TM（T字形ER）によるモデリング

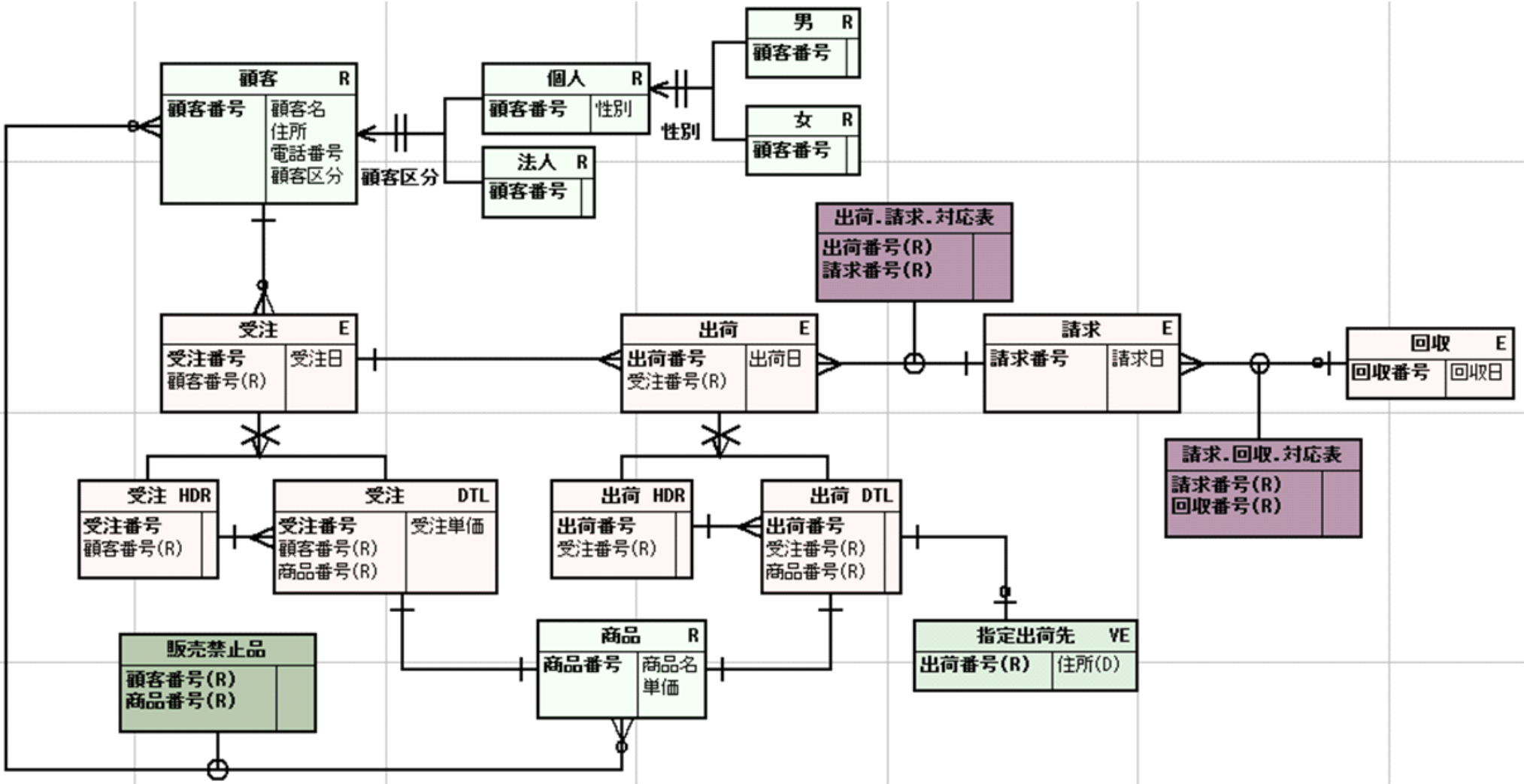
ソフトウェアシンポジウム2009　WG：モデリングの方法論とツール

時本　永吉
株式会社日本システムディベロップメント
2009/6/18

Agenda

1. 入門編：TM（T字形ER）とは
2. 実演１：まずは書いてみる
3. 理論編：無矛盾性と完全性を正確に記述する
4. 応用編：ビジネスを検証して、ソリューションを提案する
5. 実演２：検証しながら書く
6. 発展編：展望、今後の課題
7. 文献

1．入門編：TM（T字形ER）とは



1．1．正規形とER図とTM

- 考案者：佐藤正美氏
  - RDBと設計手法のER図を日本に持ち込んだ
  - ER図がコッド(Codd, E.F.)の正規形を表せていないことを知っていた
    - 構造的意味論を表せていない
    - すべてを「並べられるもの」としている
    - 他に表現方法がなかった
  - 正規形は文法として優れていたが、ビジネスに適用しようとすると、うまくいかない
    - 正規形は「並びがあるもの」しか対象としていない
    - 「並びがないもの」は定義できずに対象外としたが、実態として存在する
  - TMは「並びがあるもの」（全順序）と「並びがないもの」（半順序）を明確にし、実態を扱う
    - 数学、哲学（言語哲学→科学哲学）を徹底的に学び、正規形を見直した

1．2．TMはビジネスを表現するモデル

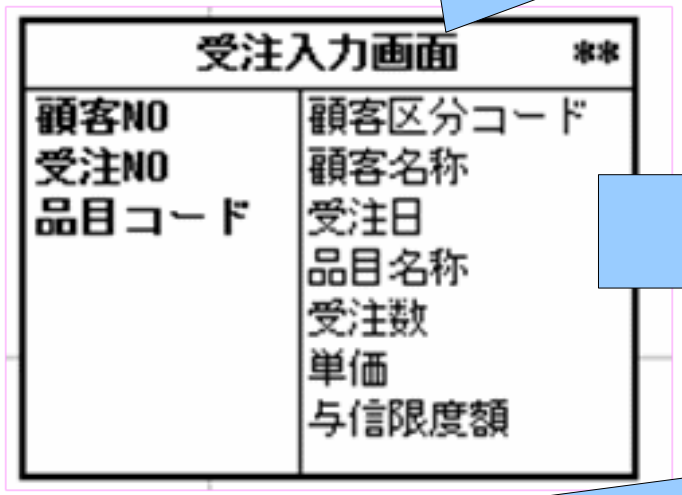
- TMはデータモデルではなく、モデルである
  - ダイアグラム（図）ではない
    - 図は表現方法の一つでしかない
  - データモデルではビジネスの実態を書き表せないため、正規形は作れない
    - ビジネスの実態がわからない段階で、妥当な構造はわからない
    - ビジネスと差異があると、新規ビジネスを考えたときにシステムの作り直しが発生する
  - TMにとって、モデルとは次のものである
    - 現実的事態（事業過程、管理過程）について、無矛盾で完全に、正確に書き表す
    - 構造や環境適応能力を見直す
  - 論理的に完全に書けないのであれば、モデルではない
  - 論理的に証明できないのであれば、プログラマはいらない
    - 真理値表と補集合、対偶の考え方（一般的なテスト手法）だけでも知っていればいい

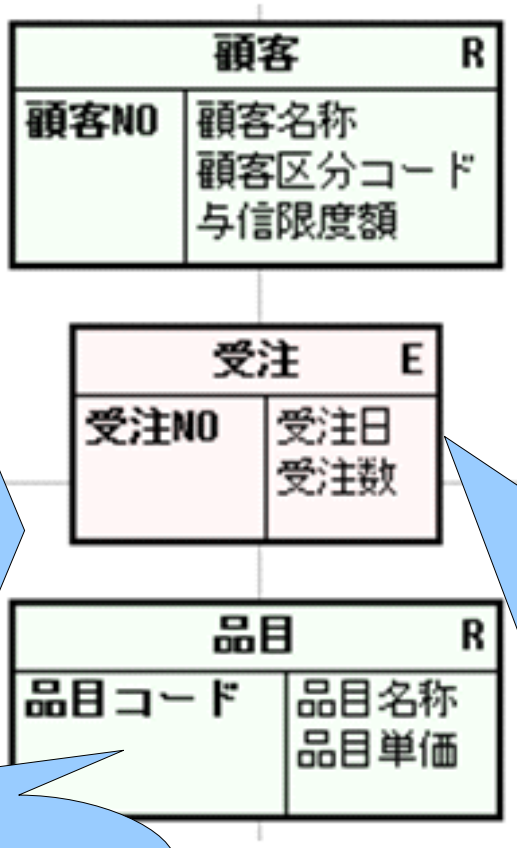
1．2．TMはビジネスを表現するモデル

- 一般的な位置づけとしては、分析モデル
  - 現実的事態を書き表す
  - 検証のために、網羅性、データの構成、制約束縛を整理するため、設計、製造時に書く内容も大体まとまっている
    - 構成、制約束縛を整理すると、クラスになる。あとはその中のどれを実際にクラスとして作るかを検討するくらい
    - チェックはモデル上に定義された関係で行えるので、実装するのは、関係を確認するSELECT文程度
- 1つのモデルで考える
  - 分析、設計、製造、さらに細かいレベルで異なる担当者、モデル（ドキュメント、プログラム、など）が登場する
    - 異なるドキュメント間の整合性を保証しなければならない
    - ビジネスを見直したいときに、すべてのドキュメントに手を加えないといけないのか？

