Chapter 11

Domain Models of "The Market" — in Preparation for $\mathcal{E}$–Transaction Systems

Dines Bjørner

Informatics & Mathematical Modelling
Technical University of Denmark
DK–2800 Kgs.Lyngby, Denmark
db@imm.dtu.dk

Abstract

By an $\mathcal{E}$–Transaction System we shall understand a computer & communications–based system which support, and, in parts automates exchanges of action–invoking local state–changing messages (ie., transactions) between a wide variety of actors (traders). By "The Market" we shall first understand a structure of consumers, retailers, wholesalers and producers — ie., the traders. Later we shall extend our notion of "The Market".

We present informal English language descriptions (narractives) and formal models of a "Market" concept. What generally characterise traders are the kind of interactions they engage in: Issuing inquieries and offering quotations, placing and accepting orders, effecting and accepting deliveries, posting and paying invoices, etc. Traders dynamically form 'supply chains'. Any trader may, potentially, over different interactions, act any one of the trader rôles listed earlier. We then "lift" the market to include agents[1] (acting on behalf of any one of the traders listed earlier), and brokers (acting on behalf of 'sequences' of two or more "adjacent" traders while basically engaging in the kind of transactions enumerated above). We finish by first making some remarks on the use of the model presented as a basis for requirements development. Then we "lift" the notion of traders to not just representing pairs of buyers and sellers in a conventional supply chain of merchandise, but any pairs of (institutional) $\mathcal{G}$overnment to $\mathcal{G}$ ($\mathcal{G}2\mathcal{G}$), $\mathcal{G}$ to (private or public) $\mathcal{B}$usiness, $\mathcal{G}$ to (individual, human) $\mathcal{C}$itizen, $\mathcal{B}2\mathcal{G}$, $\mathcal{B}2\mathcal{B}$, $\mathcal{B}2\mathcal{C}$, $\mathcal{C}2\mathcal{G}$, $\mathcal{C}2\mathcal{B}$, and $\mathcal{C}2\mathcal{C}$ transaction possibilities.

[1] We stress that the notion of 'agents'used here is not the same notion as it is currently en vogue in the AI community. But, as we point out in a concluding section, Section 11.5.2, the two relates.

11.1 Introduction

The aim of this paper is to contribute to precise understandings of what is meant by $\mathcal{E}$–Commerce, –Business, –Government, etc. We suggest to achieve this goal by "imposing" orderly, formal techniques–based processes, first of analysis, later of synthesis, on the development of software for "such" $\mathcal{E}$ applications.

11.1.1 The Setting

Some facts: Before software and computing systems can be developed, their requirements must be reasonably well understood. Before requirements can be finalised the application domain, as it is, must be fairly well understood.

Some opinions: In today's software and computing systems development very little, if anything is done, we claim, to establish fair understandings of the domain. It simply does not suffice, we further claim, to record assumptions about the domain when recording requirements. Far more radical theories of application domains must be at hand before requirements development is even attempted.

In this presentation we advocate a strong rôle for domain engineering. We argue that domain descriptions are far more stable than are requirements prescriptions for support of one or another set of domain activities. We further argue, that once, given extensive domain descriptions, it is comparatively faster to establish trustworthy and stable requirements than it is today. We finally argue that once we have a sufficient (varietal) collection of domain specific, ie. related, albeit distinct, requirements, we can develop far more reusable software components than using current approaches.

Thus, in this contribution we shall reason, at a meta-level, about major phases of software engineering: Domain engineering, requirements engineering, and software design.

In other papers[2] we suggest a number of domain and requirements engineering as well as software design concerns, stages and steps, notably: Domain facets, including domain intrinsics, support technologies, management & organisation, rules & regulations, as well as human behaviour. respectively requirements: Domain requirements, interface requirements, and machine requirements. Specifically: Domain requirements projection, determination, extension, and initialisation.

11.1.2 The Background

We mention two facets of the background upon which this paper is put forward.

[2] Most recent, but survey papers are: [Bjø02a, Bjø02b, Bjø02c].

Our main research direction is that of trying to come to grips with "*What is software engineering ?*". The enumerations of the previous section provides a glimpse into how we intend to answer that question: Software development as a three phase "affair" consisting of the (possibly overlapping and re–iterated) developments of domain descriptions, requirements prescriptions and software designs — with each of these phases typically consisting of up to several stages of developments ("refinements", "transformations"), and with stages typically consisting of several steps. Our quest, along this first facet of methodology, is one of discovering (identifying), researching, and developing principles and techniques for sensible dispatches of phases, stages and steps.

A derivative "research" direction is then that of applying these proposed principles and techniques to the building of specific domain theories. Just as we have the theories of mechanics, electricity, thermodynamics, etc., of physics, so we would like to see theories emerge from domain models of man–made universes such as transport and logistics, financial service industry, health–care, etc.

The present paper constitutes a second beginning (cf. [Bjø01b]) with respect to a possible theory of some form of "market" domain. We can refer to similar such "first beginnings" for a railway domain [Bjø94, BLP94, BGP99, BPG99a, BPG99b, Bjø00b, Bjø03i], a resource management domain [Bjø00a], a projects (ie., a project management) domain [Bjø99b, Bjø03h], a sustainable development domain [Bjø99a], a logistics domain [Bjø03e], a health–care domain [Bjø03d], a fisheries (industry) domain [Bjø98], a financial service–industry domain [BRH98, Bjø03c], and many others. The present stage of most of these "first beginnings" is that they are all somewhat sketchy: Their ongoing development is pursued as much for the reason of testing development method principles and techniques — and discovery of new or alternative such, as for the reason of these conjectured, respective domain theories.

In general, our second research facet aims at covering various examples of the concept of infrastructure components and has, as its objective, that of eventually being able to characterise the infrastructure concept [Bjø95, Bjø96, Bjø01a, Bjø02d][3].

11.1.3 The Structure of the Paper

Sections 11.2–11.3, besides constituting the "bulk" of this paper, also is in the form of a of the documentation that we favour for the early stages of development of software. Section 11.2 briefly outlines a project of developing a domain model leading up to $\mathcal{E}$–Commerce. Section 11.3 presents main components of the formal documentation of a domain model for "the market". Section 11.4 speculates on possible requirements for "$\mathcal{E}$–Business", where, from a limited view of "the market" as consisting of traders in the private enterprise sphre

[3] We present an embarrased apology for exclusively citing own reports and publications.

and of consumers, we generalise by re–interpreting the market interactions of inquieries, quotes, declinations, refusals, orders, confirmations, deliveries, acceptances, invoicings, payments, acknowledgements, returns, and refunds (soon to be identified) — with slight "re–labellings" — into interactions between and within $\mathcal{G}$overnment, $\mathcal{B}$usinesses, and $\mathcal{C}$onsumers, that is: $\mathcal{G}2\mathcal{G}$, $\mathcal{G}2\mathcal{B}$, $\mathcal{G}2\mathcal{C}$, $\mathcal{B}2\mathcal{G}$, $\mathcal{B}2\mathcal{B}$, $\mathcal{B}2\mathcal{C}$, $\mathcal{C}2\mathcal{C}$, $\mathcal{C}2\mathcal{B}$, and $\mathcal{C}2\mathcal{G}$.

The next two sections are thus presented as we would develop and present main documents of software development, from initial, pragmatic project documents, via domain modelling documents to, as here, requirements documents. We do not present all documents, but comment briefly on those left out.

11.2 Initial Project Documents

The scope[4] of our concern is the market of consumers, retailers, wholesalers and producers. The span[5] of our concern is the set of those activities of the market than can be mechanised using computers and communication.

11.2.1 Needs and Ideas

An overall need is perceived for deploying computers and communication (including the Internet) to support a number of `market transactions`. The main ideas are to support those of *inquiring* as to which *products* are available, their `price` and `delivery conditions`, *replying* to such *inquieries*, *ordering* `products`, *confirming*, *delivering*, *accepting*, *invoicing*, *paying*, *rejecting*, *returning*, and *refunding* such `orders`, respectively `deliveries`.[6]

Further ideas are to provide speech act–based software "agents"[7] [Pet02] focused around modal logics — not covered in the present paper.

11.2.2 The Design Brief

The overall design brief is to provide a sketch of a domain model of the market. The model shall be both informally and formally described. And the model shall enable requirements development.

[4] We use the term 'scope' in the sense of [Jac95].

[5] We use the term 'span' in the sense of [Jac95].

[6] Observe the use of three type fonts: sans serif for document types, *slanted* for domain operations, and `tele type` for domain entities.

[7] Cf. Footnote 1: We are now, here, referring to what is traditionally viewed as an AI concept. See also Section 11.5.2.

11.3 The Domain

By a domain we understand an area of human (or other) activity. Examples are: "the railway domain", "the health–care domain", the domain of the "financial service industry", etc. Elsewhere the composite term 'application domain', where 'application' signals that the person who utters the composite term intends to apply computers & communication to problems of the domain.

We present our understanding of a domain through documents. Software development is focused on the development of (semantically meaningful) documents.

