Store attribute reflects, for every product name the zero, one or more merchandise of that name.
72. The programmable warehouse WHTransHist attribute records a time-stamped list of all warehouse transactions, be they vis-a-vis suppliers, its retailer's inventory, its retailer's order management, and customers.

4.3.5.4 Formalisation

type
70. WH_WorkToDo = (S_WH_Del|OM_WH_Dispatch)-**set**
71. WH_Store = ProdName $\overrightarrow{m}$ M-**set**
72. WH_TransHist = WH_Trans*

value
70. attr_WH_WorkToDo: WH → WH_WH_WorkToDo
71. attr_WH_Store: WH → WH_Store
72. attr_WH_TransHist: WH → WH_TransHist

4.3.6 Supplier Attributes

4.3.6.1 Narrative

73. Suppliers, in this model, partake in two transactions:
74. accepting wholesale orders for merchandise from retailers' inventories, and
75. delivering such merchandise orders to retailers' warehouses.

4.3.6.2 Formalisation

73. S_Trans = IV_S_Order | S_WH_Del
74. IV_S_Order = IV_S_Pref × Order
75. S_WH_Del = S_WH_Pref × Order × M-**set**

4.3.6.3 Narrative

76. The programmable supplier attribute S_WorkToDo temporarily contains 'replicas' of "incoming" IV_S_Orders.
77. The programmable supplier attribute S_Products reflects, for every product name a sufficient[10] number of merchandise of that name.
78. The programmable supplier attribute S_TransHist records a time-stamped list of all supplier transactions, be they vis-a-vis retailers' inventory, and retailers' warehouses.

4.3.6.4 Formalisation

type
76. S_WorkToDo = IV_S_Order-**set**
77. S_Products = ProdNm $\xrightarrow[m]{}$ M-**set**
78. S_TransHist = S_Trans*

value
76. attr_S_WorkToDo: S → S_WorkToDo
77. attr_S_Products: S → S_Products
78. attr_S_TransHist: S → S_TransHist

4.3.7 Courier Attributes

4.3.7.1 Narrative

79. Courier servicess, in this model, partake in two transactions:
80. accepting merchandise delivery orders to customers from retailers' order management, and
81. delivering merchandise to customers

4.3.7.2 Formalisation

type
79. CS_Trans = WH_CS_Del | CS_C_Del
80. WH_CS_Del = WH_CS_Pref × Order × M-**set**
81. CS_C_Del = CS_C_Pref × Order× M-**set**

4.3.7.3 Narrative

82. The programmable courier service attribute CS_WorkToDo reflects current, "live" deliveries, and
83. the programmable attribute CS_TransHist the time stamped history of transactions.

[10]

4.3.7.4 Formalisation

type
82. CS_WorkToDo = CS_C_Del-**set**
83. CS_TransHist = CS_Trans*

value
82. attr_CS_WorkToDo: CS → CS_WtD
83. attr_CS_TransHist: CS → CS_TransHist

5 Merchandise

Merchandise (in [26]: Goods) are, using DS&E, modeled as parts. In [26] they are not considered beyond being somehow identified. It is not clear.

84. We shall model merchandise as atomic parts.

type
84. M

5.1 "Unique Identity"

85. As parts merchandise have unique identity.
86. Although we shall treat merchandise as behaviours we shall assume that merchandise identities are distinct from any other unique identities of the market.

type
85. MI

value
85. uid_M: M → MI

axiom
86. ∀ m:M • uid_M(m) ∉ *cuis* ∪ *omuis* ∪ *ivuis* ∪ *whuis* ∪ *suis* ∪ *tuis*

5.2 "Mereology"

Although merchandise, throughout its lifetime, can be related to suppliers, warehouses, courier services and customers we shall omit modeling the mereology of merchandise.

5.3 "Attributes"

We suggest the following merchandise attributes:

87. product name;
88. wholesale price;
89. suggested retail price;
90. sales price;
91. actual price;
92. further product information: goods category, weight, packaging measures, volume, manufacturer (with place-of-origin), manufacturing date, sale-by-date, an "how-to-use" guide, guarantee, etc., etc.

type
87. ProdNm
88. WhoSalPrice
89. SugRetPrice
90. SalPrice
91. ActPrice
92. ProdInfo

5.4 Representation

We shall not be concerned with the representation of attributes.

6 Perdurants

We refer to [16, *Chapters 6–7*].

By transcendental deduction we now "morph" endurants into perdurants. Parts "morph" into behaviours, here modeled in the style of CSP. Their mereology determine the *channels* between part processes.

6.1 Channels

We refer to [16, *Sect. 7.5*].

In this report we shall postulate a channel array indexed by pairs (expressed as two-element sets) of unique identifiers. These identifiers are prescribed in the mereology of the relevant parts.

93. So there is a channel whose index sets allow the expression of communication between customers, order management, inventories, warehouses, suppliers and courier services.

94. The type of the messages communicated is the union type of the customer, order management, inventory [management], warehouse, supplier and courier service transactions.

channel
93. {ch[{ui,ui′}]|ui,ui′:UI•ui≠ui′∧{ui,ui′}⊆uis}:Channel_Trans
type
94. Channel_Trans = C_Trans|OM_Trans|IV_Trans|WH_Trans|S_Trans|CS_Trans

6.2 Behaviours

We refer to [16, *Sects. 7.6–7.8*].

There now follows a sequence of informal narrative and formal specification texts. The formal texts, in a sense, are a culmination of all the previous formal definitions. The formulas involve rather may identifiers. Some are defined locally, some as behaviour function definition parameters, others in previous formal definitions. To help find you way around all these a set of indexes is provided in Appendix A.

6.2.1 Customer Behaviour

6.2.1.1 Narrative

95. Customers alternate between retailer shopping and otherwise going about their daily life.

    Shopping manifests itself in three related events:

    - A the customer issuing a purchase order;

- O the receiving of acknowledgment of upcoming delivery;
- T the final acceptance of delivery.

Daily life is "modeled" by T. Customers alternate, internal don-deterministically, ⌈⌉, between these four events.

96. A When internal non-deterministically choosing to order merchandise, the customer must decide on which retailer, product, how many and at what cost.
97. The customer then assembles a purchase order
98. which it sends to some retailer's order management.

    We refer to [16, Sect. 2.5.3] for understanding the rôle of record_𝕋𝕀𝕄𝔼.

    We presently omit defining date.
99. Whereupon the customer resumes being a customer, however with updated transaction history.
100. O At some time the customer receives an acknowledgment from a retailer's order management as to the [positive] acceptance of an order which was purchased some while ago (omui,dati).
101. The customer records this in its transaction history while resuming being a customer
102. T At some time the customer receives the delivery of previously ordered mercandise.
103. The customer records the identities (as well as the merchandise) and
104. resumes being a customer.
105. U Et cetera.[11]

6.2.1.2 Formalisation

```
value
96. C: c_ui:CUI×c_mer:(omuis,csuis):C_Mer×C_Catalog → (C_Merchandise×C_TransHist)
96.                 in out { ch[{c_ui,om_ui}] | om_ui:OMUI • om_ui ∈ omuis }
96.                 in { ch[{c_ui,cs_ui}] |  cs_ui:CSUI • cs_ui ∈ csuis } Unit
96. C_Beh(c_ui,c_mer:(omuis,csuis),c_ctlg)(c_merch,c_hist) ≡
96.     A  let (om_ui,order) = decide_on_purchase((custinfo,mertbl),c_hist) in
98.         A–B ch[{c_ui,om_ui}] ! ordr:C_OM_Order(((c_ui,record_𝕋𝕀𝕄𝔼())),order) ;
99.         C(c_ui,c_mer,c_ctlg)(c_merch,⟨ordr⟩^c_hist) end
100.  ⌈⌉ O let M–O ack:OM_C_Ack(prefix,order) = ch[{om_ui,c_ui}] ? in
101.        C(cui,cmer,c_ctlg)(merch,⟨ack⟩^c_hist) end
102.  ⌈⌉ T let S–T del:CS_C_Del(prefix,order,ms) = ch[{cs_ui,c_ui}] ? in
103.        let ms_uis = {uid_MI(m)|m:M•m ∈ ms} in
104.        C(c_ui,c_mer,c_ctlg)(merch ∪ ms,⟨CS_C_Del(prefix,order,ms_uis)⟩^c_hist) end end
105.  ⌈⌉ U ... C(cui,cmer,ctlg′)(merch′,c_hist)
```

[11] We leave it to the reader to be more specific. The "etcetera" could, for example, describe possible updates to the catalog and merchandise repository.