2. 実践1:まずは書いてみる

1－1. 要件(伝票、画面)から抜き出す。左にNO、コードを、それ以外を右に書く(区分や種別も右)。
名前はできるだけそのまま使う。
登場しないものは書かない。

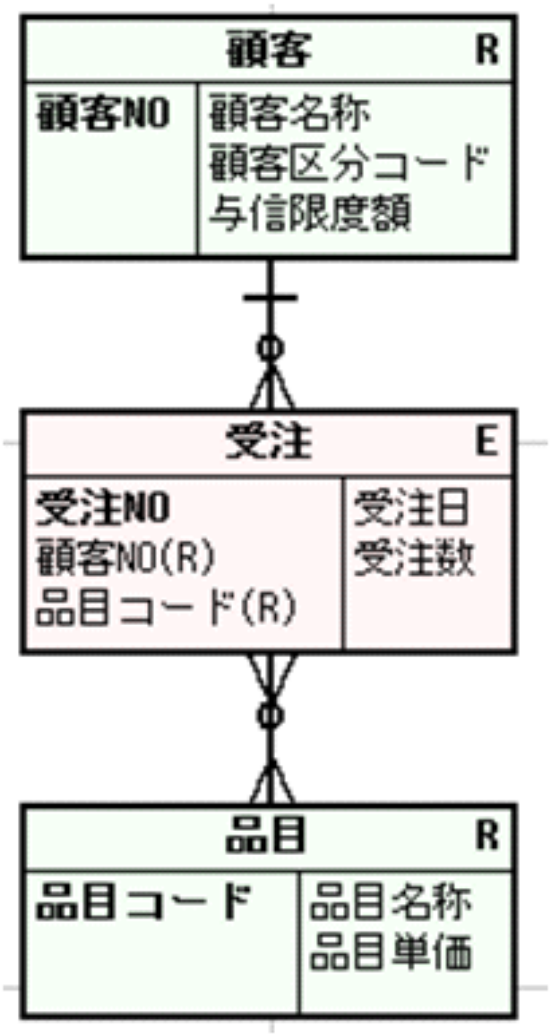




2. 箱の右に「日付」があるものには箱の右上に「E」(Event)と書く。
それ以外は「R」(Resource)と書く。
但し、履歴用の日付や適用日などについては「R」とする。また、箱の名前が動詞にならないものも「R」とする(社員の入社日、生年月日、品目の発売日など)。

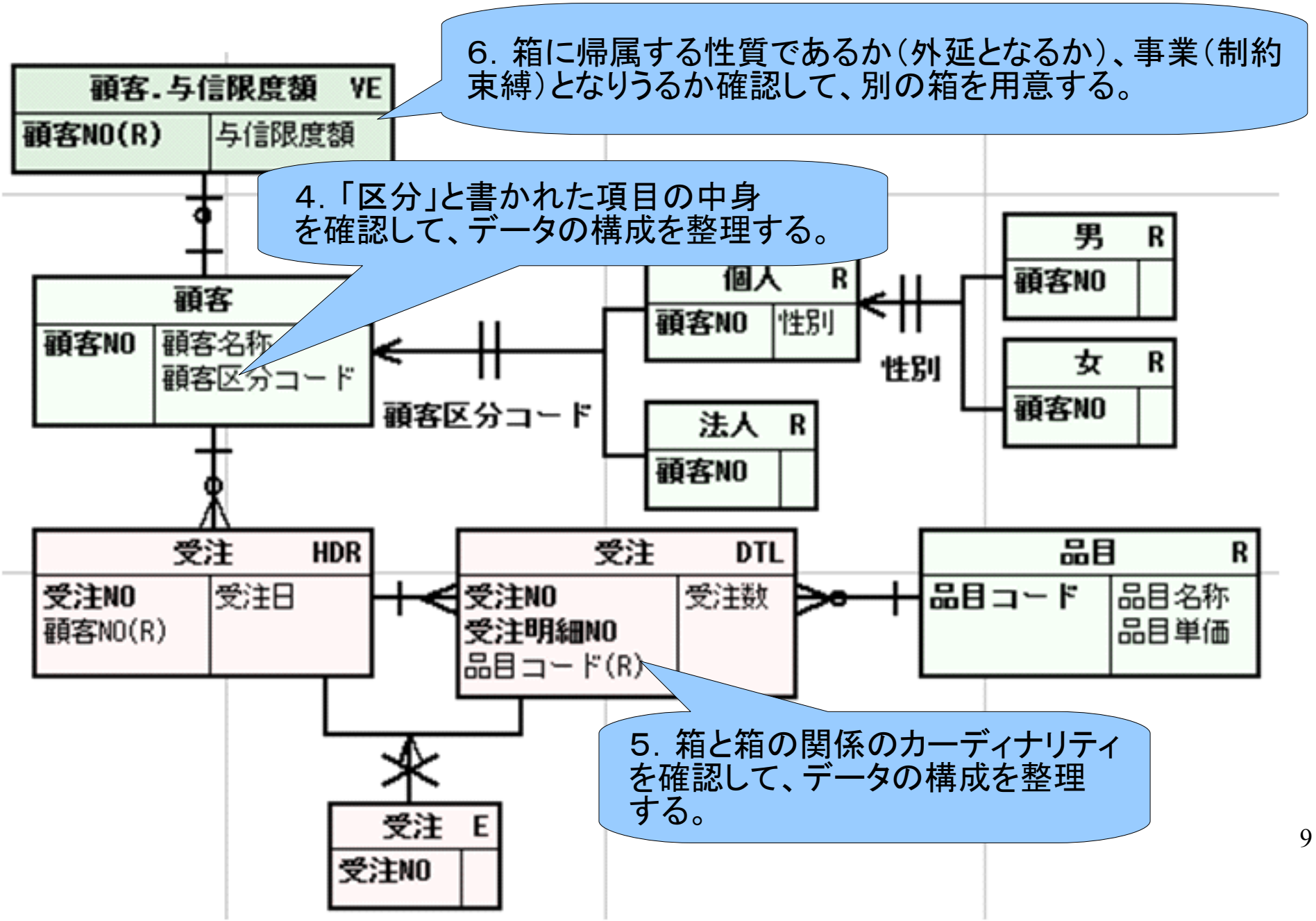
1－2. 左の項目ごとに箱を分ける。
どの箱に入れればいいかわからないものは確認する。
「単価」は受注時の単価ではなく品目の単価なので、「品目単価」として「品目」に入れた。「与信限度額」は「顧客」の情報のため、「顧客」に入れた。

2. 実践1:まずは書いてみる



3. 箱と箱の組み合わせを確認して、関係のあるものを線でつなぐ(カーディナリティにも注意する)。
R-Eに線を引いた場合は、Rの左の項目をEの左に記述する。このとき、項目名の後ろに「(R)」を書く。
(R)の意味と、R-R、E-Eの関係は後述する。
この時点で、受注と品目の関係がおかしいが、このあとで整理する。

2. 実践1:まずは書いてみる



３．理論編：無矛盾性と完全性を正確に記述する

- 実践１でやったこと
  - 認知されたNO、コードごとに箱（entity）を作り、性質（attribute）を振り分ける
    - 性質が何に帰属するか明らかになる
    - 勝手なNOやコードを書かない
  - 箱（entity）は事態（event)と主題（resource）で分ける
  - 箱を線（関係）でつなぐ
  - データの区分構成を確認する
    - 区分が排他的ORであり、包摂（包含）関係が妥当であるか
    - 区分は集合を切断するためのものか
  - データの多値構成を確認する
    - OR（排他的OR）か、ANDか区別する
  - クラス
    - 箱に帰属する性質であるか確認する
    - 箱と箱の間にある制約束縛が他の箱と関係を為さないか確認する

３．１．TMでなければダメなのか？

- TMでなくてもよい
  - うまくいっているところは、だいたい同じようなチェックがされている
  - ツール、方法論独自の方法で無矛盾性と完全性が保証されている
  - TMは正規形を現場に正しく適用しようとしているだけ
    - ER図だけでなく、RDBすらも正規形を表せていない
    - セット理論だけでなく、クラス理論も取り入れて、正規形を補足する

３．２．モデリングのポイント

- 現実的事態（事業過程、管理過程）を記述する
  - 正当化条件、構成条件を網羅する
    - ユーザー言語（現実的事態）を変形せず、出来るだけ機械的に書く
  - 正確に記述して、証明可能（検証可能）であること
    - 無矛盾性（論理法則）
    - 完全性（意味として正しい）
- 形式的構造＝個体＋関係
  - ２つのアプローチ
    - 実態主義：　個体ありき、関係は二次的（ユーザーの言葉）
    - 関係主義：　関係ありき、個体は付値（関数）（データモデル、科学分析）
  - TMは関係主義
    - 個体＝無定義語（パラメータ）
    - 個体の解釈は現実的事態（関係）で異なり、現実的事態（関係）だけが個体の意義を定める