11.3.1 Informative Documents

We normally operate with three kinds of documents: Informative (ie., pragmatic), descriptive (ie., syntactic and semantic), respectively analytic (validation & verification) documents. We shall exemplify only the first two kinds of documents, and then only some of these.

Needs and Ideas

We need both informal and formal decriptions. Informal descriptions serve validation purposes: *Are we dealing with the right "thing" ?* [Boe81]. Formal descriptions serve verification purposes: *Are we getting the "thing" right ?* [Boe81].

The ideas are to let the informal descriptions "follow" the formal models, and to let the formal models be expressed in the `Raise` Specification Language (`RSL`) [GHH+92, GHH+95].

Etcetera: *An actual 'needs and ideas' document may contain further details !*

The Design Brief

The domain design brief is to provide descriptive and analytic documents: (i) Rough sketches of the concepts of the market, (ii) the formulation of abstract market concepts (based on (iii) analyses of the rough sketches), a (iv) terminology for the market domain as it applies to the given problem, and a combined (v+vi) narrative (ie., informal) and formal description of the market.

Etcetera: *An actual 'domain design brief' document will (shall) contain further details !*

On the Contract

A contract describes *'parties to the contract'*, *'the subject matter'* and *'considerations'*.

The contractual relationship potentially involves the following market stake–holders (ie., 'parties'): consumers, retailers, wholesaler, producers, agents,

brokers, financing (payment, etc.), and delivery (ie., transport) services — as well as the following computing systems stake–holders: domain engineers. The 'subject matter' is described in the overall design brief and in the domain design brief. The contract states the following 'considerations': (a) that the market stake–holders shall validate the domain descriptions provided by the domain engineers, and (b) that the latter shall have "such–and–such free" access to the market stake–holders (in order to obtain the necessary and sufficient domain knowledge).

Etcetera: *An actual 'contract' document will (shall) contain further details !*

11.3.2 Descriptive Documents

We present a fair selection of parts of descriptive documents.

A Rough Sketch and its Analysis

We first bring an example rough sketch, then its analysis. After that we bring both rough sketches and analyses.

Buyers and Sellers: First a rough sketch of what is meant by buyers and sellers, then its analysis.

Rough Sketch: Consumers, retailers, wholesalers and producers form the major "players" in the market.

A consumer may inquire with a supposedly appropriate retailer as to the availability of certain products (cum merchandise): Their price, delivery times, other delivery conditions (incl. quantity rebates), and financing (ie., payment). A retailer may respond to a consumer inquiry with either of the following responses: A quote of the requested information, or a (courteous) declination, or a message that the inquiry was misdirected (refusals), or the retailer may decide to not, or fail to, respond ! A consumer may decide to order products with a supposedly appropriate retailer, whether such an order has been or has not been preceded by a related inquiry. The retailer may respond to a consumer order with either of the following responses: Confirming, declining or "no–response", with a confirmation being following either by a delivery, or no delivery — or the retailer may just provide a delivery, or inform the consumer that a back–order has been recorded: The desired products may not be in store, but has been (or will be) ordered from a wholesaler — for subsequent delivery. A delivery may deliver the ordered or some other, not ordered, products ! The consumer may decide to not accept, or to accept a delivery. The retailer may invoice the consumer before, at the same time as, or after delivery. The consumer may pay, or not pay an invoice, including performing a payment based on no invoice, for example at the same time as placing the order. The retailer may acknowledge payments. The consumer may

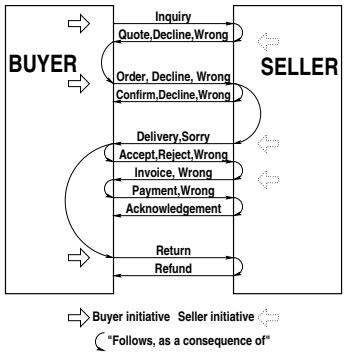
find faults with a previously accepted delivery and return that (or, by mistake, another) delivery. The retailer may refund, or not refund such a return.

Analysis: Based on an analysis of the above rough sketch we suggest to treat market interactions between retailers and wholesalers, and between wholesalers and producers in exactly the same way as interactions between consumers and retailers. That is: we observe that retailers acts as (a kind of) "consumers" vis-a–vis wholesalers (who, similarly acts as retailers).

We thus summarise the interactions into the following enumeration: inquieries, quotes, declinations, refusals, orders, confirmations, deliveries, acceptances, invoicings, payments, acknowledgements, returns, and refunds.

Figure 11.1 attempts to illustrate possible transaction transitions between buyers and sellers.

Figure 11.1: Buyer / Seller Protocol

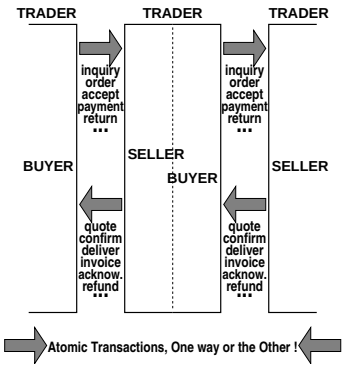


Traders: As a consequence of the analysis we shall "lift" the labels 'consumer', 'retailer', 'wholesaler' and 'producer' into the labels 'buyer' and 'seller'. And we shall use the term 'trader' to cover both a buyer and a seller. Since the consumers and producers mentioned in the rough sketch above may also act as any of the other kinds of traders, all will be labelled traders.

Figure 11.2 on the next page attempts to show that a trader can be both a buyer and a seller. Thus traders "alternate" between buying and selling, that is: Between performing 'buy' and performing 'sell' transactions.

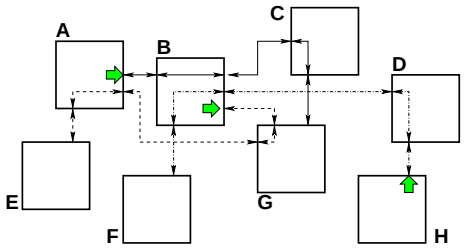
Supply Chains: Figure 11.3 on the following page attempts to show "an arbitrary" constellation of buyer and seller traders. It highlights three supply

Figure 11.2: Trader=Buyer+Seller



chains. Each chain, in this example, consists, in this example, of a "consumer", a retailer, a wholesaler, and a producer.

Figure 11.3: A Network of Traders and Supply Chains



A collection, a set, of traders may thus give rise to any set of supply chains, with each supply chain consisting of a sequence of two or more traders. Supply chains are not static: They form, act and dissolve. They are a result of positive inquieries, orders, deliveries, etc.

'Likeness', 'Kinds', 'Adjacency', and 'Supply Chain Instances': As a result of analysis we identify a need for some abstract concepts: 'likeness', 'kinds', and 'supply [chain] instances' (where [...] expresses that we can omit the ...).

Like traders are of the same 'kind', where the 'kind' of a trader is either

consumer, retailer, wholesaler, or producer.

We can also speak of the 'kind' of a transaction.

The 'kind' of a transaction is either than of inquiery, quote, declination, refusal, order, confirmation, delivery, acceptance, invoice, payment, acknowledgement, return, or refund.

There may be chains of one or more wholesalers: Global, regional, national, or, within a state, area wholesalers. We therefore allow for the following kinds of adjacent traders: (consumer,retailer), (retailer,wholesaler), (wholesaler,wholesaler), and (wholesaler,producer).

A supply [chain] instance is a specific and related occurrence of two or more transactions. The following is an elaborate supply chain instances — where we omit reference to the specifics by only mentioning the transaction kinds: (i) *inquiry* (consumer to retailer), → *inquiry* (retailer to wholesaler), → *quote* (wholesaler to retailer), → *quote* (retailer to consumer), → *order* (consumer to retailer), → *order* (retailer to wholesaler), → *order* (wholesaler to producer), → *confirm* (producer to wholesaler), → *confirm* (wholesaler to retailer), → *confirm* (retailer to consumer), → *delivery* (producer to wholesaler), → *acceptance* (wholesaler to producer), → *delivery* (wholesaler to retailer), → *acceptance* (retailer to wholesaler), → *delivery* (retailer to consumer), → *acceptance* (consumer to retailer), → *invoice* (retailer to consumer), → *payment* (etc., the reader fills in possible details), → *acknowledge*, → *invoice*, → *invoice*, → *payment*, → *payment*, → *acknowledge*, → *acknowledge*, → *return*, and → *refund*.

Agents and Brokers: Although not formalised explicitly in the present paper we discuss the concepts of brokers and traders. We then, later on, "reduce" agents and brokers to become like traders are.

Agents: An agent,[8] α, in the domain, is any human or any enterprise, including media advertisement, who, or which, acts on behalf of one trader, t_1, in order to mediate possible purchase (or sale) of goods from another trader, t_2. So t_1 may be a consumer, or a retailer, or a wholesaler who, through α acquires goods from t_2 who, respectively, is a retailer, a wholesaler and a producer. Or t_1 may be a retailer, or a wholesaler, or a producer who, through α sells to t_2 who, respectively, is a consumer, a retailer, and a wholesaler. One can generalise the notion of agents to such who (or which) acts on behalf of a group of like traders to "reach" a corresponding group of like and adjacent traders.