6.2.2 Order Management Behaviour

6.2.2.1 Narrative

106. Being order management, **OM**, manifests itself in six events:
107. **B** accepting customer order,
108. **C** offering inventory order,
109. **L** accepting inventory acknowledgment,
110. **M** offering **OM** acknowledgment acknowledgment to customer, and

     N offering dispatch order to warehouse, and
111. **V** doing other **OM** business.
112. The **OM** behaviour internal non-deterministically (107., 108., 109., 110. and 111.) alternates between **B, C, L, M, N** and **V**:

6.2.2.2 Formalisation

```
106. OM: om_ui:OM_UI × (ommer:(cuis,ivui,whui)):OM_Mer × OM_ProdSupp →
106.       (OM_WorkToDo × OM_TransHist)
106.          in out { ch[{c_ui,om_ui}] | c_ui:CUI•c_ui ∈ cuis }
106.          in out ch[{om_ui,iv_ui}] out ch[{om_ui,wh_ui}] Unit
106. OM(om_ui,om_mer:(cuis,iv_ui,wh_ui),om_prodsupp)(om_wtd,om_hist) ≡
107.       B OM.C_OM_Order(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist)
108.    ⌈⌉ C OM.OM_IV_Order(om_ui,om_mer,_om_prodsupp)(om_wtd,om_hist)
109.    ⌈⌉ L OM.IV_OM_Ack(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist)
110.    ⌈⌉ M,N OM.Handle_Input(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist)
111.    ⌈⌉ V ... OM(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist)
```

6.2.2.3 Narrative

113. **B** **OM.C_OM_Order** external non-deterministically offers to accept purchase orders from customers.
114. In response, **OM.C_OM_Order** makes a note of this request in its work-to-do dossier, that is, of eventually issuing an inventory order.
115. Thereupon **OM.C_OM_Order** resumes being 'order management' with an appropriately udated work-to-do state.

6.2.2.4 Formalisation

```
value
107. COM._OM_Order: om_ui:OM_UI × (om_mer:(cuis,iv_ui,wh_ui)):OM_Mer × OM_ProdSupp →
107.       (OM_WorkToDo × OM_TransHist)
107.          in out { ch[{c_ui,om_ui}] | c_ui:CUI•c_ui ∈ cuis }
107.          in out ch[{om_ui,iv_ui}] out ch[{om_ui,wh_ui}] Unit
107. COM._OM_Order(omui,om_mer:(cuis,ivui,whui),om_prodsupp)(om_wtd,om_hist) ≡
113.    B ⌈⌉ { let A–B ordr:C_OM_Ordr(((c_ui,dati)),order)=ch[{c_ui,om_ui}] ? in
114.           let om_wtd′ = om_wtd ∪ {OM_IV_Ordr(((c_ui,dati)),order,_)} in
115.           OM(om_ui,om_mer,om_prodsupp)(om_wtd′,⟨ordr⟩⌢omhist)
107.         | c_ui:C_UI • c_ui ∈ cuis end end }
```

6.2.2.5 Narrative

116. C OM.OM_IV_Order inquires as to whether order_management has a 'work-to-do' note on ordering a quantity of a named product.
117. If so, it selects that note.
118. It then selects a suitable product supplier and a sufficient quantity of the named product.
119. Finally it offers an inquiry to the enventory.
120. Whereupon it resumes being OM.
121. If OM.OM_IV_Order finds no such note it resumes being OM.

6.2.2.6 Formalisation

```
value
108. OM.OM_IV_Order: om_ui:OMUI × (om_mer:(cuis,ivui,whui)):OM_Mer × OM_ProdSupp →
108.         (OM_WorkToDo × OM_TransHist)
108.             in out { ch[{c_ui,om_ui}] | c_ui:C_UI•c_ui ∈ cuis }
108.             in out ch[{om_ui,iv_ui}] out ch[{om_ui,wh_ui}] Unit
108. OM.OM_IV_Order(om_ui,om_mer:(cuis,iv_ui,wh_ui),om_prodsupp)(om_wtd,om_hist) ≡
116.     C if OM_IV_Ordr(((c_ui,dati)),order,_) ∈ wtd
117.           then let ordr:OM_IV_Ordr(((c_ui,dati)),order,_) • ordr ∈ wtd in
118.                let s_ui:S_UI • find_supplier(order)(om_prodsupp), dati′ = record_𝕋𝕀𝕄𝔼() in
119.                C–D ch[{om_ui,iv_ui}] ! ordr′:OM_IV_Ordr(((om_ui,dati′),(c_ui,dati)),order) ;
120.                OM(om_ui,om_mer,om_prodsupp)(om_wtd \ {ordr},⟨ordr⟩^om_hist) end end
121.           else OM(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist) end

118.   find_supplier: Order × OM_ProdSupp → S_UI, find_supplier(order)(om_prodsupp) ≡ ...
```

6.2.2.7 Narrative

122. L OM.IV_OM_Ack offers to accept an order acknowledgment from the retailer inventory.
123. It places this acknowledgment in the OM's 'work-to-do' "basket" as a "matching" pair of customer acknowledgment and warehouse order dispatch notes.
124. And resumes being OM.

6.2.2.8 Formalisation

```
value
109. OM.IV_OM_Ack: OM_UI × OM_Mer × OM_ProdSupp →
109.         (OM_WorkToDo × OM_TransHist)
109.             in out { ch[{c_ui,om_ui}] | c_ui:C_UI•c_ui ∈ cuis }
109.             in out ch[{om_ui,iv_ui}] out ch[{om_ui,wh_ui}] Unit
109. OM.IV_OM_Ack(om_ui,om_mer:(cuis,iv_ui,wh_ui),m_prodsupp)(om_wtd,om_hist) ≡
122.     L let K–L iv_om_ack:IV_OM_Ack(pref,ordr) = ch[{om_ui,iv_ui}] ? in
123.       let om_wtd′ = om_wtd ∪ {OM_C_Ack(((om_ui,_),pref),ordr),
123.                               OM_WH_Dispatch(((om_ui,_),pref),ordr)} in
124.       OM(om_ui,om_mer,om_prodsupp)(om_wtd′,⟨iv_om_ack⟩^om_hist) end end
```

6.2.2.9 Narrative

125. M,N If a suitable, i.e., "matching", pair of customer acknowledgment and warehouse order dispatch notes, can be found in the 'work-to-do' dossier,
126. then time is recorded,
127. the pair of to-do notes identified and
128. that pair removed from the work-to-do basket, whereupon
129. the customer is notified of the acknowledgement, and
130. the warehouse is notified of the order dispatch;
131. an updated order management transaction history is prepared, and
132. the **OM.OM_C_Ack_OM_WH_Desp** resumes being **OM_Beh**;
133. else **OM.OM_C_Ack_OM_WH_Desp** resumes being **OM**.

6.2.2.10 Formalisation

```
value
110. OM.Handle_Input: OM_UI×OM_Mer×OM_ProdSupp→(OM_WorkToDo×OM_TransHist) Unit
110.         (OM_WorkToDo × OM_TransHist)
110.            in out { ch[{c_ui,om_ui}] | c_ui:CUI•c_ui ∈ cuis }
110.            in out ch[{om_ui,iv_ui}] out ch[{om_ui,wh_ui}]  Unit
110. OM.Handle_Input(om_ui,om_mer:(cuis,iv_ui,wh_ui),om_prodsupp)(om_wtd,om_hist) ≡
125.    M  if ∃ two:{IV_OM_Ack(((om_ui,_),pref),ordr),OM_WH_Dispatch(((om_ui,_),pref),ordr)} • two ⊆ om_wtd
126.          then let dati′ = record_TIME(),
127.                   iv_om_ack = OM_C_Ack(((om_ui,_),pref),ordr) • iv_om_ack ∈ om_wtd,
127.                   om_wh_dis = OM_WH_Dispatch(((om_ui,_),pref),ordr) • om_wh_dis ∈ wtd,
128.                   om_wtd′ = om_wtd \ {iv_om_ack,om_wh_dis} in
129.               { M–O ch[{om_ui,c_ui}] ! om_c_ack′:OM_C_Ack(((om_ui,dati′),pref),ordr) ‖
130.                 N–P ch[{om_ui,wh_ui}] ! om_wh_dis′:OM_WH_Dispatch(((om_ui,dati′),pref),ordr) } ;
131.               let om_hist′ = ⟨om_c_ack′,om_wh_dis′⟩^om_hist in
132.               OM(om_ui,om_mer,om_prodsupp)(om_wtd′,om_hist′) end end
133.          else OM(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist) end
```

6.2.2.11 Narrative

134. V We leave this behaviour further undefined.

6.2.2.12 Formalisation

134. V ...

6.2.3 Inventory Behaviour

6.2.3.1 Narrative

135. The **IV** (inventory) behaviour communicates with the order management and the warehouse of the retailer to which it belongs, and with a variety of suppliers.

    The **IV** behaviour alters between External non-deterministically offering to accept

136. **D** order input communications from its order management;

137. **J** order acknowledgment input communications from its warehouse; and

138. α while internal non-deterministically handling incoming orders;

139. **E** internal non-deterministically offering order output communications to a designated supplier; or

140. **K** internal non-deterministically offering acknowledgment communications to its order management.