３．３．モデルの構成規則

- 生成規則（構造論）
  - 文法、論理法則によって表された形式的構成
  - 健全性、無矛盾性を保証する
  - 無矛盾性だけを証明しても、正しいこと（完全性）が証明されたとはいえない
  - 現実的事態（事業思想、管理思想）に対する解釈（ユーザーの自然言語そのまま）のこと
- 指示規則（意味論）
  - 解釈、現実的事態によって表された形式的構造
  - 完全性を保証する
  - 解釈によって表された形式的構造であり、現実的事態のこと
    - 関係の網羅性の確認、制約束縛（アトリビュートリスト）

３．４．モデリングの進め方

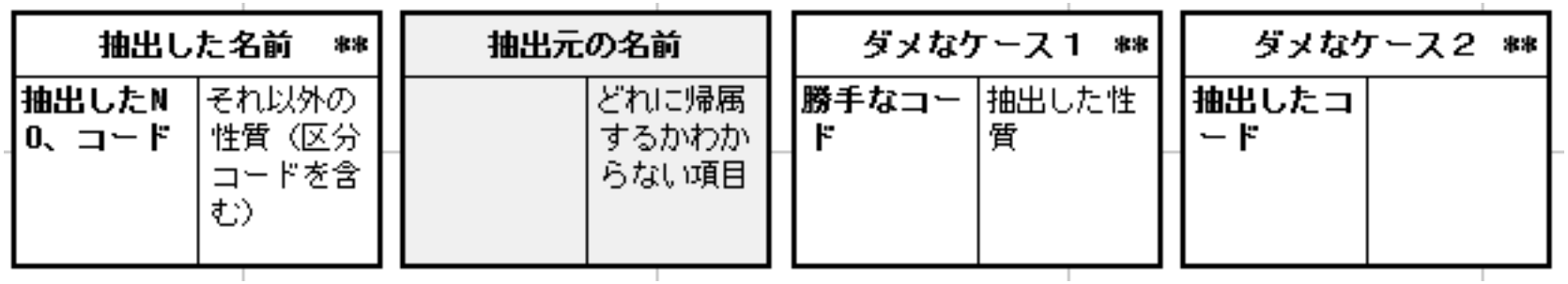
観点		フェーズ	方法・手順
生成規則（構文論）	文法	事業分析	１．個体の認知
			２．関係の性質（データの並び）
			３．関係の文法
	データの検証	設計	４．データの周延
			５．データの多値
指示規則（意味論）	意味論	製造	６．クラス

- 手順１～５は一般手続きがある
  - 誰が書いても同じになる
  - 誰が書いても同じようになるように、勝手な解釈をいれない
- 手順６は一般手続きがない
  - 何をクラスとしてまとめるかは恣意的

３．４．１．個体の認知

- ヒアリングした内容や、現行システムの画面、帳票などを元に、認知されたNO、コードごとに箱（entity）を作り、性質（attribute）を振り分ける
  - ここで書かれたNOやコードがユーザ言語であり、合意されたもの(認知単位)
  - 箱を書いたときに対応するNOやコードがないからと言って、勝手な項目を書いてはいけない（合意されていない）
  - 名前（修飾子）をもとに、性質を考え、わからないものは確認して、振り分ける
- ユニーク（一意性）にするためのキーを作るのではない
  - NO、コードは箱そのものを表すので、１つだけで認識できる
  - 複合キーは関係によってのみ現れる
    - 箱を複合キーで認識すると、論理が隠れてしまう
  - 性質が箱に帰属する単位がわかることがポイント
  - 第１正規形では「スカラのみで成り立つもの」としている
  - 一意性を保証するために作ってしまうと、変更が困難になったり、RDBの制約（インデックス、探査ルールなど）を受けて、逆に遅くなってしまう

３．４．１．個体の認知



- 箱：　抽出した名前
  - 左に1つのNO、コード、右にそれ以外を置く
  - 箱の左の項目を個体指定子（認知番号）と呼ぶ
  - 箱の右の項目を性質（アトリビュート）と呼ぶ
- 箱：　抽出元の名前
  - 対応する箱（NO、コード）がわからないものは、抽出元単位でまとめる
- 箱：　ダメなケース１
  - 対応する箱（NO、コード）がわからないからといって、勝手なコードを書いてはいけない
- 箱：　ダメなケース２
  - 性質がないコードは認知番号ではなく性質である

３．４．１．個体の認知

- 問題のあるケース
  - 「青色」や「12cm」は定義できない
    - 事業によっては分類や型紙として合意している場合もある
    - 合意されていない場合は、他の箱の性質でしかない
  - 合成キーは帰属先がわからない
    - JANコードは世間で認知されているからよいだろう
    - 「大分類（３桁）＋中分類＋（３桁）＋小分類（３桁）」からなる商品コードは作るべきではない
      - 現実の事態を表せているかもしれないが、業態を変えることが困難になってしまう
      - こういった項目は項目名の右側に「*」をつけてチェックしておく
  - 複合キー（社員番号＋組織番号）は関係を隠してしまう
  - 性別は箱にならない
    - 性質がないものは箱にならない（性別ごとの染色体の数を管理するような事業ならば箱になるかもしれない）
    - 性質がないものは区分コード（性質）かもしれない

３．４．２．関係の性質（データの並び）

- 関係には、対称性（半順序）と非対称性（全順序）があり、区別しなければならない
  - 非対称性：　並び替えても意味が通る
    - 愛川欽也と佳村萌には親子関係がある
  - 対称性：　並び替えると意味が通らない
    - 愛川欽也は佳村萌の親である
- コッドの正規形は全順序ののみを対象としており、半順序を対象外としている
  - ビジネスの実態は半順序だらけ
  - 全順序と半順序の関係は別物である
  - 半順序を定義できなかった
    - 全順序は定義できた
    - 半順序は全順序の補集合だといえる

３．４．３．関係の性質（データの並び）

- 箱／個体（entity）を事態（event)と主題（resource）で分ける
  - 箱の右上に「E」、または「R」をつける
- 個体（entity）
  - 事態（case, event：以下、EまたはeventとＥ記す）
    - 行為や出来事
    - 非対称性（全順序）
    - 「並びがある」とは「並べる基準」があること
      - 日付が並べる基準、すなわち、日付の有無が全順序の定義となる
        - 前後関係を表す日付のみが全順序の条件となる
        - 誕生日や更新日、適用日などはただの性質の一つ
  - 主題（subject, resource：以下、RまたはresourceとＲ記す）
    - 行為者
    - 対称性（半順序）
    - 事態の補集合が主題となりうる

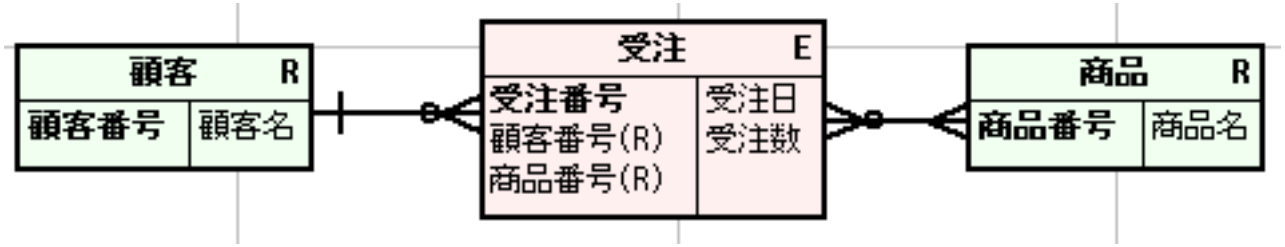
３．４．３．関係の文法

集合	種類	文法	
		１ 対 １	複数 対 複数
2つの集合の間	R-E	R（主体）がE（事態）に関与する	クラス理論（多値）
	E-E	先行・後続	対応表（マッピングリスト）
	R-R	対照表（データディクショナリ／制約束縛）	
1つの集合の中	再帰	メンバーを並べる（先行・後続関係）	

- 全ての箱と箱の間に関係があるか確認する（網羅性）
  - 関係がある箱の間に線を引く
    - 線(関係）が重要
    - 線が箱の意義を定め、線の数だけ意義がある
  - 関係によって生まれたentityには個体指定子は存在しない
  - 関与するresource、eventの個体指定子があるだけ
    - 区別して、個体指定子の後に(R)を付ける
    - RとはRe-Useのことであり、Referenceではない