Figure 11.4 on the next page attempts to show a buyer–agent (lefthand figure), respectively a seller–agent (righthand figure). The buyer–agent "searches" the market for suitable sellers of a specific product. The seller–agent searches the market for suitable buyers of a specific product.

[8] We remind the reader of Footnote 1.

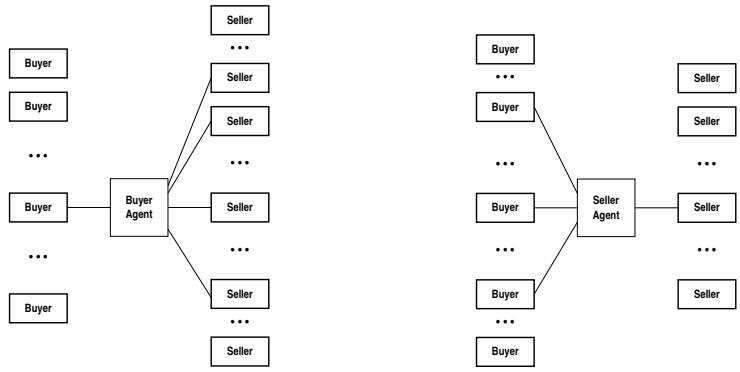


Figure 11.4: Buyer and Seller Agents

The idea is that the two kinds of agents behave like buyers, respectively like sellers: The buyer–agent "learns" from the buyer[9] about what is to be inquired, is instructed when to order, etc. The buyer–agent then iterates over a set of sellers known to meet inquired expectattion[10]

Similarly for seller–agents (the right–hand side of Figure 11.4).

Brokers: A broker, β, in the domain, is any human or any enterprise, including media advertisement, who, or which, acts on behalf of two (or more, respectively) adjacent groups of like traders, bringing them together in order to effect instances of supplies.

Figure 11.5 on the facing page attempts to diagram a broker mediating between m buyers and n (adjacent kind) sellers.

The idea is that a combination of buyer and seller searches, and hence a combination of the buyer– and seller–agent behaviours are needed.

Brokers can span more than one stage.

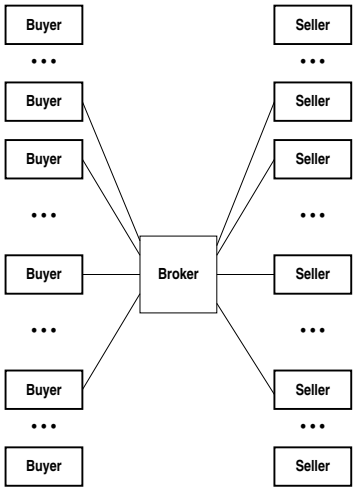
Figure 11.6 on page 12 attempts to diagram a broker mediating between m_1 consumers, m_2 retailers, m_3 wholesalers and m_4 producers — subsets of all the known such.

The three sets of dashed lines in the three vertical "stems" of the broker shall designate "local" brokerage between adjacent pairs of buyers and sellers. The set of dashed lines in the horisontal branch of the broker shall designate overall, "global" brokerage between all parties.

[9] This is designated by the single line (between the Buyer and the Buyer Agent rectangles) of the left–hand side of Figure 11.4.

[10] This is designated by the mostly slated lines (between the Buyer Agent and the Seller rectangles) of the left–hand side of Figure 11.4.

Figure 11.5: A Simple ("One Stage") Broker

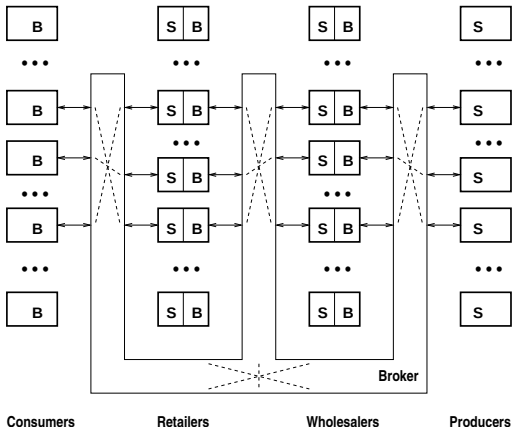


The aim of the mediation is to create a consortium of subsets of consumers, retailers, wholesalers and producers. The objective of the consortium is, like a *"Book of the Month Club"*, to create a stable set of complete supply chains for a given set of products.

As for simple brokers we shall (ever so briefly) argue that the same iterated searching of resolution protocols and mechanisms as for agents are to be deployed, and that these are based on the all the transaction kinds as first sketched.

Catalogues: An important concept of the market is that of a catalogue. It may be implicit, or it may exist explicitly. A catalogue, in a widest sense of that term, is any form of recording that lists what merchandise is for sale, its price, conditions of delivery, payment, refund, etc. An ordinary retailer — your small neigbourhood *"Mom & Pop"* store — may not be able to display a catalogue in the form of, for example, a ring binder each of whose pages lists, in some order, the merchandise by name, order number, producer, etc., and which records the above mentioned forms of information. But, from the shelves of that store one can "gather" that information. For wholesalers and producers we can probably assume such more formal catalogues. But, as a concept, we can in any case speak of catalogues. And hence we can speak of such concepts as *searching* in a catalogue, marking entries as being *out of stock, how many sold, when, to whom* etc.

Figure 11.6: A Multiple (here: Three) Stage Broker



The Transactions: We have, above, just hinted at the kind of transactions, to wit: inquiry, quote, declination, refusal, order, confirm, delivery, acceptance, invoice, payment, acknowledge, return, and refund. Instead of treating them in more detail — as part of a narrative — we relegate, for the sake of brevity, such a treatment to the terminology section, next, and to the formalisation, following.

• • •

This completes our, lengthy, rough sketch of "The Market" domain. It was made deliberately long in order to make the point: That rough sketching is an important process, and that rough sketches serve a purpose — as we shall subsequently see.

Terminology

In any development, in any (domain, requirements or software design) phase of such a development, it is judged important to start, after rough sketching, by establishing a terminology, by adhering to it throught the develpment, and, when eed arises, to update the terminology while securing that previous uses ("adherencves") remain valid. Failure to follow this 'terminology codex' usually results in term confusion and thus leads to low quality software.

We sketch a rather informal terminology. That is: A proper terminology must be far more pedantic in its precision, consistency of term usage, and in its completeness.

1. **Accept:** A delivered merchandise is inspected by the buyer and found acceptable.
2. **Acknowledge:** Receipt of payment has been recorded by a seller and the buyer (the payee) is given an acknowledgement.
3. **Agent:** A surrogate buyer or a surrogate seller who acts on behalf of either a buyer, or a seller, and otherwise performs the following protocols of transactions: ....
4. **Back–order:** An order whose delivery cannot be ffected immediately.
5. **Broker:** a surrogate both buyer & seller who acts on behalf of both buyers and sellers, and otherwise performs the following protocols of transactions: .....
6. **Buy:** A buy transaction is the same as an order transaction.
7. **Buyer:** A buyer is any trader who may issue an inquiry or an order transaction.
8. **Catalogue:** When inquiring or ordering, an implicit reference is made to a catalogue — something that records products bought or sold, their prices, delivery consitions, etc. — ie., something that the seller is supposed to have and to be able to quote from, or from which crucial information for an invoice is culled, or which the buyer ... etc.
9. **Confirm:** If an order can be effected, either now, or as later (as a back-order), then a confirmation can be issued to that effect.

Etcetera:

10. **Consumer:** ...
11. **Decline:** ...
12. **Deliver:** ...
13. **Inquire:** ...
14. **Inventory:** ...
15. **Invoice:** ...
16. **Market:** ...
17. **Merchandise:** ...
18. **On–order:** ...
19. **Order:** ...
20. **Pay:** ...
21. **Price:** ...
22. **Price–list:** ...
23. **Producer:** ...
24. **Product:** ...
25. **Quantity:** ...
26. **Refund:** ...
27. **Reject:** ...
28. **Retailer:** ...
29. **Return:** ...
30. **Sell:** ...
31. **Seller:** ...
32. **Service:** ...
33. **Store:** ...
34. **Supply Chain:** ...
35. **Trader:** ...
36. **Warehouse:** ...
37. **Wholesaler:** ...

...

&c. ...

This completes our rather short terminology. It is large enough, we think, to have partially made our point — as stated in the opening paragraph of this section. A full justification for 'terminologisation' can only be made spme time after the end of a project.

Narrative and Formal Model

We combine, into one document, the informal description and the formal description of the domain of traders. We describe only the basic protocols for inquiry, quote, order, confirmation, delivery, acceptance, invoice, payment, etc. transactions. We thus do not describe agents and brokers. We leave that to a requirements modelling phase.

Please observe the extensive need for expressing selection of and responses to transactions non–deterministically. In the real world, ie., in the domain, all is possible: Dilligent staff will indeed follow–up on inquiries, orders, payments, etc. Loyal consumers will indeed respond likewise. But sloppy such people may not. And outright criminals may wish to cheat, say on payments or rejects. And we shall model them all. Hence non–determinism.