6.2.3.2 Formalisation

```
value
135. IV: IV_UI × (om_ui,wh_ui,suis):IV_Mer → (IV_WtD×IV_Inventory×IV_TransHist) Unit
135.           in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
135.           out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
135. IV(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
136.   D    IV.OM_IV_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
137.   J  ⫿ IV.WH_IV_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
138.   α  ⫿ IV.Handle_Input(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
139.   E  ⫿ IV.IV_S_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
140.   K  ⫿ IV.IV_OM_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
135.   pre: iv_ui ∈ ivuis ∧ om_ui ∈ omuis ∧ wh_ui ∈ whuis
135.      ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧iv_ui=uid_IV(r)∧om_ui=uid_OM(r)∧wh_ui=uid_WH(r)
```

6.2.3.3 Narrative

141. **D** The **IV.OM_IV_Order** behaviour offers to accept an order [input] communication from its order management, which, when received, that order is put in the inventory 'work-to-do' basket –

142. to eventually be handled.

143. Whereupon the **IV.OM_IV_Order** resumes being the **IV** behaviours.

6.2.3.4 Formalisation

```
type
??.  Handle_OM_IV_Order :: Prefix × Order
value
136. D IV.OM_IV_order: IV_UI × IV_Mer → (IV_WorkToDo×IV_Inventory×IV_TransHist)  Unit
136.           in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
136.           out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
136. D IV.OM_IV_order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
141.     let C–D OM_IV_Order(prefix,order) = ch[ {om_ui,iv_ui} ] ? in
142.     let iv_wtd′ = {Handle_OM_IV_Order(prefix,order)} ∪ iv_wtd in
143.     IV(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd′,iv_inv,iv_hist) end end
```

6.2.3.5 Narrative

144. [J] The **IV.WH_IV_Ack** behaviour offers to accept a supply availability acknowledgment [input] communication from its warehouse.

145. When received that acknowledgment is put in the inventory 'work-to-do' basket.

146. Whereupon the **IV.OM_IV_Order** resumes being the **IV** behaviours.

6.2.3.6 Formalisation

```
value
137. [J] IV.WH_IV_Ack: IV_UI × IV_Mer → (IV_Inventory × IV_TransHist) Unit
137.           in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
137.           out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
137. [J] IV.WH_IV_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_inv,iv_hist) ≡
144.     let [I–J] WH_IV_ack(prefix,order) = ch[ {wh_ui,iv_ui} ] ? in
145.     let iv_wtd′ = {} ∪ iv_wtd in
146.     IV(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd′,iv_inv,iv_hist) end end
```

6.2.3.7 Narrative

147. [α] If there exists, in the 'work-to-do' basket, a handle_OM_IV_order, cf. Item **??** on page ??.,

148. then observe that order's product name, pn, quantity, q, price, p and payment reference, ref, and

149. observe that product's entry (its wholesale price, wp, suggested retail price, srp, sales price, sp, a recommended supplier, s_ui, and the quantity at hand in the warehouse stock) in the inventory [catalog].

150. If the order quantity is lower than the warehouse stock for that product,

151. then choose a suitable re-order quantity, q′,

152. concoct an inventory-to-supplier order,

153. add that to, and remove the handle order from the 'work-to-do' basket, and

154. adjust the stock quantity in the inventoty catalog,

155. before resuming being the **inventory** behaviour;

156. else update the 'work-to-do' basket with an inventory-to-order management acknowledgment to, and remove the handle order from the 'work-to-do' basket

157. and resume being the **inventory** behaviour.

158. If there does not exists, in the 'work-to-do' basket, a handle_OM_IV_order, then resume being the **inventory** behaviour.

6.2.3.8 Formalisation

```
type
??.  Handle_OM_IV_Order :: Prefix × Order
value
138. [α] IV.Handle_Input: IV_UI × IV_Mer → (IV_WorkToDo × IV_Inventory × IV_TransHist)
138.           in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
138.           out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
138. [α] IV.Handle_Input(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
147.     if ∃ ho:Handle_OM_IV_Order(prefix,order) • ho ∈ iv_wtd
147.       then let ho:Handle_OM_IV_Order(prefix,order) • ho ∈ iv_wtd
148.            let (pn,q,p,ref) = ho in axiom pn ∈ dom iv_inv
149.            let (wp,srp,sp,mi,s_ui,stock) = iv_inv(pn) in axiom [ p ∈ {srp,sp} ]
150.            if q < stock
151.              then let q′:Nat • q′>q ∧ ... in
152.                   let iv_s_order = IV_S_Order(prefix,(pn,q′,wp,iv_ref)) in
153.                   let iv_wtd′ = {iv_s_order} ∪ iv_wtd \ {ho},
154.                       iv_inv′ = iv_inv † [ pn ↦(wp,srp,sp,mi,s_ui,stock − q) ] in
155.                   IV(iv_ui,iv_mer)(iv_wtd′,iv_inv′,iv_hist) end end end
156.              else let iv_wtd′ = {IV_OM_Ack(prefix,order)} ∪ iv_wtd \ {ho} in
157.                   IV(iv_ui,iv_mer)(iv_wtd′,iv_inv,iv_hist) end end
158.       else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end end end end
```

6.2.3.9 Narrative

159. [E] If there exists, in the 'work-to-do' basket, a handle_OM_IV_order,
160. then retrieve that order
161. and remove it from the 'work-to-do' basket while
162. communicating the order, updated with a date-timed prefix, to a designated supplier,
163. and resuming being the **inventory** behaviour.
164. If no handle_OM_IV_order is in the basket, then resume being "an unchanged" **inventory** behaviour.

6.2.3.10 Formalisation

```
139. [E] IV.IV_S_Order: IV_UI × IV_Mer → (IV_WorkToDo × IV_Inventory × IV_TransHist)
135.           in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
135.           out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
139. [E] IV.IV_S_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
159.     if ∃ o:IV_S_Order(prefix,order,s_ui) • o ∈ iv_wtd axiom s_ui ∈ suis
160.       then let o:IV_S_Order(prefix,order,s_ui) • o ∈ iv_wtd in
161.            let iv_wtd′ = iv_wtd \ {o}, dati = record_𝕋𝕀𝕄𝔼() in
162.            E–F ch[ {iv_ui,s_ui} ] ! msg:IV_S_Order(((iv_ui,dati),prefix),order,wh_ui) ;
163.            IV(iv_ui,iv_mer)(iv_wtd′,iv_inv,⟨msg⟩^iv_hist)
139.            end end
163.       else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end
```

6.2.3.11 Narrative

165. K If there exists, in the 'work-to-do' basket, an IV_OM_Ack(prefix,order),
166. then retrieve that order
167. and remove it from the 'work-to-do' basket while
168. communicating the order, updated with a date-timed prefix, to a designated supplier,
169. and resuming being the **inventory** behaviour.
170. If no handle_OM_IV_order is in the basket, then resume being "an unchanged" **inventory** behaviour

6.2.3.12 Formalisation

```
value
140. IV.IV_OM_Ack: IV_UI × IV_Mer → (IV_Inventory × IV_TransHist)
140.               in out ch[ {om_ui,iv_ui} ] in ch[ {wh_ui,iv_ui} ]
140.               out { ch[ {s_ui,iv_ui} ] | s_ui:S_UI•s_ui ∈ suis } Unit
140.  IV.IV_OM_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_inv,iv_hist)
165.     if ∃ a:IV_OM_Ack(prefix,order) • a ∈ iv_wtd
166.        then let a:IV_OM_Ack(prefix,order) • a ∈ iv_wtd in
167.             let iv_wtd′ = iv_wtd \ {a}, dati = record_TIME() in
168.             K–L ch[ {iv_ui,om_ui} ] ! msg:IV_OM_Ack(((iv_ui,dati),prefix),order) ;
169.             IV(iv_ui,iv_mer)(iv_wtd′,iv_inv,⟨msg⟩^iv_hist)
139.             end end
170.        else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end
```

6.2.4 Warehouse Behaviour

6.2.4.1 Narrative

171. The **WH** (warehouse) behaviour accepts supplies from any supplier, provides supply acknowledgments to its inventory, accepts dispatch order from its order management and provides merchandise to any courier service.

     Internal non-deterministically the **WH** behaviour alternates between

172. H external non-deterministically accepting deliveries from suppliers,
173. P accepting order dispatches from its order management,
174. H,P handling deferred [but accepted] inputs,
175. I offering acknowledgments of supplies to its inventory, and
176. Q delivering merchandise (orders) to any one of a number of designated courier services.