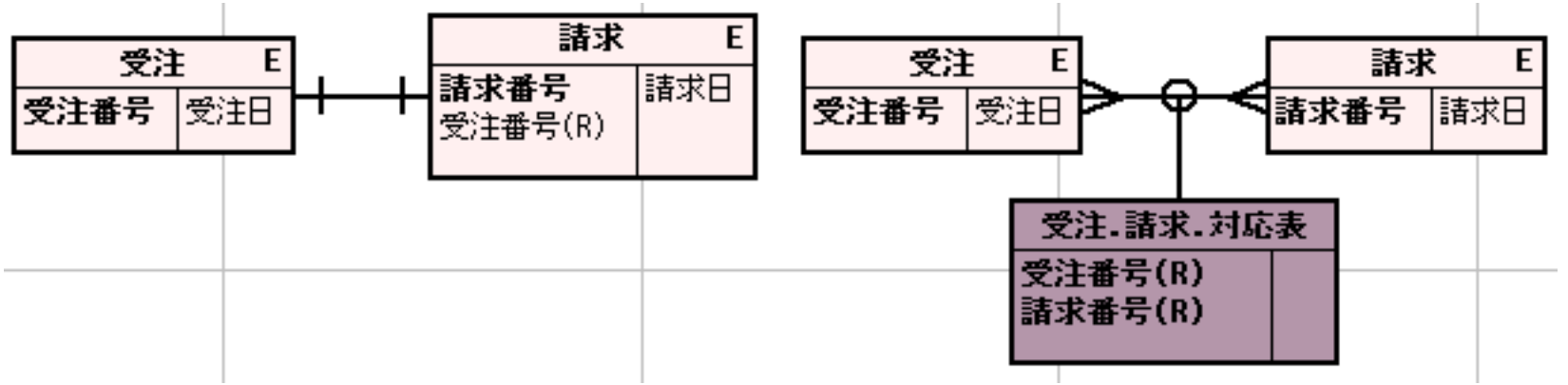
３．４．３．１．関係の文法：R-E

- resourceがeventに関与する
  - 一般的に、ビジネスを表すときは、この関係を示す
    - 読み方「顧客Aから商品Bについて、C日にD個、受注した」
    - resourceの個体指定子をeventの箱の左側に侵入させる
    - 侵入した項目に「R」（Re-Use)を付ける
    - 受注の個体指定子は「受注番号」のみ
  - 複数対複数の場合は、セット理論では表現できない
    - １回の受注で複数種類の商品を扱うことがほとんど
    - クラス理論、すなわちANDの多値関係（HDR/DTL）で表現する



３．４．３．２．関係の文法：E-E（後続関係／対応関係）

- eventの後続関係
  - ある受注に対して出荷する
  - 後続の箱の左側に、先行の箱の個体指定子を侵入させる
    - 侵入した項目に「R」（Re-Use)を付ける
    - 請求の個体指定子は「請求番号」のみ
- eventの対応関係（対応表）
  - 複数の受注に対して月でまとめて請求する
  - 箱（マッピングリスト）を書き、先行、後続の個体指定子の対応関係を書く
    - １対１以外の時はすべて対応表を書く

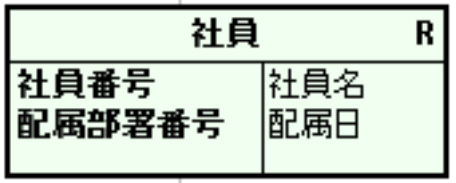


３．４．３．３．関係の文法：R-R（対照表）

- 対照表は2つの意味を持つ
  - resource間に制約束縛が存在する（resourceとしての対照表：構文論）
    - ある取引先にこの商品を売ってはいけない
  - resource間に事態を表す関係が存在する（eventとしての対照表：意味論）
    - ある社員をこの部署にいつ所属させた
- 正規形は写像理論（射影）を前提にしているため、全域関数（直積）を想定しているが、実際の事業は部分関数である
  - 4値理論：True, False, Undefined（未定義）, Unknown（未知）
  - ER図、RDBMSは3値になってしまっている
  - nullは値ではなく状態（充足されない状態）なので、関係の有無で表すべき

３．４．３．３．関係の文法：R-R（対照表）

- 対照表はデータディクショナリ
  - 事業は頻繁に変わるものだが、個体が変わるのは稀
  - 変わるのは個体間の関係（制約束縛）であり、対照表はこれを表したもの
    - 社員の配属先を変えるとき、変わるのは社員ではなく、社員と部署の配属関係である
  - 関係を表さなかった場合の悪弊
    - 所属が変わるとき、データを消すか、配属先をキーにするか、履歴番号が必要になる
    - 配属先が変わるだけで社員は何も変わらないのに、「社員」を更新しなければならない（配属部署番号は社員に帰属しない）
    - 専務のように配属部署がない社員を扱うようになったとき、Null状態（充足しない状態）が生まれ、社員を定義できなくなる（社員の定義は社員番号と配属部署番号としている。新入社員で配属先が決まっていない社員なのか、配属先を持たない社員なのか、判断できない）。

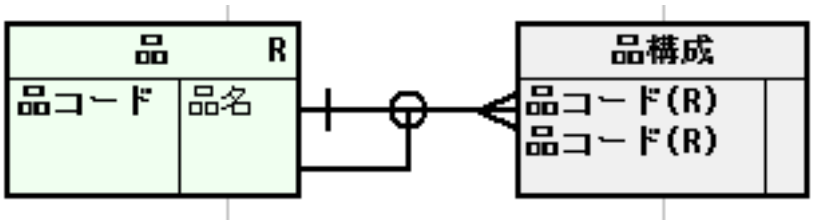


３．４．３．３．関係の文法：R-R（対照表）

- resourceの対照表か、eventの対照表か、どちらを表すかは現実的事態を見なければわからないが、大抵、性質に日付があるかないかで判断できる（箱の性質を考えるときと同じ）
  - 「販売禁止リスト」にも日付がありそうだから、eventではないか？
    - そうかもしれないが、現在の事業ではそのように管理していないことがわかる（現実的事態だけが正しいことを決める）
    - 日付を持つべきかどうかはビジネスにより、提案すべき内容だろう
  - 「配属日」はただの履歴または適用日ではないか？実践１ではこういった日付はresourceにしておくと言っていた
    - 実践１ではなかったが、こういった日付は手順６のクラスを考えるときに整理される
    - 基本的に対照表は関係を表し、そこには関係が始まった時点が存在するため、たいていはeventとなる

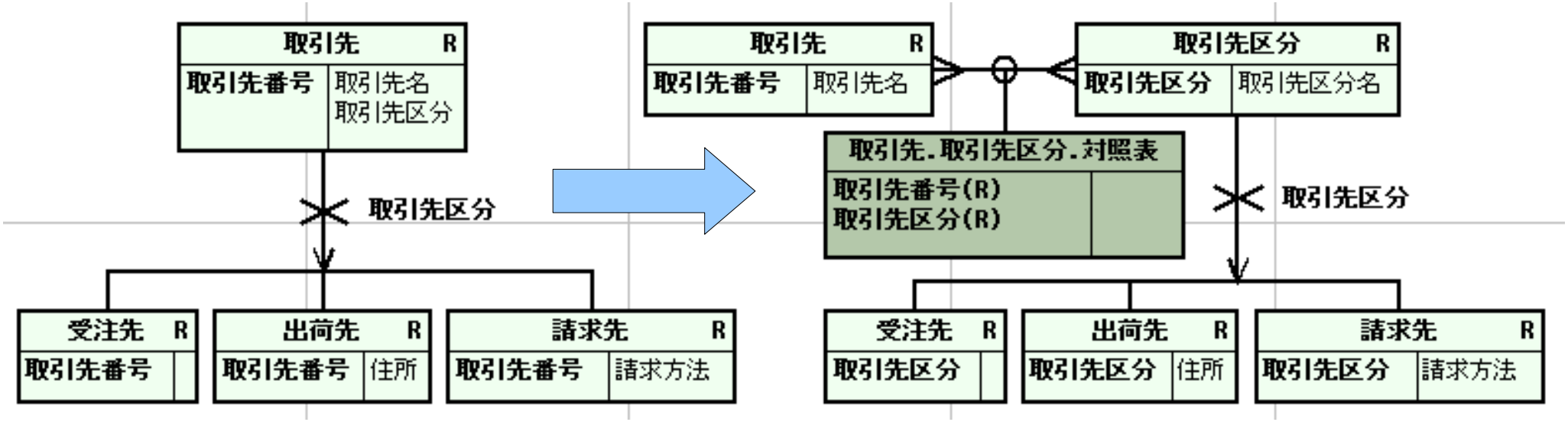
３．４．３．４．関係の文法：再帰

- １つの集合(箱）の中のメンバーを並べる
  - 品と品の間には構成関係が存在する（部品表）
  - 受注と受注の間には取り消し関係が存在する
- 再帰表に参照元の箱の個体指定子を侵入させる
- 「並べる」ということは、前後関係がある、ということ
  - 箱が並びを表現しており、侵入させた個体指定子を変化させない（親部品―子部品）
  - RDBでは並びをサポートしておらず、同じ名前は使えないため、実装の時に限り、名前を変える