Se initial remarks on determination of undesirable non–determinisms in Section 11.4 on 'Requirements'.

Formalisation of Syntax:

```
type
  Trans == Inq|Ord|Acc|Pay|Rej
         | Qou|Con|Del|Acc|Inv|Ref
         | NoR|Dec|Mis
```

The first two lines above list the 'buyer', respectively the 'seller' initiated transaction types. The third line lists common transaction types.

In the domain we can speak of the uniqueness of a transaction: *"it was issued at such–and–such time, by such–and–such person, and at such–and–such location,"* etcetera.

U below stand for (supposedly, or possibly) unique identifications, including time, location, person, etc., stamps (T, P, L), Sui (where i=1,2) stands for surrogate information, and MQP alludes to Merchandise identification, Quantity, and Price.

```
type
  U, M, Q, P, T, Su1, Su2, Inf
  Inq :: MQP × U
  MQP == mk(m:M,q:Q,p:P,...)
  Quo :: ((Inq|Su1) × Inf) × U
  Ord :: Qou|Su2 × U
  Con :: Ord × U
  Del :: Ord × U
  Acc :: Del × U
  Inv :: Del × U
  Pay :: Inv × U
  Rej :: Del × U
  Ref :: Pay × U
  NoR :: Trans × U
  Dec :: Trans × U
  Mis :: Trans × U
value
  obs_T: U → T
```

The above defines the syntax of classes of disjoint transation commands, of the abstract form mk_Name(kind,u) where Name is either of Inq, Quo, Ord, Con, Del, ... or Mis.

An inquiry:Inq consists of a pair, some (desired) merchandise, (desired) quantity and (desired) price information, and a supposedly unique identification (of time, location, person, etc.) of issue – this "mimics" a consumer inquiry of the form *"I am in the market for such–and–such merchandise, in such–and–such a quantity, and at such–and–such prices. What can you offer ?"*..

An quote:Quo either refers to the inquiry in which the quote is a response or presents surrogate information — typically (where the seller takes the initiative to advertise some merchandise and then) of a form similar to an inquiry: *"If you are in the market for such–and–such merchandise, in such–and–such a quantity, and at such–and–such prices, then here is what we offer"*.

information:Inf is then what is offered.

In general we model, in the *domain,* a "subsequent" transaction by referring to a complete trace of (supposedly) unique time, location, person, etc., stamped transactions. Thus, in general, a transaction "embodies" the transaction it is a manifest response to, and time, location, person, etc. of response.

Do not mistake this for a requirement. A requirement may or may not impose unique identification wrt. time and location and person etc. Therefore we do not detail U. Nor do we actually say that no two transactions can be issued with the same uniqueness.

Formalisation of Market Interactions: "The Market" consist of n traders, whether buyers, or sellers, or both; whether additionally agents or brokers. Each trader τ_i is able, potentially to communicate with any other trader:

$$\{\tau_1, \ldots, \tau_{i-1}, \tau_{i+1}, \ldots, \tau_n\}.$$

We omit formal treatment of how traders come to know of one another. An arbiter for such information is just like a trader. Other traders sell information about their existince to such an arbiter. Thus no special formal treatment is necessary.

We focus on the internal and external non–determinism which is always there, in the *domain,* when transactions are selected, sent and received.

Our model is expressed in a variant of CSP, as "embedded" in RSL [GHH+92].

type
[0] Θ, MSG
[1] Idx = {| 1..n |}

value
[2] sys: (Idx $\xrightarrow[m]{}$ Θ) → **Unit**
[3] sys(mθ) ≡ || { tra(i)(mθ(i)) | i:Idx }

channels {tc[i,j]:MSG | i,j:Idx • i< j}

value
[4] tra: i:Idx → Θ → **in** {tc[j,i]|j:Idx•i≠j} **out** {tc[i,j]|j:Idx•i≠j} **Unit**
[5] tra(i)(θ) ≡ tra(i)(nxt(i)(θ))

[6] nxt: i:Idx → Θ → **in** {tc[j,i]|j:Idx•i≠j} **out** {tc[i,j]|j:Idx•i≠j} Θ
[7] nxt(i)(θ) ≡
[8] **let** choice = rcv ⌈⌉ snd **in**
[9] cases choice **of** rcv→receive(i)(θ), snd→send(i)(θ) **end end**

(0) Θ is the type space that any trader may span. MSG is type space of all messages that can be exchanged between traders (ie., over channels). We

detail neither Θ nor MSG: In the "real world", ie., in the domain, all is possible. Determination of Θ and MSG is usually done when "deriving" the functional requirements from the domain model. (1) Idx is the set of n indices, where each trader has a unique index. We do not detail Idx. That usually is done as late as possible, say during code implementation. (2) The system initialises each trader with a possibly unique local state (from its only argument). (3) The system is the parallel combination of n traders. (4) A trader has a unique, constant index, i, and is, at any moment, in some state θ. (4) Traders communicate (both **in**put and **out**put) over channels: tc[i,j] — from trader i to trader j. (5) Each trader is modelled as a process which "goes on forever", (5) but in steps of next state transitions. (8) The next state transition non—deterministically (internal choice, ⌈⌉) "alternates" between (9) expressing willingness to receive, respectively desire to send.

In "real life", ie. in the domain, the choice as to which transactions are pursued is non–deterministic. And it is an internal choice. That is: The choice is not influenced by the environment.

We model receiving as something "passive": No immediate response is made, but a receive state component of the trader state is updated. A trader that has decided to send (something), may non–deterministicaly decide to inspect the receive component of its state so as to ascertain whether there are received transactions pending that ought or may be responded to.

The update_rcv_state invokes further functions.

```
receive: i:Idx → Θ → in {tc[j,i]|j:Idx•i≠j} Θ
receive(i)(θ) ≡
    [] {let msg=tc[j,i]? in  update_rcv_state(msg,j)(θ) end | j:Idx}
```

Once the internal non–deterministic choice (⌈⌉) has been made ((8) above): Whether to receive or send, the choice as to whom to 'receive from' is also non–deterministic, but now external ([]). That is: receive expresses willingness to receive from any other trader. But just one. As long as no other trader j does not send anything to trader i that trader i just "sits" there, "waiting" — potentially forever. This is indeed a model of the real world, the domain. A subsequent requirement may therefore, naturally, be to provide some form of time out. A re–specification of receive with time out is a correct implementation of the above.

```
[2]  update_rcv_state: MSG × i:Idx → Θ → Θ
[3]  update_rcv_state(msg,j)(θ) ≡
[4]    cases obs_Trans(msg) of
[5]      mk_Del(_,_)
[6]        → upd_rcv(msg,j)(upd_del(msg,j)(θ)),
[7]      mk_Ret(_,_)
[8]        → upd_rcv(msg,j)(upd_ret(msg,j)(θ)),
```

```
[9]    _ → upd_rcv(msg,j)(θ)
[10]   end
```

(2) any message recieval leads to an update of a 'receive' component of the local trader state (upd_rec). (5–6) If the received "message" constitutes a (physical package) delivery, then a 'Merchandise' component of the local trader state is first updated (deposit_delivery). (7–8) If the received "message" constitutes the return (of a physical package), then the 'merchandise' component of the local trader state is first updated (remove_return).

```
[0]  upd_rec(msg,j)(θ) ≡ deposit_trans((sU(msg),j),msg)(cond_rec(msg,j)(θ))
[1]  upd_del(msg,j)(θ) ≡ deposit_delivery((sU(msg),j),msg)(θ)
[2]  upd_ret(msg,j)(θ) ≡ remove_return((sU(msg),j),msg)(θ)

[3]  cond_rcv(msg,j)(θ) ≡
[4]    if intial_trans(msg)(θ)
[5]      then θ
[6]      else remove_prior_trans(sU(msg),j)(θ) end

       sU: Trans → U, sU(_,u) ≡ u
```

(0) The upd_rec operation invokes the cond_rec operation and then extends the possibly new state by depositing the argument message under the unique identification and message–sending trader identification. (3–6) The cond_rec operation examines ((4) initial_trans) whether the received message is a first such, ie., "contains" no prior transactions, or whether it contains such prior transactions. In this latter case (6) the prior transaction may be conditionally removed (remove_prior_trans) — this is not shown here, but commented upon below.

```
[0]  send: i:Idx → Θ → in {tc[i,j]|j:Idx•i≠j} Θ
[1]  send(i)(θ) ≡
[2]    let choice = ini ⌈ res ⌈ nor in
[3]    cases choice of
[4]      ini → send_initial(i)(θ),
[5]      res → send_response(i)(θ),
[6]      nor → remove_received_msg(θ) end end
```

Either a trader, when communicating a transaction chooses (2,4) an initial (ini) one, or chooses (2,5) one which is in response (res) to a message received earlier, or chooses (2,6) to not respond (nor) to such an earlier message The choice is again non–deterministic internal (2). In the last case (6) the state is thus non–deterministically internal choice updated by removing the, or an earlier received message.