6.2.4.2 Formalisation

```
value
171. WH: wh_ui:WH_UI × (om_ui,iv_ui,suis,csuis):WH_Mer →
171.         (WH_WorkToDo×WH_Store×WH_TransHist)
171.             in { ch[{s_ui,iv_ui}] | s_ui:S_UI•s_ui ∈ suis }
171.             out ch[{wh_ui,iv_ui}] in ch[{wh_ui,om_ui}]
171.             out { ch[{wh_ui,c_ui}] | c_ui:C_UI•c_ui ∈ cuis } Unit
171. WH(wh_ui,wh_mer:(wh_ui,om_ui,iv_ui,suis,csuis))(wh_wtd,wh_store,wh_hist) ≡
172.   H   WH.S_WH_Del(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
173.   P  ⌈⌉ WH.OM_WH_Disp(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
174. H,P  ⌈⌉ WH.Handle_Input(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
175.   I  ⌈⌉ WH.WH_IV_Ack(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
176.   Q  ⌈⌉ WH.WH_CS_Deliv(whui,whmer)(wh_wtd,wh_store,wh_hist)
171.   pre: wh_ui ∈ whuis ∧ om_ui ∈ omuis ∧ iv_ui ∈ ivuis
171.       ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)∧om_ui=uid_OM(r)∧iv_ui=uid_IV(r)
```

6.2.4.3 Narrative

177. H The WH.S_WH_Del behaviour external non-deterministically offers to accept a supplier to warehouse delivery message.

178. When received the WH.S_WH_Del behaviour deposits this message in its 'work-to-do' basket.

179. It then resumes being the WH behaviour.

6.2.4.4 Formalisation

```
value
172. H WH.S_WH_Del: wh_ui:WH_UI × (_,_,suis,_):WH_Mer →
172.         (WH_WorkToDo×WH_Store×WH_TransHist)
172.             in { ch[{s_ui,wh_ui}] | s_ui:S_UI•s_ui ∈ suis } Unit
172. H WH.S_WH_Del(wh_ui,wh_mer:(_,_,suis,_))(wh_wtd,wh_store,wh_hist) ≡
177.     { let G–H delivery:S_WH_Del(prefix,order,ms) = ch[{s_ui,wh_ui}] ? in
178.       let wh_wtd′ = wh_wtd ∪ {delivery} in
179.       WH(wh_ui,wh_mer)(wh_wtd′,wh_store,wh_hist)
177.       end end | s_ui:S_UI•s_ui ∈ suis }
172.   pre: wh_ui ∈ whuis ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)
```

6.2.4.5 Narrative

180. P The WH.OM_WH_Disp behaviour offers to accept an order management to warehouse [order] dispatch message.

181. When received the WH.OM_WH_Disp behaviour deposits this message in its 'work-to-do' basket.

182. It then resumes being the WH behaviour.

6.2.4.6 Formalisation

```
value
173. [P] WH.OM_WH_Disp: wh_ui:WH_UI×(om_ui,_,_,_):WH_Mer →
173.         (WH_WorkToDo×WH_Store×WH_TransHist)
173.             in ch[{om_ui,wh_ui}] Unit
173. [P] WH.OM_WH_Disp(wh_ui,wh_mer:(om_ui,_,_,_))(wh_wtd,wh_store,wh_hist) ≡
180.     let [N–P] dispatch:OM_WH_Dis(prefix,order) = ch[{om_ui,wh_ui}] ? in
181.     let wh_wtd′ = wh_wtd ∪ {dispatch} in
182.     WH(wh_ui,wh_mer)(wh_wtd′,wh_store,wh_hist)
180.     end end
171.   pre: wh_ui ∈ whuis ∧ om_ui ∈ omuis ∧ iv_ui ∈ ivuis
171.       ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)∧om_ui=uid_OM(r)∧iv_ui=uid_IV(r)
```

6.2.4.7 Narrative

183. [H,P] The rôle of the **WH.Handle_Input** behaviour is to service either of the two kinds of inputs received by the warehouse, from suppliers, **S**, and from its order management, **OM**.
184. There are owo possible kinds of "deferred" messages.
185. If there is a supplier-to-warehouse, S_WH_Del(prefix,order,ms), delivery message,
186. then that message is "converted" into a warehouse-to-inventory delivery acknowledgment message in, the 'work-to-do' basket,
187. and the **WH.Handle_Input** behaviour reverts to being the **WH** behaviour;
188. else if there is an order management-to-warehouse, OM_WH_Dis(prefix,order), message,
189. then that message is "converted" into a warehouse-to-courier service delivery message, WH_CS_Del(prefix,order), in the 'work-to-do' basket,
190. and the **WH.Handle_Input** behaviour reverts to being the **WH** behaviour;
191. if there are no messages in the basket then the **WH.Handle_Input** behaviour reverts to being the **WH** behaviour.
192.

6.2.4.8 Formalisation

```
value
183. [H,P] WH.Handle_Input: wh_ui:WH_UI × (_,iv_ui,_,_):WH_Mer →
183.                 (WH_WorkToDo×WH_Store×WH_TransHist)
183.                     out ch[{wh_ui,iv_ui}] Unit
183. [H,P] WH.Handle_Input(wh_ui,wh_mer:(_,iv_ui,_,_))(wh_wtd,wh_store,wh_hist) ≡
184.     case wh_wtd of
185.       {S_WH_Del(prefix,order,ms)} ∪ wh_wtd′ →
186.           let wh_wtd″ = wh_wtd′ ∪ {WH_IV_Ack(prefix,order,ms)} in
187.           WH(wh_ui,wh_mer)(wh_wtd″,wh_store,wh_hist) end
188.       {OM_WH_Dis(prefix,order)} ∪ wh_wtd′ →
189.           let wh_wtd″ = wh_wtd′ ∪ {WH_CS_Del(prefix,order)} in
190.           WH(wh_ui,wh_mer)(wh_wtd″,wh_store,wh_hist) end
191.       _ → WH(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
184.     end
```

184. **pre**: wh_ui ∈ *whuis* ∧ iv_ui ∈ *ivuis*
184. ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)∧iv_ui=uid_IV(r)
184. **axiom**: [there can only at most be the two kinds of messages as '**cased**' in the wtd basket.]

6.2.4.9 Narrative

193. I If there exists a warehouse-to-inventory [supplier] delivery acknowledgment message in the 'work-to-do' basket,
194. then select and remove that message from the basket,
195. record the current time, and
196. communicate the acknowledgment message to the inventory,
197. and resume being the appropriately updated **WH** behaviour;
198. else resume being the othereise unchanged **WH** behaviour.

6.2.4.10 Formalisation

value
175. I WH.WH_IV_Ack: wh_ui:WH_UI × (_,iv_ui,_,_):WH_Mer →
175. (WH_WorkToDo×WH_Store×WH_TransHist)
175. **out** ch[{wh_ui,iv_ui}] **Unit**
175. I WH.WH_IV_Ack(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist) ≡
193. **if** ∃ a:WH_IV_Ack(prefix,order,ms) • d ∈ om_wtd
194. **then let** a:WH_IV_Ack(prefix,order,ms) • a ∈ om_wtd **in**
194. **let** wh_wtd′ = wh_wtr \ {a},
195. dati = record_𝕋𝕀𝕄𝔼() **in**
196. I–J ch[{wh_ui,iv_ui}] ! ack:WH_IV_Ack(((wh_ui,dati),prefix),order) ;
197. WH(wh_ui,wh_mer)(wh_wtd′,wh_store,⟨ack⟩⌢wh_hist) **end end**
198. **else** WH(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist) **end**
171. **pre**: wh_ui ∈ *whuis* ∧ iv_ui ∈ *ivuis*
171. ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)∧iv_ui=uid_IV(r)

6.2.4.11 Narrative

199. Q If there exists a order management to warehouse [supplier] delivery message in the 'work-to-do' basket,
200. then select and remove that message from the basket,
201. record the current time, and
202. communicate the acknowledgment message to the inventory,
203. and resume being the appropriately updated **WH** behaviour;
204. else resume being the othereise unchanged **WH** behaviour.

6.2.4.12 Formalisation

```
value
176. [Q] WH.WH_CS_Deliv: wh_ui:WH_UI × (_,_,_,csuis):WH_Mer →
176.         (WH_WorkToDo×WH_CS_Dire×WH_Store×WH_TransHist)
176.             out { ch[{wh_ui,c_ui}] | c_ui:C_UI•c_ui ∈ cuis } Unit
176. [Q] WH.WH_CS_Deliv(wh_ui,wh_mer:(_,_,_,csuis))(wh_wtd,wh_store,wh_hist) ≡
199.   if ∃ a:OM_WH_Disp(prefix,order,cs_ui) • a ∈ om_wtd
200.     then let d:OM_WH_Disp(prefix,order,cs_ui) • a ∈ om_wtd in
200.          let wh_wtd′ = wh_wtr \ {d},
201.              dati = record_TIME(),
201.              os:M-set • os⊆wh_store ∧ card os = q in
202.          [Q–R] ch[{wh_ui,cs_ui}] ! ack:WH_CS_Disp(((wh_ui,dati),prefix),order,os) ;
203.          WH(wh_ui,wh_mer)(wh_wtd′,wh_store \ {os},⟨ack⟩⌢wh_hist) end end
204.     else WH(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist) end
171.   pre: wh_ui ∈ whuis ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt))∧wh_ui=uid_WH(r)
```

6.2.5 Supplier Behaviour

6.2.5.1 Narrative

205. The **S**upplier behaviour internal non-deterministically "alternates" between
206. [F] accepting orders from any retailers' inventory, and
207. [G] delivering such orders to retailers' warehouses.