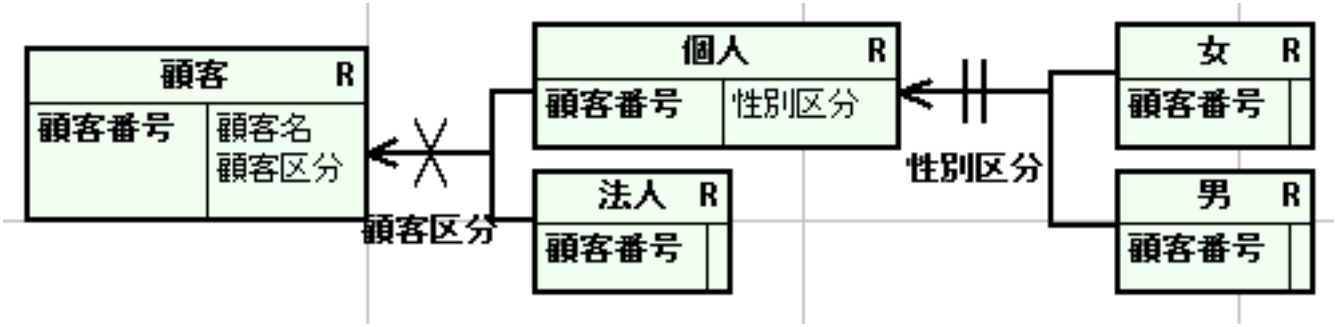
３．４．４．データの周延

- ある箱に「区分」があったとき、その集合中身（構成）を確認する
  - 区分が切断目的で使われているか
    - 区分は集合を切断するものであり、（排他的）ORでなければならない（交わらない）
      - 性別が男であり女である人間は存在しない
      - ある取引先が受注先であり請求先である場合は、取引先と取引先種別の対照表を作る（ORの多値関係）



３．４．４．データの周延

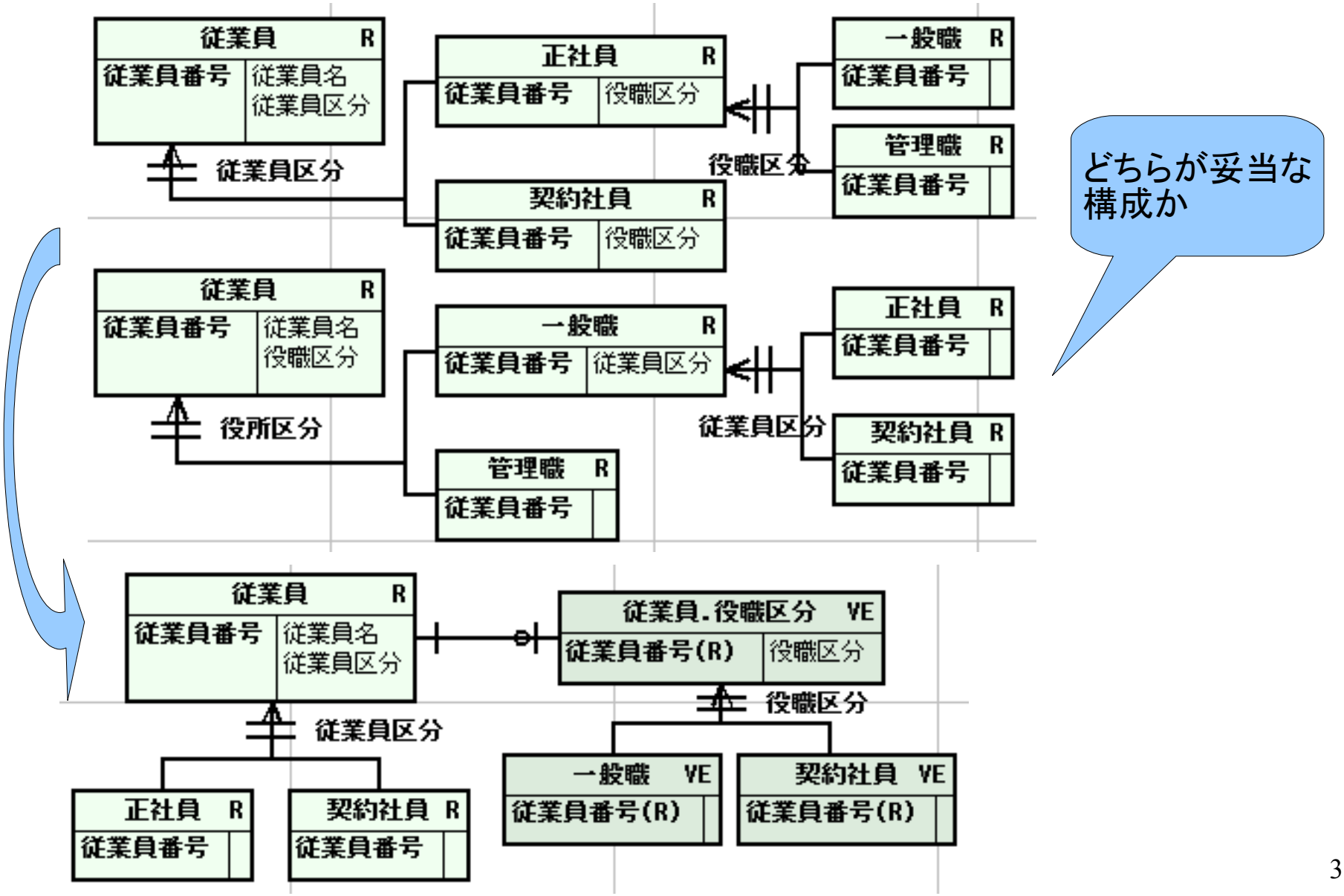
- 区分は1つの観点においてのみ切断されているか
  - 顧客区分が{男、女}や{個人、法人}はありえるが、{男、女、法人}は観点がまったく異なる
  - 補集合で考える（男の補集合は{女、法人}は妥当か）



３．４．４．データの周延

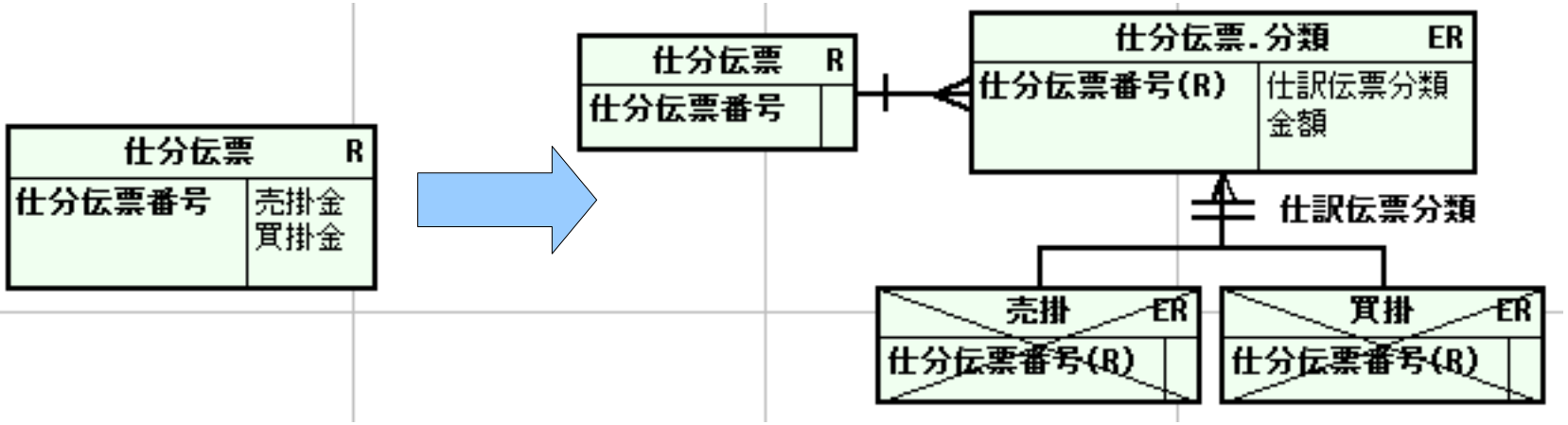
- １つの箱に複数の区分が存在しないか
  - 基本的には1つの箱にもてる区分は1つのみ
  - 複数ある場合は、サブセットを書き、サブセット側に区分を持たせる（包摂関係で表現する）
  - 包摂関係のない区分の場合、その箱を切断するためのものか確認する（区分の用途、事態が異なるのではないか）
- 包摂（包含）関係が妥当か
  - 区分を入れ替えて、意味が通るのであれば、包摂関係になり
  - 観点の違う区分であるならば、別の箱を用意する

３．４．４．データの周延


従業員 R
従業員番号
従業員名
従業員区分
従業員区分
正社員 R
従業員番号
役職区分
契約社員 R
従業員番号
役職区分
一般職 R
従業員番号
管理職 R
従業員番号
役職区分
どちらが妥当な構成か
従業員 R
従業員番号
従業員名
役職区分
役所区分
一般職 R
従業員番号
従業員区分
管理職 R
従業員番号
正社員 R
従業員番号
契約社員 R
従業員番号
従業員区分
従業員 R
従業員番号
従業員名
従業員区分
従業員区分
従業員.役職区分 VE
従業員番号(R)
役職区分
役職区分
正社員 R
従業員番号
契約社員 R
従業員番号
一般職 VE
従業員番号(R)
契約社員 VE
従業員番号(R)