Note that the above functions describe the internal as well as the external non–determinism of protocols. We omit the detailed description of those functions which can be claimed to not be proper protocol description functions — but are functions which describe updates to local trader states. We shall, below, explain more about these state–changing functions.

```
send_initial: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_initial(i)(θ) ≡
    let choice = buy ⌈⌉ sell in
    cases choice of
        buy → send_init_buy(i)(θ),
        sell → send_init_sell(i)(θ) end end

send_response: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_response(i)(θ) ≡
    let choice = buy ⌈⌉ sell in
    cases choice of
        buy → send_res_buy(i)(θ),
        sell → send_res_sell(i)(θ) end end
```

In the above functions we have, perhaps arbitrarily chosen, to distinguish between buy and sell transactions. Both send_initial and send_response functions — as well as the four auxiliary functions they invoke — describe aspects of the protocol.

```
send_init_buy: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_init_buy(i)(θ) ≡
    let choice = inq ⌈⌉ ord ⌈⌉ pay ⌈⌉ ret ⌈⌉ ... in
    let (j,msg,θ′) = prepare_init_buy(choice)(i)(θ) in
    tc[i,j]!msg ; θ′ end end

send_init_sell: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_init_sell(i)(θ) ≡
    let choice = quo ⌈⌉ con ⌈⌉ del ⌈⌉ inv ⌈⌉ ... in
    let (j,msg,θ′) = prepare_init_sell(choice)(i)(θ) in
    tc[i,j]!msg ; θ′ end end
```

prepare_init_buy is not a protocol function, nor is prepare_init_sell. They both assemble an initial buy, respectively sell message, msg, a target trader, j, and update a send repository state component.

```
send_res_buy: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_res_buy(i)(θ) ≡
    let (θ′,msg)=sel_update_buy_state(θ), j=obs_trader(msg) in
```

```
    let (θ'',msg') = response_buy_msg(msg)(θ') in
    tc[i,j]!msg'; θ'' end end

send_res_sell: i:Idx → Θ → out {tc[i,j]|j:Idx•i≠j} Θ
send_res_sell(i)(θ) ≡
    let (θ',msg)=sel_update_sell_state(θ), j=obs_trader(msg) in
    let (θ'',msg') = response_sell_msg(msg)(θ') in
    tc[i,j]!msg'; θ'' end end
```

sel_update_buy_state is not a protocol function, neither is sel_update_sell_state. They both describe the selection of a previously deposited, buy, respectively a sell message, msg, (from it) the index, j, of the trader originating that message, and describes the update of a received messages repository state component. response_buy_msg and response_sell_msg both effect the assembly, from msg, of suitable response messages, msg′. As such they are partly protocol functions. Thus, if msg was an inquiry then msg′ may be either a quote, decline, or a misdirected transaction message. Etcetera.

On Operations on Trader States: We have left a number of trader state operations undefined. Below we giver their signature and otherwise comment on them informally. The reason for not formally defining them is simple: Since we are modelling the domain, and since, in the domain, these updates are typically performed by humans, and since these humans are either dilligent, or sloppy, or delinquent, or outright crominal in the despatch of their duties we really cannot define the operations as we would really like to see them despatched — namely diligently.

```
value
    deposit_trans: (U × Idx) × MSG → Θ → Θ
    deposit_delivery: (U × Idx) × MSG → Θ → Θ
    remove_return: (U × Idx) × MSG → Θ → Θ
    initial_trans: MSG × Idx → Θ → Bool
    remove_prior_trans: U × Idx → Θ → Θ
    remove_received_msg: Θ → Θ
```

The above operations have all basically been motivated earlier. The deposit_trans unconsditionally deposits a received message, for example in a part of the local trader state that could be characterised as a repository for received transactions. That repository may have messages identified by the sender and the unique identificator. To specify so is not a matter of binding future requirements and therefore also not future implementations. It just models that one can, in the domain "talk" about these things.

An initial_transaction is one which does not contain prior transactions, that is: Is one which is either an inquiry tranaction or contains surrogates (Sur1, Sur2).

To remove a prior transaction models that people may no longer keep a record of such a transaction — since it is embedded in the message in response to which this removal is invoked. We do not show the details of removal, but expect a model to capture that such prior transactions need not be removed. In other words: The removal may be internal non–deterministically "controlled".

remove_received_msg unconditionally removes a message: This models that people and institutions (internal non–deterministically) may choose to ignore inquieries, quotations, orders, confirmations, deliveries, etc.

prepare_init_buy: Choice → Idx → Θ → Idx × MSG × Θ
prepare_init_sell: Choice → Idx → Θ → Idx × MSG × Θ

The above operations internal non–deterministically chooses which prior transactions to respond to.

obs_trader: MSG → Idx

No matter which transaction (ie., message) one can always identify, say from the unique identificator, which trader originated that message. We do not specify how since that might bias an implementation.

For the sake of completeness we also state the signatures of remaining and previously described operations:

value
upd_rec: MSG × Idx → Θ → Θ
upd_del: MSG × Idx → Θ → Θ
upd_ret: MSG × Idx → Θ → Θ
cond_rcv: MSG × Idx → Θ → Θ

sel_update_buy_state: Θ → Θ × MSG
sel_update_sell_state: Θ → Θ × MSG

response_buy_msg: MSG → Θ → Θ × MSG
response_sell_msg: MSG → Θ → Θ × MSG

In summary: All operations on local trader states are, in the domain, basically under–specified. It will be a task for requirements to, as we shall call it, determine precise functionalities for each of these operations.

Discussion

As for local trader state operations, so it is for the possible sequences of transactions between "market players" (ie., the traders): They are all, in the above model, left "grossly" non–deterministic.

Those trader who initiate transactions towards othjer traders can be viewd as "clients", while those others are seen as "servers". Thus it is that we see that "clients" are characterisable by internal non–determinism, while "servers" are characterisable by external non–determinism.

It is now a task for requirements to determine the extent of non–determinisms and the more precise rôles of 'clients' and 'servers'.

11.3.3 Analytic Documents

We remind the reader that this main section (Section 11.3) of the current paper is structured and mostly presented in the form of actual development documents: Their sequence, interrelations and contents.

Therefore we should, in this section bring analytic documents. Space considerations make this impossible. Instead we briefly comment as to which kinds of development stages and steps, and thus their ensuing documents, would be necessary for a reasonably professional phase of domain engineering to have taken place.

Validation

Paraphrasing Barry Boehm, [Boe81], for a product to be "the right product" we must validate also its domain model.[11]

The domain model has been "extracted" from all relevant stake–holders of the domain: Owners, managers, workers, customers, regulatory agencies, service providers (financial institutions, logistics firms, etc.) of "the market" domain. Hence validation must be conducted with such stake–holders. The validation process basically "reads" the informal terminologies and narrative. Hence it is utterly important that they be relatively complete and concise.

Verification

Again, paraphrasing Barry Boehm, [Boe81], for a "product to be right" we must verify implicitly expressed properties of a domain model.[12]

[11] Validation is "repeated", as we shall assume below, in "deriving" the product requirements, and, from requirements, a, or the software design(s). We shall not cover software design in the current paper.

[12] Verification is "repeated" in all phases of development (ie., also for requirements design and for software design): "Between" phases, stages and steps correctness verification is established, and "within" stages and/or steps, implicitly expressed (ie., derivative or "assumed") properties are verified.

The verification process is, in contrast to the validation process, basically a formal one: Formally expressed predicates are claimed (ie., interpreted) to represent "internal", respectively correctness properties. These are then either proved or model checked, as well as they or tested for.

Towards Theories of Market Domains

Just as physicists keep studying "Mother Nature": The God–created world, so, undoubtedly we (or is it: us ?) mortals ought study the man–created infrastructure components such as transportation, logistics, "the market", health services, etc. The physicists keep coming up with new discoveries, sometimes new laws, sometimes fascinating properties that follow from these laws, but are hard to otherwise ascertain. Perhaps we domain engineers could discover laws, properties, etc., about the studied infrastructure components. In other words: Domain engineers should aim at establishing theories about the domains they model. Some of these domains, like "the market", are about the *flow of people, information, materials, and control.*

11.3.4 Discussion

Domain Facets

In this short presentation of an example 'development of a domain model' we have not structured that development, nor the model, as we normally do. Namely around the concepts of domain facets: *intrinsics, support technologies, management & organisation, rules & regulations,* and *human behaviour.*

We find that these "operators" that apply to the unanalysed domain, and which, in sequence, results in a domain model, are novel.

By *intrinsics* we understand the very basics of the studied domain. What is basic depends on which stake–holder is "viewing" the domain, what rôle they play.

The intrinsics of "the market" seems to be (i) the notions of traders and their interaction sequences: Chains as well as protocols; and (ii) the notion of "inspections" of, and updates to local trader states.

Our model, above, is basically a model of the intrinsics facets of "the market". We have only represented the views of consumers, retailers, wholesaler and producers. And we have, in fact, abstracted these in terms of traders. More need be done.