6.2.5.2 Formalisation

```
value
205. S: S_UI × (ivuis,whuis):S_Mer → (S_WorkToDo×S_Products×S_TransHist)
205.         in { ch[{iv_ui,s_ui}] | iv_ui:IV_UI•iv_ui ∈ ivuis }
205.         out { ch[{s_ui,wh_ui}] | wh_ui:WH_UI•wh_ui ∈ whuis } Unit
205. S(s_ui,s_mer:(ivuis,whuis))(s_wtd,s_products,s_hist) ≡
206.   [F]  S.IV_S_Order(s_ui,s_mer:(ivuis,whuis))(s_wtd,s_products,s_hist)
207.   [G] ∏ S.S_WH_Deliv(s_ui,s_mer:(ivuis,whuis))(s_wtd,s_products,s_hist)
205.   pre: s_ui ∈ suis ∧ ∃ s:S • r ∈ obs_Ss(obs_Sa(mkt))∧sui=uid_UI(r)
```

Please note that we have omitted the "intermediary" behaviour of the **S**upplier *handling* inputs. We suggest that such handling is taken care of directly by the **S.S_WH_Deliv**ery behaviour. Also note that we do not describe payment aspects.

6.2.5.3 Narrative

208. The **S.IV_S_Order** behaviour external non-deterministically offers to accept merchandise orders from any retailer's inventory behaviour.
209. Having received such an order it proceeds to record it in its 'work-to-do' basket –
210. whereupon it resumes being the **S** behaviour (with the updated basket).

6.2.5.4 Formalisation

```
value
206. [F] S.IV_S_Order(s_ui,s_mer:(ivuis,_))(s_wtd,s_products,s_hist) ≡
208.     [] { let [E–F] IV_S_Order(prefix,order) = ch[ {iv_ui,s_ui} ] ? in
209.          let s_wtd′ = s_wtd ∪ {IV_S_Order(prefix,order)} in
210.          S(s_ui,s_mer)(s_wtd′,s_products,s_hist)
208.        | iv_ui:IV_UI•iv_ui ∈ ivuis  end end }
```

6.2.5.5 Narrative

211. If there exists a IV_S_Order(prefix,order) message in the 'work-to-do' basket,
212. then select such a message,
213. examine the order,
214. select the quantified number of merchandise of the ordered product,
215. ascertain the current time,
216. and deliver the message to the warehouse of the requesting retailer;
217. then resume being the **S**upplier behaviour with appropriately updated programmable attributes.
218. If there does not exists a IV_S_Order(prefix,order) message in the 'work-to-do' basket, then revert to being the **S**upplier behaviour.

6.2.5.6 Formalisation

```
value
207. [G] S.S_WH_Deliv(s_ui,s_mer:(ivuis,whuis))(s_wtd,s_products,s_hist) ≡
211.     if ∃ h:IV_S_Order(prefix,order,wh_ui) • h ∈ s_wtd
212.       then let h:IV_S_Order(prefix,order,wh_ui) • h ∈ s_wtd in
213.            let (pn,q,cost,payref) = order in
214.            let ms:M-set • ms ⊆ s_products(pn)∧card ms = q,
215.                dati = record_TIME() in
216.                [G–H] ch[ {s_ui,wh_ui} ] ! d:S_WH_Deliv(((s_ui,dati),pref),order,ms) ;
217.            S(s_ui,s_mer)(s_wtd \ {h},s_products†[ pn↦s_products(pn) \ ms ],⟨d⟩⌢s_hist)
212.            end end end
218.      else S(s_ui,s_mer)(s_wtd,s_products,s_hist) end
```

6.2.6 Courier Service Behaviour

6.2.6.1 Narrative

219. The **CS**, courier service, behaviour internal non-deterministically alternates between
220. [R] offering to accept a warehouse to [customer directed] courier service delivery of merchandise and
221. [S] offering a courier service to customer delivery.

6.2.6.2 Formalisation

```
value
219. CS: CS_UI × CS_Mer → (CS_WorkToDo × CS_TransHist) Unit
219. CS(cs_ui,csmer:(whis, cuis))(cs_wtd,cs_hist) ≡
220.     R     CS.WH_CS_Deliv(cs_ui,csmer:(whis, cuis))(cs_wtd,cs_hist)
221.     S  ⌈⌉ CS.CS_C_Deliv(cs_ui,csmer:(whis, cuis))(cs_wtd,cs_hist)
219.     pre: csui ∈ csuis
```

6.2.6.3 Narrative

222. R The CS.WH_CS_Deliv behaviour external non-deterministically offers to accept a customer directed delivery reqyest from any retailer's warehouse.
223. Receiving such a request it updates its 'work-to-do' basket accordingly,
224. and reverts to being the courier service CS.

6.2.6.4 Formalisation

```
value
220. R CS.WH_CS_Deliv: CS_UI × CS_Mer → (CS_WorkToDo × CS_TransHist) Unit
220. R CS.WH_CS_Deliv(cs_ui,csmer:(whis,_))(cs_wtd,cs_hist) ≡
222.       [] { let Q–R r:WH_CS_Deliv(((wh_ui,dati),prefix),order,os) = ch[{wh_ui,s_ui}] ? in
223.            let cs_wtd′ = cs_wtd ∪ {r} in
224.            CS(cs_ui,cs_mer)(cs_wtd′,cs_hist)
222.          | wh_ui:WH_UI•wh_ui ∈ whuis end end }
220.    pre: cs_ui ∈ csuis
```

225. S If there exists a WH_CS_Del(prefix,order,ms,c_ui) dispatch in the 'work-to-do' basket of a courier service
226. then retrieve this dispatch
227. pass it on to the designated customer
228. and revert to being the courier service, CS, behaviour with appropriately updated arguments.
229. Otherwise continue being the CS behaviour.

6.2.6.5 Formalisation

```
value
221. S CS.CS_C_Deliv: CS_UI × CS_Mer → (CS_WorkToDo × CS_TransHist) Unit
221. S CS.CS_C_Deliv(cs_ui,cs_mer)(cs_wtd,cs_hist) ≡
225.       if ∃ whd:WH_CS_Del(prefix,order,ms,c_ui) • whd ∈ cs_wtd
226.          then let d:WH_CS_Deliv(prefix,order,ms,c_ui) • whd ∈ cs_wtd in
227.                 S–T ch[{cs_ui,c_ui}] ! cd:CS_C_Del(((cs_ui,record_TIME),prefix),order,ms) ;
228.                 CS(cs_ui,csmer)(cs_wtd \ {d},⟨cd⟩⌢cs_hist) end
229.          else CS(cs_ui,csmer)(cs_wtd,cs_hist) end
226.    pre: cs_ui ∈ csuis
```

6.3 System Initialisation

We refer to [16, *Sect. 7.8*].

230. Given a market, cf., mkt Item 13 on page 10, we can "synthesize" an RSL clause that stands for the total behaviour of this market.

    We refer to the system state as "generated" in Sect. 3.4 on page 10.

231. The market behaviour is the parallel composition of

232. the distributed parallel compositions of all customers,

233. the distributed parallel compositions of all order managements,

234. the distributed parallel compositions of all inventories,

235. the distributed parallel compositions of all warehouses,

236. the distributed parallel compositions of all suppliers and

237. the distributed parallel compositions of all courier services.

value
230. mkt, cs, oms, ivs, whs, ss and css.
232. ‖ {C(uid_C(c),mereo_C(c),attr_C_Catalog(c))(attr_C_Merhandise(c),⟨⟩)|c:C•c∈cs}
231. ‖
233. ‖ {OM(uid_OM(om),mereo_OM(c),attr_OM_ProdSupp(om))(attr_OM_WorkToDo(om),⟨⟩)|om:OM•om∈$coms$}
231. ‖
234. ‖ {IV(uid_IV(iv),mereo_IV(iv))(attr_IV_WorkToDo(iv),attr_IV_Inventory(iv),⟨⟩)|iv:IV•iv∈ivs}
231. ‖
235. ‖ {WH(uid_WH(wh),mereo_WH(wh))(attr_WH_WorkToDo(wh),attr_WH_Store(wh),⟨⟩)|wh:WH•wh∈whs}
231. ‖
236. ‖ {S(uid_S(s),mereo_S(s))(attr_S_WorkToDo(s),attr_S_Products(s),⟨⟩)|s:S•s∈ss}
231. ‖
237. ‖ {CS(uid_CS(cs),mereo_CS(cs))(attr_CS_WorkToDo(cs),⟨⟩)|cs:CS•cs∈css}

7 Conclusion

7.1 Critique of the 𝔻A&𝔻 Model

I am, today, January 27, 2021: 15:46, not quite happy with my description.