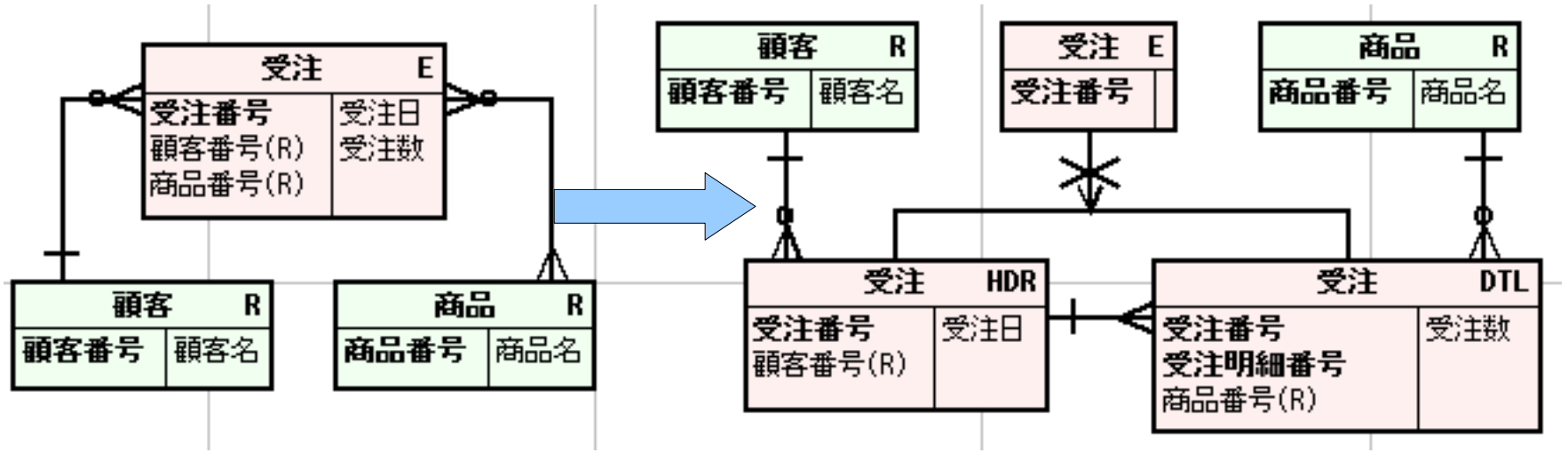
３．４．５．１．データの多値： ORの多値関係（MO/MOR）

- データの周延チェックで交わりのある（排他的にならない）OR関係
  - 対照表で表現する
- 区分もなく、並びで表現したい
  - 仕分伝票（仕分伝票番号、買掛金、売掛金）
  - MO（many value or、またはentity role）で表現する
    - 認知された区分が存在せず、RDBでサポートされていない並びを表現するために都合上、分類するコードを用意する
    - 合意されたもの、個体指定子においてはいけない
    - プログラマの都合で合意を強要してはいけない



３．４．５．２．データの多値： ANDの多値関係（MA/MAND）

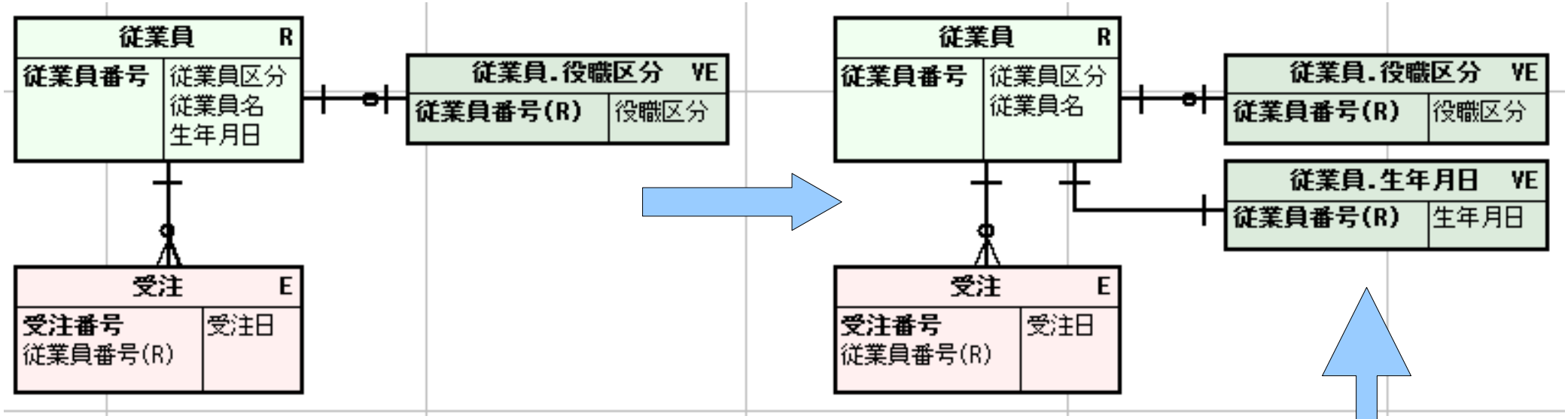
- 関係文法：R-Eの複数対複数を表現する
- 第１正規系（セット理論）はMOを前提としており、ANDを考慮していない
  - 数学の合成関数（y=f(g(x))）、または第４世紀形（クラス理論）で解決する
    - セット理論は集合と集合の関係を扱う
    - クラス理論は関係の構成を集合（個体）とみなす
      - 受注した商品の集合（受注）に対して、顧客の集合で対応関係をとる
  - 一般には伝票／伝票明細（HDR/DTL）の関係
      - 伝票／伝票明細で伝票とみなす（ばらばらに存在していない）

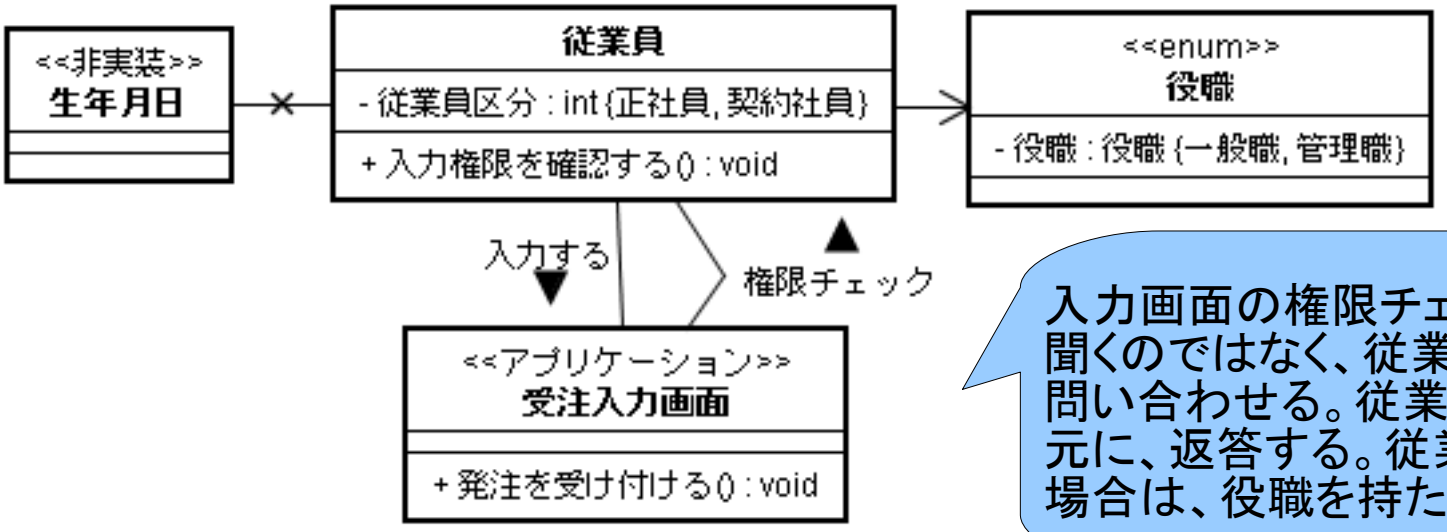


３．４．６．１．クラス： VE（Virtual entity）

- entityではない＝個体指定子を持たない
  - entityに帰属しない性質が定義される（制約束縛）
    - 他のentityやVEと関係をもてない（線をひけない）
    - ただし、制約束縛によるクラス抽出としては線が引かれることがある
- オブジェクト指向でいうと、カプセル化されたクラスみたいなもの
  - 他の箱から直接参照されず（線が引かれず）、ある箱を経由する
  - ある箱との関係
    - Rの中に侵入しているEの性質を抜き出す
    - Eの中に侵入しているRの性質を抜き出す
    - R, Eの中に侵入している充足しない状態(null)を抜き出す
    - R, Eの中に侵入している帰属しない性質を抜き出す
      - 資本金は顧客の性質となるか？
        - 会社の定義によって異なるため、文脈から判断するしかない