By *support technologies* we understand any technology that supports activities in the domain — as distinguished (ie., apart) from the people facet — which we treat separately (see below). Examples of support technologies for "the market" — before the introduction of $\mathcal{E}$–Transactions — are such things as

telephone, faxes, ordinary PC (including possibly ordinary $\mathcal{E}$–Mail) and other computing for accounting, purchasing, personnel administration, etc. Pls. observe that as long as no $\mathcal{E}$–Transaction support has been intriduced that part is not (yet) a supporting technology. It becomes part of the domain, and then has the facet type 'support technology' once it has been installed and taken into operation.

Our model, above, abstracts from support technologies. When we model that a consumer directs a query at a retailer, then we abstract how that is done. Whether by being personally, ie., physically present at the retailer's premises, or by phoning, or faxing, or sending an $\mathcal{E}$–Mail.

If we were to model such "communication" facilities, then we would also have to model their (un–)reliability, (in–)stability, etc. That is: That, as support technology they may fail. That is: We are not modelling how to achieve safety or security, but that safety or security may be compromised by support technologies upon which we cannot **depend.** Modal logics, including temporal logics, in particular those of the Duration Calculi [CH03], are presently our favourite formal tools in modelling support technologies.

By *management & organisation* we understand the way in which different stake–holders within an enterprise (ie., an infrastructure component) are structured and assumptions about who does takes which initiatives: Manager $\leftrightarrow$ subordinate protocols, etc.

In our model, above, we have not modelled real aspects of management & organisation. Such aspects could be: Certain staff in a retailer must approve of certain quotes to consumers; certain staff at a wholesaler have the obligation to make sure that merchandise stores are kept "reasonably full", etc. The notion of agents and brokers, as well as the notion of clearing house traders for information about "the players" in "the market", those notions also, to us, belong to the management and organisation facet of "the market".

By *rules & regulations* we understand two kinds of statements: Rules lay down guide–lines for human behaviour. Regulations stipulate what remedial actions should be instituted should a rule be broken. There are general rules and there are rules prescribed by specific stake–holders.

In our model, above, we have not expressed any rules, let alone any regulations. So that remains to be done. An example of a general rule of "the market" is: *Thou shall pay thy bills !* An example of a specific rule is: *"We give 30 days credit"*. An example of a regulation could, in this connection, be: *"If, upon the third 'Please pay' reminder you fail to do so, we shall invoke a debt collecting agency"*.

And by *human behaviour* we mean what the next section will deal with specifically.

"In the Domain All is Possible"

We have presented salient stages of development of a domain model for trading in the market.

Some readers may well claim: Well, have we really *"presented salient stages of development of a domain model for trading in the market"* ?

Those readers, typically, have expected us to describe "features" of "the market" which we would now claim are not really properly manifest properties in the domain. We remind the reader that *"in the domain all is possible"*.

In the domain there are dilligent stake–holders: People who despatch of their work as expected by others. But there are also sloppy stake–holders, delinquent, yes outright criminal people who set low standards for their work, or, by design, desire criminal conduct. And the domain model must, in some sense, model that.

We thus claim that the above model, to the extent that it really covers transactions, describe such a spectrum of behaviours. It does so by leaving indeterminate whether properly begun transaction sequences (from inquiry via quotation to ordering and delivery, etc.) are actually properly concluded. And it does so by not detailing any of the specific operations on the trader states.

11.4 $\mathcal{E}$–Transaction System Requirements

From now on, in the current paper, we shall be discursive: The reason is first that the concept of domain may be well understood, but that the importance of its precise informal and formal description is far from sufficiently appreciated — certainly not to the extent that software clients expects and software developers demand a clear, sufficiently extensive domain model.

We shall therefore examine the transition from domains to requirements.

We thus claim that in the past, and still, requirements are basically exprssed without any technically sound reference to any form of domain model.

Thus we need to examine such notions as needs, goals, and requirements, and, within these, the rôles that the concept of domain plays.

Also we observe that past and present treatments of the concept of requirements have missed two, to us very important points: *Namely, how might a well–structured set of requirements be formulaed;* and *what exactly should a requirements documentation contain ?* In much of the current literature on $\mathcal{E}$–Market (etc.) we find that *very little, if anything is done to really understand "the market", as a domain,* and that *serious, contemplative consideration of orderly development of well–structured requirements,* as a consequence, *is lacking.*

11.4.1 Needs, Goals and Requirements

"In the beginning there was the domain !" Then computers & communication came into being. Now needs[13] are "felt" for bringing about changes whose goal[14] it is (or whose goals are) to achieve certain effects. *"In the end there is again the domain, but now with computers, communication and software — such that the software satisfies requirements and, when used according to assumptions being met by the environment — actors of the domain — achieve the goals, fulfills the needs !"*

In our example, "the market", examples of low–level **needs** may be: A need for *relieving humans from many chores of buying and selling,* and/or for *saving costs on bureaucratic business processes in connection with checking and double–checking on order deliveries, invoicing and payments, etc.*, has been identified. A high–level **need** for a *computing system* is settled upon accordingly.

Commensurate with this identified high–level need some **goals** for the computing systems are therefore enunciated: *The computing system shall help bring about relief, savings,* etc.

In Section 11.2.1 we first introduced the concept of 'formulated needs'. There we linked it to the formulation of 'ideas' (by means if which the needs could possibly be implemented). Where do these 'ideas' come from ? Well, the simple answer is: The domain and the possible 'machine': The computing system including the desired software ! From where else ? Hence domains and software are "linked" by requirements.

11.4.2 From Goals to Requirements

Goals cannot serve as a basis for a reasonably manageable software development: They are simply "too lofty", expressed, as they are, in terms of human sentiments, non–computable socio—economic "values", etc. So we need to transliterate the goals into something that is computable. It is this we call requirements.

Requirements are a set of statements, expressed in terms of some understanding of what kind of, in our current example, *"a market abstract machine"* can be created inside the computing system.

The purpose of requirements is to express what the machine: The combination of hardware and software that is desired, shall offer.

We refer to the discussion (*"In the Domain All is Possible"*) presented above (Section 11.3.4).

[13] **Need:** (i) a lack of something requisite, desirable, or useful; (ii) a physiological or psychological requirement for the well-being of an organism; (iii) a condition requiring supply or relief, (iv) lack of the means of subsistence. Merriam–Webster On–Line: http://www.m-w.com/cgi-bin/dictionary

[14] **Goal:** the end toward which effort is directed. Merriam–Webster On–Line: http://www.m-w.com/cgi-bin/dictionary. [Aim(s), objective(s)]

The undesirable indeterminacies of "the market" may now be "remedied" through computing & communications support, or even automation. That is: The purpose of the requirements engineering phase of computing systems (cum software) development is to ensure proper behaviours of "market" stake–holders as well as of its supporting technology.

A set of requirements amount to a set of logical statements of the form: *"Provided the environment of (1) stake–holders, (2) support technology", (3) management and organisation, behave as laid down in the domain model, and provided that the (4) rules & regulations are consistent (...), then a correct implementation of these requirements will lead to a desirable computing system.* Failures in achieving good software oftentimes can be blamed not on the requirements themselves, nor on their maybe not so correct implementation, but can rather be blamed on the assumptions (1–4) not holding. It is therefore of utmost importance to secure that domain modelling have laid bare all such possible assumptions.

11.4.3 The Three Dimensions of Requirements

We distinguish between three kinds of requirements: Domain, interface and machine requirements.

Domain Requirements

Domain requirements are such which can be expressed sôlely by using technical terms from the domain. Below we shall discuss some principles for "deriving" domain requirements from domain models. Since different stake–holders may give rise to different domain requirements, these need be consolidated: They must be consistent and together form a relative complete set of requirements.

If two different providers of domain requirements from within the same stake–holder group give rise to mutually inconsistent requirements, then we have an *inconsistency* that must be resolved within the pertinent stake–holder group. If two providers of domain requirements from different stake–holder groups give rise to mutually inconsistent requirements, then we have a *conflict* that must be resolved at manage,ment level.

Domain requirements (aka. 'functional requirements') can therefore only be expressed (ie., written) using domain terms. These requirements can only be undersood, by a casual reader, if that person has first (read and) understood the domain (description). Thus it is that we say: Expressing domain requirements is like stating how actions and events, that is: Behaviours, can be effected by an abstract machine, and that abstract machine is a computable part of "the domain". Domain descriptions may be concerned also with functions that are not computable, but requirements necessarily have to deal with computation, hence with "that which can be computed".

Interface Requirements

Interface requirements are such which focus on the shared phenomena: Those for which we can designate an entity, or an event, or other in the domain which is also, somehow, represented within the machine. Interface requirements thus deal with the interface between the machine and its environment: The stake–holders and the supporting technologies.

Thus interface requirements focus on the so–called *user interface* (to stake–holders), as well as the *connections* between the machine and the supporting technologies.