- It was developed too quickly. I started on this model on Dec. 28, 2020. I was the only one to develop this model, cf. Sect. 7.4 on page 40.
- Along the "road" I did not take time to carefully consider the naming of types, values, functions and behaviours.
- Also: the individual definitions of order management, inventory, warehouse, supplier and courier service behaviours (**OM, II, WH, S, CS**) into their , as of Jan. 21, 2021, is/was uneven.
  - In the **C** (customer) behaviour description (Items 96 on page 24.– 105 on page 24.) "all" is expressed in that one behaviour description, whereas in the **OM, IV, WH, S** and **CS** behaviour descriptions the descriptions are decomposed into separate internal non-deterministic behaviours, but not quite consistently.

  - I have yet to check that the mereologies and attributes of parts are consistently used in respective behaviour definitions.
  - And I have yet to check that the indexing of all defined types, sorts, unique identifiers, mereologies, attributes, channel and behaviours is consistent.

- But, on the whole, the model gives a reasonably adequate picture of how a model in the 𝔻𝕊&𝔼/𝔻𝔸&𝔻 style would express the 𝕳𝕰𝕽𝕬𝕶𝕷𝕴𝕿 *retailer* "challenge".
- All I can say, not in any defense, is: *"I am too*[12] *old for this game these days!"*

7.2 Proofs about Models

Models are developed, carefully, and honed, "perpetually, for several reasons. One is to be able to prove properties of the domain being modeled but where these properties are not explicitly stated. We speculate on a few – with more to come!

- *"The sum total of merchandise, in the market as modeled, is constant: no merchandise "arise out of the blue" (for example at suppliers), no merchandise "disappears mysteriously" (for example in warehouses, courier services or at customers)."*
- *"Product quantity_on_hands in a retailer's inventory (catalog) is always less than or equal to that retailer's corresponding quantities_at_hand in its warehouse."*
- With the ideal assumption that suppliers can always deliver requested numbers of any product: *"Customers are eventually delivered their ordered merchandise."*
- Etcetera!

We do not show any such proofs in this technical report.

7.3 Comparison of Models

We compare our model to that of [26].

7.3.1 "Minor" Discrepancies

- It seems, but this has to be checked, that orders, in the 𝔻𝔸&𝔻 model, are for any number of one particular merchandise product, whereas the 𝕳𝕰𝕽𝕬𝕶𝕷𝕴𝕿 model allows the mixing of several products and of different quantities of these.
- It also seems that ...

MORE TO COME

7.3.2 Use of Diagrams

- Somewhere, in Footnote 5 on page 5, it is said that none of the figures in this report play any rôle in the formal aspects of the 'retailer market' description.
  - That is intended to be so.
  - But is it really true?
    - When I first worked as an M.Sc. graduated engineer in designing data communications "gear" and computers for IBM (1962–1965, 1969) we all drew diagrams!
    - When I then studied computer science (1965-1968) diagrams of software systems (except for "trivial" program flowcharts) were frowned upon.

[12] I was born Oct. 4, 1937

    * `Petri nets` (around 1962–1963) are based almost exclusively on two-dimensional diagrams. They are easy to grasp,
    * The diagrams of HERAKLIT are likewise appealing.
  - Figure 2 on page 6 of this report is rather crucial, I found, in keeping track, while I was developing the description, of all the various segments of that description – in particular in making sure that the interfaces between behaviours "fitted".
  - I can imagine that some readers will find, especially Fig. 2 on page 6 useful when reading the description.

- So, perhaps, diagrams, of the kind that Fig. 2 on page 6 represents, ought be "woven" into the D&D analysis & description, into its *principles, techniques* and *tools*.

7.3.3 Interleave versus "True" Concurrency

The reader is assumed to be quite familiar with these two kinds of semantic terms and their meaning.

- As such, the HERAKLIT-based model, appears to be at an advantage – in that it expresses "true concurrency".
- But, before you get all too excited, the DS&E/D&D model, as its behaviours are defined, "do not lack far behind", if-at-all!
  - On one hand you can read the basically `CSP` clauses as if actions in separate behaviours do indeed occur "truly concurrent".
  - On the other hand, by "splitting" up, as in the behaviour definitions of **C, OM, IV, WH, S** and **CS**, these into separate actions, such as *input*, *handling*, etc., an as full "measure" of local "true concurrency" seems to be achieved.

7.4 Development Management

How to organise the development of domain descriptions such as presented in this report? Here is some advice.

- First conduct a "full trial run", such as the work behind this report represents.
- Then do "the same thing again", but now, "in earnest".
  - The "full trial run" shall cover, basically, the full domain.
  - But it is allowed to "waver", as do the report you are holding in your hand, between different styles of analysis & description.
  - The aim of the "full trial run" is for the development team to settle on "exactly" the style to be followed.
  - At the same time management can better ascertain the manpower and time to be used for the various segments of the work – such as itemized next.

The below items now hint at the, "in earnest" work to be done after a "full trial run".

- Assemble a group of, in this case, as also 'judged' from the "full trial run", six (more) software engineers cum computing scientists, professionally educated, that is, knowledgeable of such texts as [2, 3, 4, 16].

  The group reads the "full trial run" report.

`For the present project I estimate 2 six-seven person-weeks.`[13,14,15]

- Together the whole group, including the project manager, analyses and describes the external qualities – cf. Sects. 2 and 3.

  `For the present project I estimate 2 six-seven person-weeks.`

- Based on the decomposition of the domain endurants, one project member is assigned to distinct parts, as here the *customer, order management, inventory, warehouse, supplier* and *courier service* parts.

  `For the present project I estimate 1 six-seven person-weeks.`

- Each of these staff members now take care, in synchrony with all others, of the *unique identifier* section, then the *mereology* section, and the *attributes* section, one-by-one.

  `For the present project I estimate (1+2+3) six-seven person-weeks.`

- Each member, on a rotating basis, receives the work of a colleague, every morning, reviews it, and all meets, say at 11am, every work day, to discuss and debate each others' models, whereupon they spend the afternoon on continuing their section of work.

- When work on *internal qualities* has ended, they all meet for as longs as it takes, half a day – three days (!) – to settle on channels.

  `For the present project I estimate 1 six-seven person-weeks.`

- Thereupon they work on defining behaviour signatures and, when all such signatures are thought finalised, then the behaviour definitions.

  `For the present project I estimate 2 six-seven person-weeks.`

- And so forth.[16]

- Do not forget the daily late morning meetings!

7.5 What Next?

- It is my sincere hope that Messrs Fettke and Reisig will comment on the present report.

- I need to know where I have misunderstood the [intentions of the] HERAKLIT model [26].

- I need to know where my model fails in modeling what [26] achieves.

- etcetera!

[13] This means: 2 weeks of calendar time for 6–7 persons.

[14] In this initial, "for earnest" project step the group settles on computerised tools, e.g. LaTeX, `RAISE/RSL`, etc.

[15] The Dansk Datamatik Center **Ada** project, January 1981 – September 1984 did just that: based on a "full trial run" of six M.Sc. students' 6 months' master thesis work (February – August 1980), [19]. These M.Sc. students background was essentially that of [2, 3], that is, knowledgeable about applicative, imperative, logic and parallel programming, operational (in those days called 'mechanical'), first-order and denotational semantics, etc.

[16] That is: I estimate a total calendar time of 4+14 weeks for the (one person) "full trial run" plus the (six to seven person) "in earnest" projects.

8 References

[1] Dines Bjørner. Domain Models of "The Market" — in Preparation for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. [17].

[2] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [5, 8].

[3] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen. See [6, 9].

[4] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [7, 10].

[5] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press, 2008.

[6] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press, 2008.

[7] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press, 2008.

[8] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[9] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[10] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[11] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, 2(4):100–116, May 2010.

[12] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, 2(3):100–120, June 2011.

[13] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[14] Dines Bjørner. Manifest Domains: Analysis & Description. *Formal Aspects of Computing*, 29(2):175–225, March 2017. Online: 26 July 2016.

[15] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modelling Languages. *ACM Trans. on Software Engineering and Methodology*, 28(2), April 2019. 68 pages. [18].

[16] Dines Bjørner. *Domain Science & Engineering, A Basis for Understanding Man-made Worlds – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, 2021.

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

[17] http://www2.imm.dtu.dk/~dibj/themarket.pdf
[18] imm.dtu.dk/~dibj/2018/tosem/Bjorner-TOSEM.pdf

[18] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

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

[20] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[21] Peter Fettke and Wolfgang Reisig. Modelling service-oriented systems and cloud services with HERAKLIT. *CoRR*, abs/2009.14040, 2020.