3．4．6．1．クラス： VE（Virtual entity）





入力画面の権限チェック時に役職を聞くのではなく、従業員に権限があるか問い合わせる。従業員は自分の役職を元に、返答する。従業員が契約社員の場合は、役職を持たない。

３．４．６．２．クラス：　概念的スーパーセット

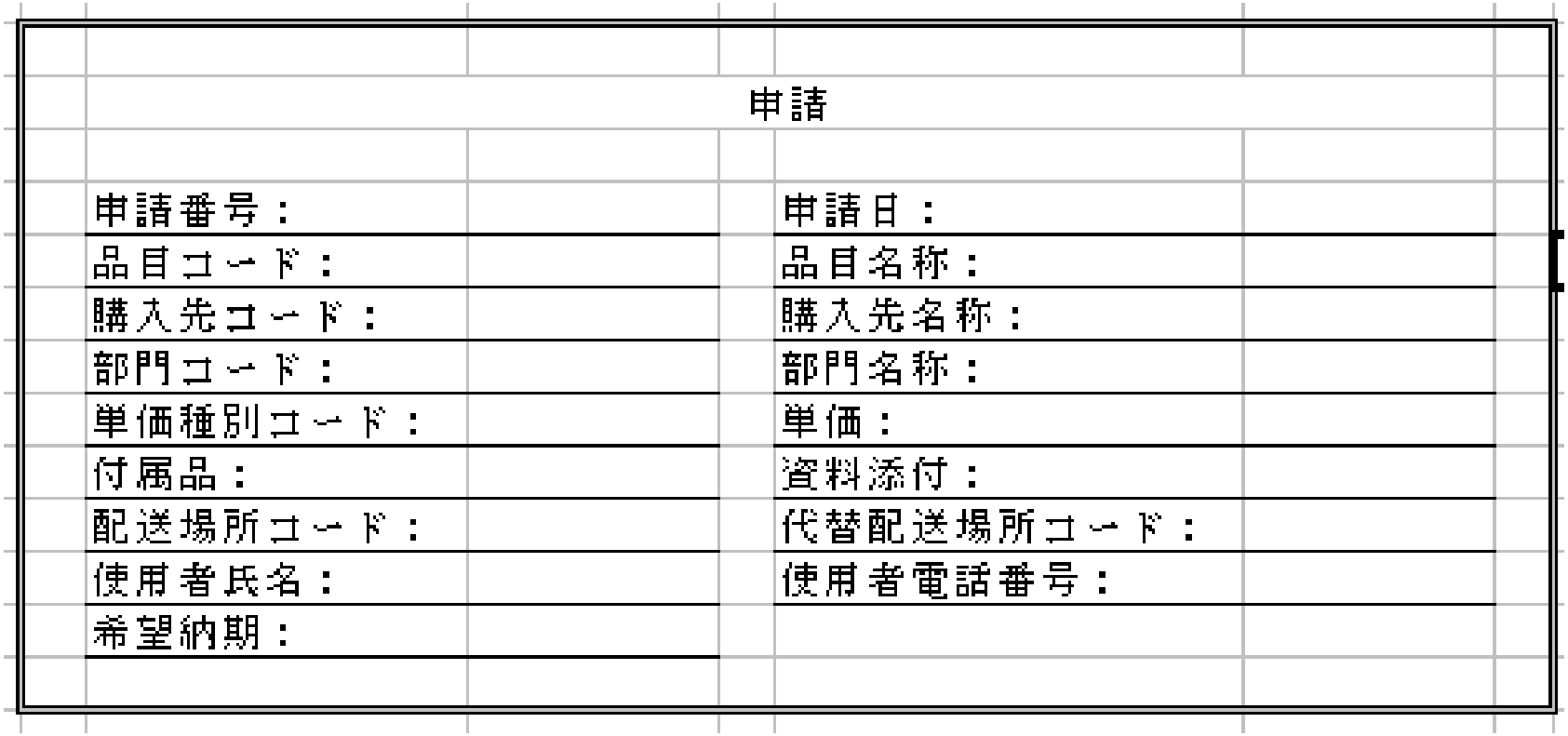
- 概念的スーパーセット
  - 制約束縛の確認をする
    - 箱と箱の間に線（関係）がある場合、箱の性質（アトリビュート）と線の先の箱の性質の関係を調べる
      - 直接関係だけでなく、間接関係もすべて調べる
      - たとえば、受注の受注日と請求の請求日には前後関係がある
    - 制約束縛関係があるが、箱の構成が異なっており、一見わかりづらいものには、概念的スーパーセットを用意して、線でつなぐ
  - 制約束縛関係はクラスとみなすことができる
    - RDBにはない概念のため、データモデルとしては実装しない
    - オブジェクト指向開発のときにはクラスを作る単位としてもよい
    - 全てが実態のあるクラスになるわけではない
      - 商品の商品単価と受注の受注時単価は「販売単価」とみなせるが、単価にそのような制約束縛が存在することだけを表す

4. 応用編:ビジネスを検証して、ソリューションを提案する

- 箱と箱の線(関係)を見る
  - eventはその事業内容、resourceはその会社の資産
  - 「eventの数＞resourceの数」だと、戦略の幅が狭くならざるをえない
  - resource関係をみれば、新しいビジネスを考えることができる
  - 新しいビジネス(event)のために、何が足りないか(resource)を調べる
- 箱の中身(性質)を見る
  - resourceの性質を見ると、何のために管理しているかわかる
    - 顧客の性質に住所しかない場合は、DMを出すだけだが、性別、年齢、地域、家族構成まであれば、CRMシステムになる
- 線がつながっている箱の中身(性質)の関係(制約束縛)を見る
  - ビジネス戦略は頻繁に変わるが、新規ビジネスを起こすのは稀であり、頻繁に起きるのは、資産をどう活用するか、制約束縛のコントロールである
    - 商品と取引先の間に、単価を下げる代わりに量を増やす仕組み(契約体系)を作る、など

5. 実演2：検証しながら書く

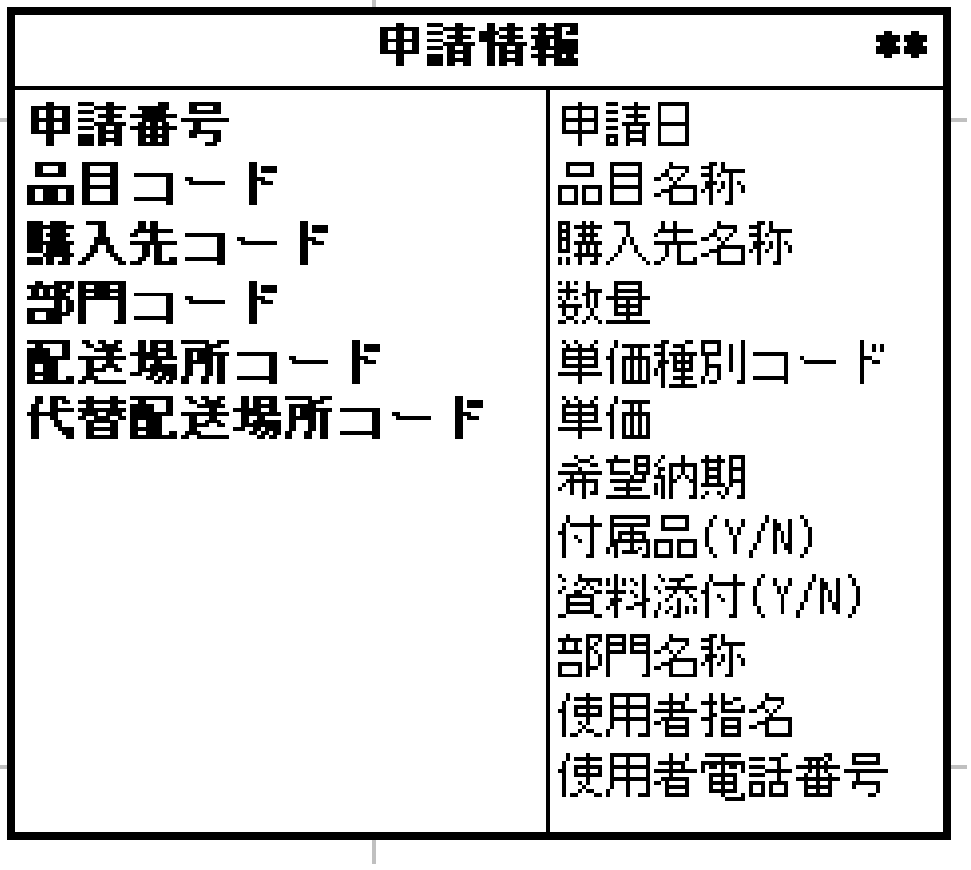
- ケース
  - 書籍「データ設計の方法：数学の基礎とT字形ER手法　論理データベース論考」のP.226～P.232
- step1.問題（事業）を捉える
  - ヒアリング or 現行システム帳票 or 画面