Examples of $\mathcal{E}$–Transaction user interfaces are basically those of the Graphic User Interfaces (GUIs) and the user dialogues, Other examples are the mass input of for example catalogues, and rules and regulations (if enforced through computing).

We shall not deal further with interface requirements.

The "abstract machine" of interface requirements is thus composed of phenomena which belong both in the "abstract machine" of the domain and the "abstract machine" of the desired computing system — which, in a more narrow sence, now not using double quotation marks, we call the machine.

Machine Requirements

Machine requirements are such which can be expressed sôlely by using technical terms of the desired computing system itself: Abstract machine properties of the application software to be developed as well as of the platform (the hardware and supporting systems software) upon whose services the application software depends.

Examples of machine requirements are such which deal with time and space performance, dependabilities (such a the reliability, fault tolerance, security, safety, availability, accessability, etc., "ilities". No reference is made to any specific domain concepts when stating machine requirements, only at an abstract level: *The following operations must execute within such and such time–bounds: ...*, etc.

We shall not deal further with machine requirements.

11.4.4 Domain Requirements Development

When establishing domain requirements the requirements engineer is well–adviced in structuring the requirements modelling process and in presenting the requirements models along the lines given next: Projection, determination, instantiation, extension, fitting and initialisation.

We find that these "operators" that apply to domain requirements, and which, in sequence, results in domain requirements, are novel.

Projection

In projection we start with a domain description. usually with a rather encompassing one. From that we "cut down" to focus only on those aspects of the domain with which the remaining requirements are to be concerned. We go, so–to–speak, from the *scope* to the *span* of the problem to be solved — using concepts clearly enunciated by Michael Jackson [Jac95].

In our example of "the market", we might have described and modelled how real agents and real brokers behave. But we might settle on an $\mathcal{E}$–Transaction System that does not extend to agents and brokers. Thus we cut that part of the domain description out: It is no longer part of the requirements. It may still be part of the environment in which the desired computing systems is to perform, and hence the assumptions laid down in the domain model about agent or broker traders are still valid — and may have to be referred to in reasoning about the correctness of some $\mathcal{E}$–Transaction System functions.

We do not show 'formal projection' in the current paper.

Removing Undesirable Non–Determinacy

Usually "the domain is fickle": Human behaviour as well as that of supporting technologies. Desired computing support, or even automation, normally wishes to avoid a number of in–determinacies.

One purpose of requirements, amongst others, is to determine what is in–determinate in the domain: To render human behaviour predictable, to guard against sloppy, delinquent, even criminal actions, etc.

In our domain model of "the market" we left open very many possibilities of market transactions: (i) Inquiries were not guaranteed to be always followed–up by quotations, (ii) acknowledged orders were not guaranteed to be always followed–up by deliverries and invoices, (iii) invoices were not guaranteed to be always followed–up by payments, etcetera. With computing & communication much of this can be "followed–up" more strigently.

Thus requirements can be expressed that "removes" uncertainties. In–determinate behaviours can be made into determinate transaction protocols.

A requirements model can thus be derived from the projected model, one in which the internal non–deterministic choices have been replaced by determinate ones.

We do not show 'formal determination' in the current paper.

Extension

With computing & communication certain functions are possible — function that it was simply not feasible to do, in the domain, without for example computers. Brokering "across" any number of levels of wholesalers etc. is now feasible. Letting a great number of potential merchandise acquisitions

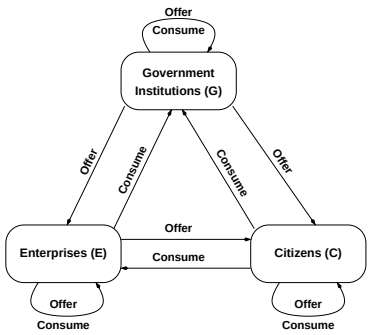
(purchases) be giverned by one or another form of auxtioning is now feasible — but was basically unthinkable of in support technology–weak domains. Making more visible complete supply chains so that all "players" are made more aware of what is going on, *"towards an open [market] society,"* is also now possible.

All are examples of extensions: Of the domain but only made believable through computing.

A special form of extension will be mentioned next.

$\mathcal{G}$overnment, $\mathcal{B}$usinesses, and $\mathcal{C}$itizens: Traditionally we have viewed $\mathcal{E}$–Transactions as something only involving consumers and retailers. Then "extensions" were made to the traditional $\mathcal{C}2\mathcal{B}$ paradigm. In general any *"direction of first initiative"* is worth considering. Simple redefinitions of inquieries, quotes, orders, deliveries, etc., can be readily put forward.

Figure 11.7: The $\mathcal{G}$overnment $\mathcal{B}$usiness $\mathcal{C}$itizen "Market Triangle"



- $\mathcal{G}2\mathcal{G}$: Speeding up internal, local and/or state government transactions, one branch og government buying services from another branch, etc.
- $\mathcal{G}2\mathcal{B}$: Government servicing business better: Government buying taxes from business in return for guaranteering stable economies, etc.
- $\mathcal{G}2\mathcal{C}$: Government buying taxes from citizens in return for offering social welfare, public education, etc.
- $\mathcal{B}2\mathcal{G}$: Businesses buying for example accreditation or certification from government paying with real monies.
- $\mathcal{B}2\mathcal{B}$: Yes, You gues what !
- $\mathcal{B}2\mathcal{C}$: Well–known !

- $\mathcal{C}2\mathcal{C}$: Enabling grass–root movements, and what not.
- $\mathcal{C}2\mathcal{B}$: Citizens selling their skills to businesses.
- And $\mathcal{C}2\mathcal{G}$: Citizens offering their votes to local and state politicians.

Discussion

In the current paper we are not covering such other domain requirements techniques as *instantiation*, *fitting*, and *initialisation*.

The discussion of the $\mathcal{G}$overnment $\mathcal{B}$usiness $\mathcal{C}$itizen "Market Triangle" is rather indicative. It hints at, but does not really substantiate specific requirements. We hope that the hints are sufficient to justify the claim, that only when we have reasonably concise domain models, such as the formal model of Section 11.3 — that only then — can we hope to conquer an emerging complecity of "Full scale $\mathcal{E}$–Transation Systems".

11.4.5 Discussion

We have covered some issues of requirements development based on domain models. May other requirements issues have not been covered, but are, of course, equally relevant: The processes, as they were for domain acquisition, for eleciting requirements; the issues of discivering inconsistencies, conflicts, and unwanted incompleteness.

Since $\mathcal{E}$–Transaction Systems typically involve many more, and in many cases, new kinds of, stake–holder groups than are usually encountered in requirements acquisition, special care need be taken — and also wrt. validation.

11.5 Conclusion

It is time to conclude. We do so, but only partially, and in three parts.

11.5.1 Summary: What has been Achieved ?

Two "achievements" stand out: One is methodological, the other is instantial.

Methdologically we have surveyed two major phases of computing, notably software systems development: Domain description and requirements prescription. We claim that our approach to domain modelling: Detailed, and both informal narration and formalised models, is novel. We claim, further , but do not show technical consequences of the claim, that the special techniques for domain facet modelling: *Intrinsics*, *support technologies*, *management & organisation*, *rules & regulations*, and *human behaviour* are new. Likewise we claim that the special techniques for domain requiremens modelling: *Projection*, *determination*, *extension*, etc., are also new. Although we have not shown

it explicitly, it is the possibility of strong, formal relations between domain descriptions and requirements prescriptions that is novel.

These (*emphasized*) techniques have then been exemplied on an **instance of a domain model:** That of "The Market". We have yet to see such clear models of "the market" being even related to in the expanding literature on $\mathcal{E}$–Business (etc.).

11.5.2 Insufficency of Current Modelling Techniques

We will only point out, in this section, that the notions of agents and brokers as first introduced was one of "the market". In further specifying those kinds of agents and brokers we first find that a need for formalised modal logic oriented [Che80, Pop94, CZ97, Kra99, BdRV01] specification languages arise, languages in which we can then describe, respectively prescribe how agents and brokers behave "in the real domain", respectively how we might require them to offer their services. We then extend that need to also include possibilities of providing implemented trader agents and brokers, ie. autonomous agents (as the term is used now in the AI literature) with *speech act* capabilities [Pet02].

11.5.3 Who Should be doing Domain Engineering ?

Is it really the idea that computing scientist cum software engineers cum knowledge engineers should be the ones who create domain models ? Well, for the time being, yes ! In collaboration — as was always assumed above — with domain stake–holders. But we foresee that gradually professionals of respective domains will have learned basic techniques of abstraction and modelling as part of their domain specific academic education, but in courses that basically propagate computing science concepts and techniques. And graudally they will take over the continued modelling of their domain. Just like physicists, after centuries of studying "Mother Nature", still study that domain, intensely, so we expect the man–made domains to be studied for as long as man sustain these domains. And: Just as most science–based disciplines: Biology, economy, mechanical engineering, etc., now "feature" a significant own (applied) mathematics study, so it is that we foresee that academics as well as professional practitioners within the kind of infrastructure components listed earlier (transport, health–care, etc.) will feature their own, domain–specific, ie., applied computing science studies and practices.