[22] Peter Fettke and Wolfgang Reisig. HERAKLIT – epistemologically motivated modeling of computer-integrated systems. HERAKLIT working paper, v1, December 15, 2020, http://www.heraklit.org, 2020.

[23] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: 8-second hell. HERAKLIT working paper, v1, December 12, 2020, http://www.heraklit.org, 2020.

[24] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: adder. HERAKLIT working paper, v1, December 5, 2020, http://www.heraklit.org, 2020.

[25] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: parallel adder. HERAKLIT working paper, v1, December 5, 2020, http://www.heraklit.org, 2020.

[26] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: retailer. HERAKLIT working paper, v1, December 21, 2020, http://www.heraklit.org, 2020.

[27] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: service system. HERAKLIT working paper, v1, November 20, 2020, http://www.heraklit.org, 2020.

[28] 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.

[29] 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.

[30] Charles Anthony Richard Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8), Aug. 1978.

[31] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[32] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: [19] (2004).

[33] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.

[34] Wolfgang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.

[35] Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003.

[36] Wolfgang Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Leitfäden der Informatik. Vieweg+Teubner, 1st edition, 15 June 2010. 248 pages; ISBN 978-3-8348-1290-2.

[19] usingcsp.com/cspbook.pdf

[37] Wolfgang Reisig. *Understanding Petri Nets Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. 230+XXVII pages, 145 illus.

[38] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. [20].

[39] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

A Indexes

A.1 Sorts and Types

Attribute Types
- **Courier Services**
  - CS_ C_ Del ι81, 17, 21
  - CS_ Trans ι80, 21
  - CS_ TransHist ι83, 22
  - CS_ WorkToDo ι82, 22
  - WH_ CS_ Del ι69, 20
  - WH_ CS_ Del ι80, 21
- **Customer**
  - C_ Catalog ι48, 18
  - C_ Merchandise ι49, 18
  - C_ OM_ Order ι46, 17, 18
  - C_ Trans ι45, 17
  - C_ TransHist ι50, 18
  - CS_ C_ Del ι81, 17, 21
  - CustInfo ι42, 17
  - OM_ C_ Ack ι47, 17, 18
  - Order ι44, 17
  - PayRef ι43, 17
- **Inventory**
  - IV_ Inventory ι63, 19
  - IV_ OM_ Ack ι61, 18, 19
  - IV_ S_ Order ι59, 19
  - IV_ S_ Order ι74, 21
  - IV_ Stock ι63, 19
  - IV_ Trans ι58, 19
  - IV_ TransHist ι64, 19
  - IV_ WorkToDo ι62, 19
  - OM_ IV_ Ordr ι52, 18
  - WH_ IV_ Ack ι60, 19
  - WH_ IV_ Ack ι67, 20
- **Order Management**
  - C_ OM_ Order ι46, 17, 18
  - IV_ OM_ Ack ι61, 18, 19
  - OM_ C_ Ack ι47, 17, 18
  - OM_ IV_ Order ι52, 18
  - OM_ ProdSupp ι56, 19
  - OM_ Trans ι51, 18
  - OM_ TransHist ι57, 19
  - OM_ WH_ Dispatch ι53, 18
  - OM_ WH_ Dispatch ι68, 20
  - OM_ WorkToDo ι55, 19
- **Supplier**
  - IV_ S_ Order ι59, 19
  - IV_ S_ Order ι74, 21
  - S_ Products ι77, 21
  - S_ Trans ι73, 21
  - S_ TransHist ι78, 21
  - S_ WH_ Del ι66, 20
  - S_ WH_ Del ι75, 21
  - S_ WorkToDo ι76, 21
- **Warehouse**
  - CS_ Trans ι80, 21
  - OM_ WH_ Dispatch ι68, 20
  - S_ WH_ Del ι66, 20
  - S_ WH_ Del ι75, 21
  - WH_ CS_ Del ι69, 20
  - WH_ CS_ Del ι80, 21
  - WH_ IV_ Ack ι60, 19
  - WH_ IV_ Ack ι67, 20
  - WH_ Store ι71, 20
  - WH_ Trans ι65, 20
  - WH_ TransHist ι72, 20
  - WH_ WorkToDo ι70, 20

Mereology Types
- C_ Mer=OM_ UI-**set**×CS_ UI-**set** ι24, 13
- CSMer = WH_ UI-**set** × CS_ UI-**set** ι29, 15
- IV_ Mer=OM_ UI×WH_ UI×S_ UI-**set** ι26, 14
- OM_ Mer=C_ UI-**set**×IV_ UI×WH_ UI ι25, 14
- S_ Mer=IV_ UI-**set**×WH_ UI-**set** ι28, 15
- WMer=OM_ UI×IV_ UI×S_ UI-**set**×CS_ UI-**set** ι27, 14

Part Sorts
- CSa ι5, 8
- CSTa ι2, 8
- IV ι11, 10
- MKT ι1, 8
- OM ι10, 10
- RETa ι3, 8
- SUPa ι4, 8
- WH ι12, 10

Part Types
- CSs=CS-**set** ι6, 9
- CSTs=C-**set** ι6, 9
- RETs=R-**set** ι6, 9
- SUPs=S-**set** ι6, 9

Transaction Prefix
- C_ OM_ Pref ι32, 16
- CS_ C_ Pref ι41, 16
- IV_ OM_ Pref ι37, 16
- IV_ S_ Pref ι34, 16
- OM_ C_ Pref ι38, 16
- OM_ IV_ Pref ι33, 16
- OM_ WH_ Pref ι39, 16
- S_ WH_ Pref ι35, 16
- UI_ Pref ι31, 16
- WH_ CS_ Pref ι40, 16
- WH_ IV_ Pref ι36, 16

[20] http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf

Unique Identifier Types
C_ UI ι20, 12
CS_ UI ι20, 12
IV_ UI ι20, 12
OM_ UI ι20, 12
S_ UI ι20, 12
UI ι21, 12
WH_ UI ι20, 12

DaTi=𝕋𝕀𝕄𝔼 ι30, 16

UI_ Prefix ι31, 16

A.2 "Global" Values

cs ι14, 11
css ι19, 11
csuis ι22, 13
cuis ι22, 12
ivs ι16, 11
ivuis ι22, 13
oms ι15, 11
omuis ι22, 13
ss ι18, 11
suis ι22, 13
uis ι22, 13
whs ι17, 11
whuis ι22, 13

mkt ι13, 11

A.3 Functions

Attribute Observers
Courier Services
attr_ CS_ TransHist ι83, 22
attr_ CS_ WtD ι82, 22
Customers
attr_ C_ Catalog ι48, 18
attr_ C_ Merchandise ι49, 18
attr_ C_ TransHist ι50, 18
Inventory
attr_ IV_ Inventory ι63, 19
attr_ IV_ TransHist ι64, 19
attr_ IV_ WorkToDo ι62, 19
Order Management
attr_ OM_ ProdSupp ι56, 19
attr_ OM_ TransHist ι57, 19
attr_ OM_ WorkToDo ι55, 19
Supplier
attr_ S_ Products ι76, 21
attr_ S_ Products ι77, 21
attr_ S_ TransHist ι78, 21
Warehouser
attr_ Store ι71, 20
attr_ WH_ TransHist ι72, 20
attr_ WH_ WorkToDo ι70, 20
Extraction Functions
xtr_ C_ UIs ι22, 12
xtr_ Cs ι14, 11
xtr_ Cs ι18, 11
xtr_ CS_ UIs ι22, 12
xtr_ CSs ι19, 11
xtr_ IV_ UIs ι22, 12
xtr_ IVs ι16, 11
xtr_ OM_ UIs ι22, 12
xtr_ OMs ι15, 11
xtr_ S_ UIs ι22, 12
xtr_ WH_ UIs ι22, 12
xtr_ WHs ι17, 11
Mereology Observers
mereo_ C ι24, 13
mereo_ CS ι29, 15
mereo_ IV ι26, 14
mereo_ OM ι25, 14
mereo_ S ι28, 15
mereo_ WH ι27, 14
Part Obervers
obs_ CSa ι5, 8
obs_ CSTa ι2, 8
obs_ CSTs ι6, 9
obs_ IV ι11, 10
obs_ OM ι10, 10
obs_ RETa ι3, 8
obs_ RETs ι7, 9
obs_ SUPa ι4, 8
obs_ SUPs ι8, 9
obs_ SUPs ι9, 9
obs_ WH ι12, 10
Unique Identifier Observers
uid_ C ι20, 12
uid_ CS ι20, 12
uid_ IV ι20, 12
uid_ OM ι20, 12
uid_ S ι20, 12
uid_ WH ι20, 12