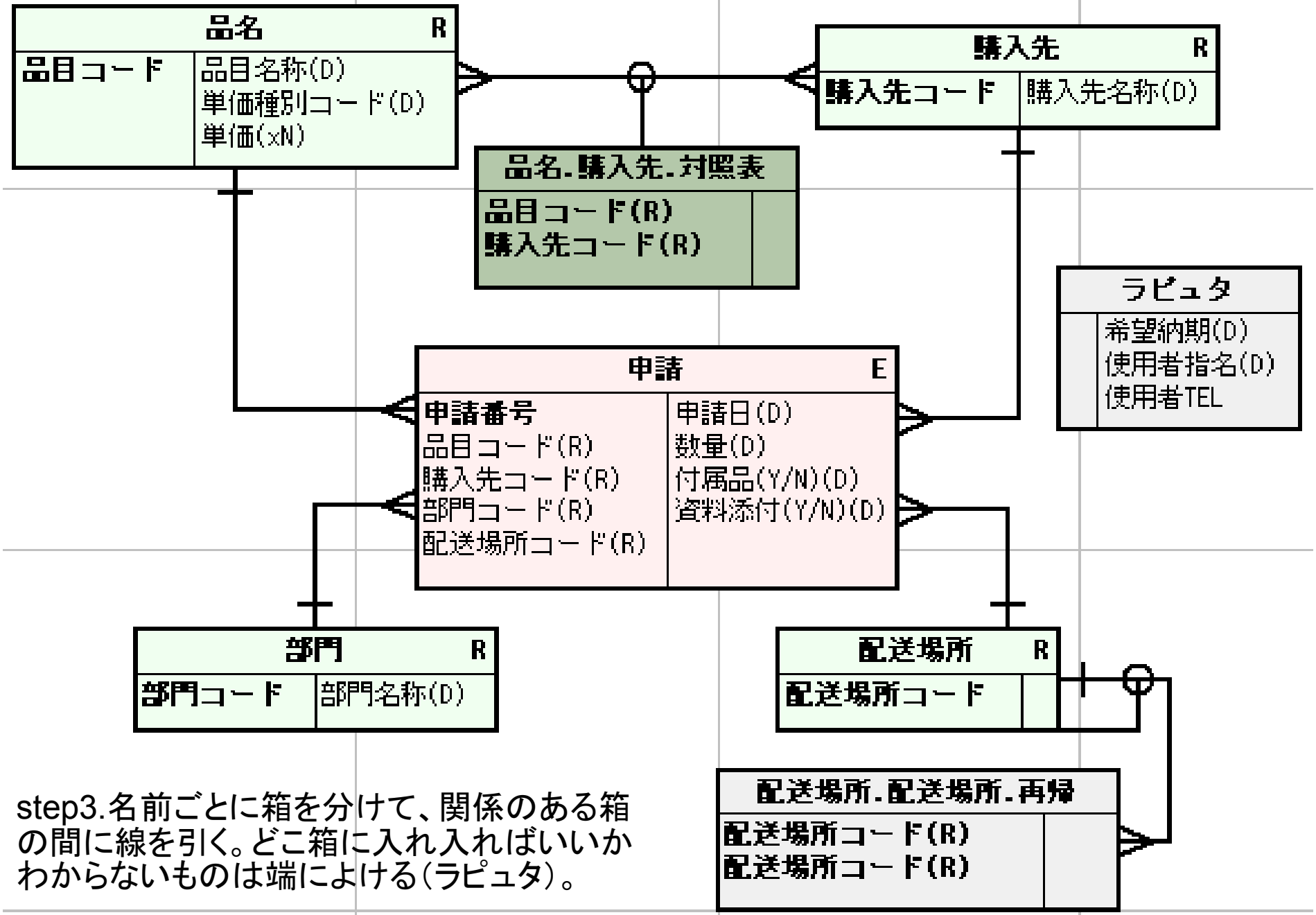
申請	
申請番号：	申請日：
品目コード：	品目名称：
購入先コード：	購入先名称：
部門コード：	部門名称：
単価種別コード：	単価：
付属品：	資料添付：
配送場所コード：	代替配送場所コード：
使用者氏名：	使用者電話番号：
希望納期：	

5. 実演2：検証しながら書く

- step2.項目を洗い出す
  - 箱の名前に抽出した情報元の名前
  - 箱の左側に番号、コードの項目（顧客が認知／管理している単位を表すもの）
  - 箱の右側にそれ以外の項目

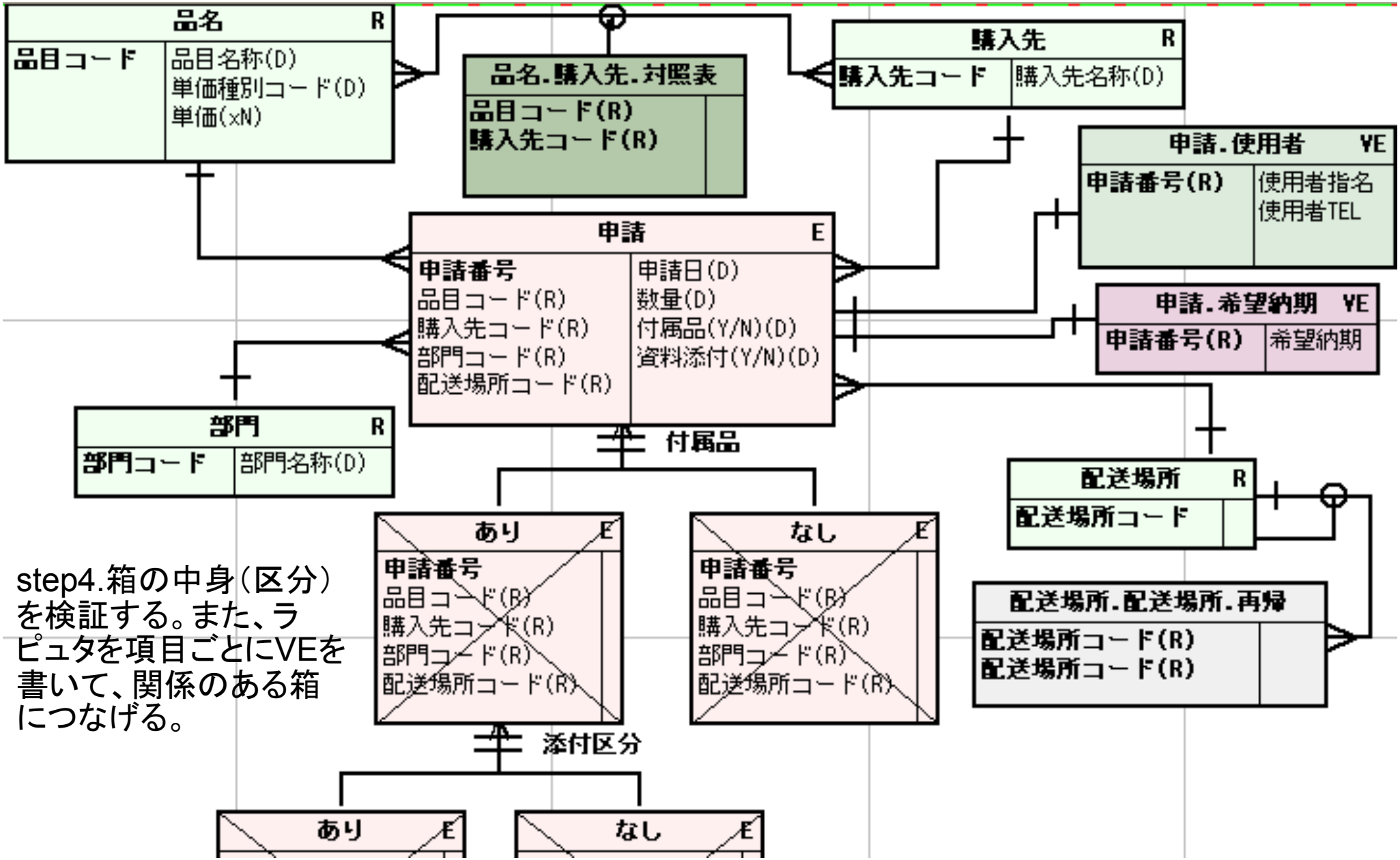


5. 実演2：検証しながら書く



step3.名前ごとに箱を分けて、関係のある箱の間に線を引く。どこ箱に入れ入ればいいかわからないものは端によける(ラピュタ)。

5. 実演2：検証しながら書く



step4.箱の中身（区分）を検証する。また、ラピュタを項目ごとにVEを書いて、関係のある箱につなげる。

６．発展編：展望、今後の課題

- 生成規則（意味論）の一般手続き
  - 無理？妥当性の検証要員だけはベテランが必要か？
  - ツールの自動生成機能に委譲してみるか
    - 全部クラスにすれば、GrailsとかMDAにつなげられる？
    - TMとしてはクラスの作成単位に拘っておらず、制約束縛（責任関係）さえわかればいい
- 開発方法論
  - TMは分析モデルを書いた結果、作りたいものが整理されるだけ
  - 製造はすぐ終わるが、分析に時間がかかるか
    - 短期間に動くものを要求される現代で受け入れられるか
    - 自動化ツールで、現状はここまで動く、というのを示すのもありだが
    - 運用後の保守開発や2次受け開発だと・・・（開発と言っている時点で）
  - 整理の仕方なので、既存プロジェクトの検証としても使えるか

7. 文献

- データ設計の方法：数学の基礎とT字形ER手法　論理データベース論考
  佐藤正美(著)、SRC（出版）
- ウィトゲンシュタイン『論理哲学論考』を読む
  野矢茂樹(著)、ちくま学芸文庫（出版）
- 株式会社SDIホームページ
  http://www.sdi-net.co.jp/