11.5.4 Acknowledgements

Discussions and serious work with colleagues, especially at UNU/IIST, the UN University's International Institute for Software Technology, Macau, SAR China, and especially Messrs Chris George and Søren Prehn, is much appreciated.

11.6 Bibliographical Notes

We apologize for the rather extensive set of references to own publications. We justify this by the perceived need to support claims made by references to supporting and substantiating literature.

References

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Number 53. Cambridge University Press, June 28 2001. 554 pages.

[BGP99] Dines Bjørner, C.W. George, and S. Prehn. *Scheduling and Rescheduling of Trains*, chapter 8, pages 157–184. *Industrial Strength Formal Methods in Practice*, Eds.: Michael G. Hinchey and Jonathan P. Bowen. FACIT, Springer–Verlag, London, England, 1999. [15].

[Bjø94] Dines Bjørner. Prospects for a Viable Software Industry — Enterprise Models, Design Calculi, and Reusable Modules. In *First ACM Japan Chapter Conference*, Singapore, March 7–9 1994. World Scientific Publ. Appendix in collaboration with Søren Prehn and Dong Yulin.

[Bjø95] Dines Bjørner. Software Support for Infrastructure Systems. Technical Report 47, UNU/IIST, P.O.Box 3058, Macau, November 1995. Position statement for the *First Malaysia Information Technology Days: 1–3 November 1995* .

[Bjø96] Dines Bjørner. New Software Development. Administrative/Technical Report 59, UNU/IIST, P.O.Box 3058, Macau, January 1996. Special *Theme* paper: *New Software Technology Development*. International Symposium: *New IT Applications for Governance and Public Administration*, UNDDSMS, Beijing, June 1996.

[Bjø98] Dines Bjørner. FISH: A Fisheries Infrastructure — Hardware/Software Concept. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 1998. his document provides a basis for an M.Sc. Thesis project carried out by Audur Thorun Rögnvaldsdottir, Sept. 1998 — Aug. 1999.

[Bjø99a] Dines Bjørner. A Triptych Software Development Paradigm: Domain, Requirements and Software. Towards a Model Development of A Decision Support System for Sustainable Development. In ErnstRüdiger Olderog, editor, *Festschrift to Hans Langmaack*. University of Kiel, Germany, October 1999. [16].

[Bjø99b] Dines Bjørner. Project Information, Monitoring and Control Systems — A Domain Analysis. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 1999.

[Bjø00a] Dines Bjørner. Domain Modelling: Resource Management Strategics, Tactics & Operations, Decision Support and Algorithmic Software. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millenial Perspectives in Computer Science*, Cornerstones of Computing (Ed.: Richard Bird and Tony Hoare), pages 23–40, Houndmills, Basingstoke, Hampshire, RG21 6XS, UK, 2000. Palgrave (St. Martin's Press). An Oxford University and Microsoft Symposium in Honour of Sir Anthony Hoare, September 13–14, 1999. [17].

[15] Postscript document at URL: http://www.it.dtu.dk/ db/racosy/scheduling.ps
[16] Postscript document at URL: http://www.imm.dtu.dk/ db/langmaack/hans.ps
[17] Postscript document at URL: http://www.imm.dtu.dk/ db/hoare/tony.ps

[Bjø00b] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk. [18].

[Bjø01a] Dines Bjørner. Informatics Models of Infrastructure Domains. In *Computer Science and Information Technologies*, pages 13–73, Yerevan, Armenia, September 17–20 2001. National Academy of Sciences of Armenia, Institute for Informatics and Automation Problems. [19].

[Bjø01b] Dines Bjørner. Towards the E–Market: To understand the E–Market we must first understand "The Market". In *Government E–Commerce Development*. Ningbo Science & Technology Commission, Ningbo, Zhejian Province, China, 23–24 April 2001. [20].

[Bjø02a] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? Venice, Italy, 7–11 October (Paper was completed August 19) 2002. The present paper is the long, 55 pages, version distributed at the October 7–11 Monterey Workshop in Venice. It will find its way into two published papers: [Bjø02b] and [Bjø02c]. [21].

[Bjø02b] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Radical Innovations for Systems and Software Engineering*, The Monterey Workshops, page (This report is expected to be of size 20 pages.), Venice, Italy, October 7–11 2002. The present, to be published paper, is a short version of [Bjø02a], and is complemented by [Bjø02c] in making up for [Bjø02a]. [22].

[Bjø02c] Dines Bjørner. Towards Design Calculi for Requirements Engineering and Software Design. In *Essays and Papers in Memory of Ole–Johan Dahl*, page (This paper is of size 21 pages.), August 2002. The present, to be published paper, is a short version of one part of [Bjø02a], and is complemented by [Bjø02b] in making up for [Bjø02a].[23].

[Bjø02d] Dines Bjørner. What is an Infrastructure ? In *The UNU/IIST 10th Anniversary Symposium*. UNU/IIST, Springer, March 2002. Eds.: Armando Haeberer, Tom Maibaum and Carlo Ghezzi. [24].

[Bjø03a] Dines Bjørner. Domain Engineering — A Prerequisite for Requirements Engineering — Principles and Techniques. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[Bjø03b] Dines Bjørner. E–Business. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[Bjø03c] Dines Bjørner. Financial Service Institutions: Banks, Securities Trading, Insurance, *&c*. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[Bjø03d] Dines Bjørner. Health–care Systems. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[18] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/2ifacpaper.ps
[19] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/csiam.ps
[20] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/ningbo.ps
[21] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/venezia.ps
[22] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/venezia.ps
[23] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/olejohandahl.ps
[24] Postscript document at URL: http://www.imm.dtu.dk/ db/documents/lisboa.ps

[Bjø03e] Dines Bjørner. Logistics. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[Bjø03f] Dines Bjørner. Models, Semiotics, Documents and Descriptions — Towards Software Engineering Literacy. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. [25].

[Bjø03g] Dines Bjørner. Principles and Techniques of Abstract Modelling — Some Basic Classifications. — Towards a Methodology of Software Engineering. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. [26].

[Bjø03h] Dines Bjørner. Projects & Production: Planning, Plans & Execution. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003

[Bjø03i] Dines Bjørner. Railways Systems: Towards a Domain Theory. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.

[Bjø03j] Dines Bjørner. Requirements Engineering — Some Principles and Techniques — Bridging Domain Engineering and Software Design. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2003. [27].

[BLP94] Dines Bjørner, Dong Yu Lin, and S. Prehn. Domain Analyses: A Case Study of Station Management. In KICS'94: *Kunming International CASE Symposium, Yunnan Province, P.R.of China.* Software Engineering Association of Japan, 16–20 November 1994.

[Boe81] B.W. Boehm. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, NJ., USA, 1981.

[BPG99a] Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Domains. Technical report, Dept. of IT, Technical University of Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM. [28].

[BPG99b] Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Requirements. Technical report, Dept. of IT, Technical University of Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM. [29].

[BRH98] Dines Bjørner, Vasco Rosario, and M. Helder. A Normative Model of Concrete Banking Operations — Banking Rules & Regulations and Staff/Client Behaviours. Research, Informatics and Mathematical Modelling, Technical University of Denmark, June 1998. (Need be revised: Some typos etc. !).

[CH03] Zhou Chaochen and Michael R. Hansen. *Duration Calculus: A formal approach to real–time systems.* Monographs in Theoretical Computer Science. Springer–Verlag, 2002 (2003). A 238 page manuscript was sent to the potential publisher Monday 15 July 2002. This book collects the work of the main originator and one of the main contributors to the theory of duration calculi. As such the book represents a dozen years of research.

[25] DRAFT Postscript document at URL: http://www.imm.dtu.dk/ db/documents/series/modode.ps
[26] DRAFT Postscript document at URL: http://www.imm.dtu.dk/ db/documents/series/absmod.ps
[27] DRAFT Postscript document at URL: http://www.imm.dtu.dk/ db/documents/series/require.ps
[28] Postscript document at URL: http://www.imm.dtu.dk/ db/racosy/domain.ps
[29] Postscript document at URL: http://www.imm.dtu.dk/ db/racosy/requirements.ps

[Che80] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, December 1980. 312 pages.

[CZ97] Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*. Number 35 in Oxford Logic Guides. Oxford University Press, 1997.

[GHH+92] Chris George, Peter Haff, Klaus Havelund, Anne 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.

[GHH+95] Chris George, Anne Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[Jac95] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995. ISBN 0-201-87712-0; xiv + 228 pages.

[Kra99] Marcus Kracht. *Tools and Techniques in Modal Logic*. Studies in Logic and The Foundations of Mathematics. North–Holland, June 1 1999. 572 pages, Amazon price:US $ 127.00.

[Pet02] Hans Madsen Petersen. Agents and Speech Acts: A Semantic Analysis. Master's thesis, Informatics and Mathematical Modelling, Computer Science and Engineering, Bldg. 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs. Lyngby, Denmark, 20 June 2002.

[Pop94] Sally Popkorn. *First Steps in Modal Logic*. Cambridge University Press, December 1994.

Contents