A.4 Axioms

Ordered time stamps ι31, 16

Transaction History is time-ordered, most recent first ι50, 18

Uniqueness of identifiers ι20, 12
Uniqueness of identifiers ι22, 13

A.5 Channels

Channel Declaration	Message Type
ch ι93, 23	Channel_ Trans ι94, 23

A.6 Behaviours

C: c_ ui:C_ UI×(omuis,csuis):C_ Mer×C_ Catalog→(C_ Merchandise×C_ TransHist) **Unit** ι96, 24
IV.Handle_ Input: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι138, 30
IV.IV_ OM_ Ack: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι140, 31
IV.IV_ S_ Order: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι139, 30
IV.OM_ IV_ order: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι136, 28
IV.WH_ IV_ ack: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι137, 29
IV: IV_ UI×IV_ Mer→(IV_ Inventory×IV_ TransHist) **Unit** ι135, 28
OM.C_ OM_ Order: OM_ UI×OM_ Mer×OM_ ProdSupp→(WorkToDo×OM_ TransHist) **Unit** ι107, 25
OM.Handle_ Input: OMUI×OM_ Mer×OM_ ProdSupp→(OM_ WorkToDo×OM_ TransHist) **Unit** ι110, 27
OM.IV_ OM_ Ack: OM_ UI×OM_ Mer×OM_ ProdSupp→(OM_ WorkToDo×OM_ TransHist) **Unit** ι109, 26
OM.OM_ IV_ Order: OM_ UI×OM_ Mer×OM_ ProdSupp→(OM_ WorkToDo×OM_ TransHist) **Unit** ι108, 26
OM: OM_ UI×OM_ Mer×OM_ ProdSupp→(OM_ WorkToDo×OM_ TransHist) **Unit** ι106, 25
S.IV_ S_ Order: S_ UI×S_ Mer→(S_ WorkToDo×S_ Products×S_ TransHist) **Unit** ι206, 36
S.S_ WH_ Deliv: S_ UI×S_ Mer→(S_ WorkToDo×S_ Products×S_ TransHist) **Unit** ι207, 36
S: S_ UI×S_ Mer→(S_ WorkToDo×S_ Products×S_ TransHist) **Unit** ι205, 35
WH.Handle_ Input: WH_ UI×WH_ Mer→(WH_ WorkToDo×WH_ Store×WH_ TransHist) **Unit** ι183, 33
WH.OM_ WH_ Disp: WH_ UI×WH_ Mer→(×OMTransHist) **Unit** ι173, 33
WH.S_ WH_ Deliv: WH_ UI×WH_ Mer→(CustPur×OMTransHist) **Unit** ι172, 32
WH.WH_ CS_ Deliv: WH_ UI×WH_ Mer→(WH_ WorkToDo×WH_ Store×WH_ CS_ Dir×WH_ TransHist) **Unit** ι176, 35
WH.WH_ IV_ Ack: WH_ UI×WH_ Mer→(WH_ WorkToDo×WH_ Store×WH_ TransHist) **Unit** ι175, 34
WH: WH_ UI×WH_ Mer→(WH_ WorkToDo×WH_ Store×WH_ TransHist) **Unit** ι171, 32
CS.CS_ C_ Deliv: CS_ UI×CS_ Mer→(CS_ WorkToDo×CS_ TransHist) **Unit** ι221, 37
CS.WH_ CS_ Deliv: CS_ UI×CS_ Mer→(CS_ WorkToDo×CS_ TransHist) **Unit** ι220, 37
CS: CS_ UI×CS_ Mer→(CS_ WorkToDo×CS_ TransHist) **Unit** ι219, 37

A.7 All Formal Identifiers

attr_ C_ Catalog ι48, 18
attr_ C_ Merchandise ι49, 18
attr_ C_ TransHist ι50, 18
attr_ CS_ TransHist ι83, 22
attr_ CS_ WtD ι82, 22
attr_ IV_ Inventory ι63, 19
attr_ IV_ TransHist ι64, 19
attr_ IV_ WorkToDo ι62, 19
attr_ OM_ ProdSupp ι56, 19
attr_ OM_ TransHist ι57, 19
attr_ OM_ WorkToDo ι55, 19
attr_ S_ Products ι76, 21
attr_ S_ Products ι77, 21
attr_ S_ TransHist ι78, 21
attr_ Store ι71, 20
attr_ WH_ TransHist ι72, 20
attr_ WH_ WorkToDo ι70, 20

C_ Catalog ι48, 18
C_ Merchandise ι49, 18
C_ Mer=OM_ UI-**set**×CS_ UI-**set** ι24, 13
C_ OM_ Order ι46, 17, 18
C_ OM_ Pref ι32, 16
C_ Trans ι45, 17
C_ TransHist ι50, 18
C_ UI ι20, 12
ch ι93, 23
Channel_ Trans ι94, 23
CS_ C_ Del ι81, 17, 21
CS_ C_ Pref ι41, 16
CS_ Trans ι80, 21
CS_ TransHist ι83, 22
CS_ UI ι20, 12
CS_ WorkToDo ι82, 22
CSa ι5, 8
CSMer = WH_ UI-**set** × CS_ UI-**set** ι29, 15
CSs=CS-**set** ι6, 9
CSTa ι2, 8
CSTs=C-**set** ι6, 9
CustInfo ι42, 17

IV ι11, 10
IV_ Inventory ι63, 19
IV_ Mer=OM_ UI×WH_ UI×S_ UI-**set** ι26, 14
IV_ OM_ Ack ι61, 18, 19
IV_ OM_ Pref ι37, 16
IV_ S_ Order ι59, 19
IV_ S_ Order ι74, 21
IV_ S_ Pref ι34, 16
IV_ Stock ι63, 19
IV_ Trans ι58, 19
IV_ TransHist ι64, 19
IV_ UI ι20, 12
IV_ WorkToDo ι62, 19

mereo_ C ι24, 13
mereo_ CS ι29, 15
mereo_ IV ι26, 14
mereo_ OM ι25, 14

mereo_ S ι28, 15
mereo_ WH ι27, 14
MKT ι1, 8

obs_ CSa ι5, 8
obs_ CSTa ι2, 8
obs_ CSTs ι6, 9
obs_ IV ι11, 10
obs_ OM ι10, 10
obs_ RETa ι3, 8
obs_ RETs ι7, 9
obs_ SUPa ι4, 8
obs_ SUPs ι8, 9
obs_ SUPs ι9, 9
obs_ WH ι12, 10
OM ι10, 10
OM_ C_ Ack ι47, 17, 18
OM_ C_ Pref ι38, 16
OM_ IV_ Order ι52, 18
OM_ IV_ Ordr ι52, 18
OM_ IV_ Pref ι33, 16
OM_ Mer=C_ UI-**set**×IV_ UI×WH_ UI ι25, 14
OM_ ProdSupp ι56, 19
OM_ Trans ι51, 18
OM_ TransHist ι57, 19
OM_ UI ι20, 12
OM_ WH_ Dispatch ι53, 18
OM_ WH_ Dispatch ι68, 20
OM_ WH_ Pref ι39, 16
OM_ WorkToDo ι55, 19
Order ι44, 17

PayRef ι43, 17

RETa ι3, 8
RETs=R-**set** ι6, 9

S_ Mer=IV_ UI-**set**×WH_ UI-**set** ι28, 15
S_ Products ι77, 21
S_ Trans ι73, 21
S_ TransHist ι78, 21
S_ UI ι20, 12
S_ WH_ Del ι66, 20
S_ WH_ Del ι75, 21
S_ WH_ Pref ι35, 16
S_ WorkToDo ι76, 21
SUPa ι4, 8
SUPs=S-**set** ι6, 9

UI ι21, 12
UI_ Pref ι31, 16
uid_ C ι20, 12
uid_ CS ι20, 12
uid_ IV ι20, 12
uid_ OM ι20, 12
uid_ S ι20, 12
uid_ WH ι20, 12

WH ι12, 10
WH_ CS_ Del ι69, 20
WH_ CS_ Del ι80, 21
WH_ CS_ Pref ι40, 16
WH_ IV_ Ack ι60, 19
WH_ IV_ Ack ι67, 20
WH_ IV_ Pref ι36, 16
WH_ Store ι71, 20
WH_ Trans ι65, 20
WH_ TransHist ι72, 20
WH_ UI ι20, 12
WH_ WorkToDo ι70, 20
WMer=OM_ UI×IV_ UI×S_ UI-**set**×CS_ UI-**set** ι27, 14

xtr_ C_ UIs ι22, 12
xtr_ Cs ι14, 11
xtr_ Cs ι18, 11
xtr_ CS_ UIs ι22, 12
xtr_ CSs ι19, 11
xtr_ IV_ UIs ι22, 12
xtr_ IVs ι16, 11
xtr_ OM_ UIs ι22, 12
xtr_ OMs ι15, 11
xtr_ S_ UIs ι22, 12
xtr_ WH_ UIs ι22, 12
xtr_ WHs ι17, 11