博士学位論文

情報システム開発に向けた
日本語要求記述からの概念モデルの構築と
その活用に関する研究

2019年7月8日

井田　明男

目 次

図 目 次

表 目 次

第1章　緒論

1.1　研究の背景と問題意識

情報システムは，人間の社会的・経済的活動を支える膨大な情報を処理（捕捉，蓄積，検索，加工，管理）するためのシステムであり，その構成要素として人間そのものを含む複雑なシステムである．そのため，システム開発の最上流工程，すなわち要求分析段階で作成する「概念モデル」[1] が重要である．概念モデルが不適切であったならば，開発される情報システムは，求められるものではなくなってしまう．

しかし，その重要性とは裏腹に，我が国では，概念モデルが広く SE（ソフトウェアエンジニア）の間に普及しているとは言い難い．本論文では，その大きな原因が，概念モデルの入力となる要求記述，即ち，業務課題やシステムに関する要望の論述が日本語である点に起因するとの独自の問題意識を提起する．「誰が，何を，どうした」という形式の行為文主体の英語とは異なり，「何が，どういった状況にある」という状況描写文（状態文）が多用される日本語記述は，概念モデルとの整合性が悪いからである．この日本語と概念モデルのギャップは，概念モデルを構築する際にも問題となるが，得られた概念モデルを用いてコミュニケーションする上でも，課題となる．因みに，英語や中国語については，それぞれの文構造および構成要素と実体関連図との対応を調べた研究（P. Chen による [1][2] など）が比較的古くから存在するが，日本語を対象にした研究は，金田らによる [3][4] など，最近になってからである．

そこで，本研究では，(1) 日本語要求記述からより良い概念モデルを構築すること，(2) 構築した概念モデルを開発の要求定義以降の作業分野においても有効に活用すること，を具体的なテーマとした．

本研究では，概念モデルを，「業務を成立させる本質的な要素を，業務プロセスの側面，および，業務プロセスで扱うオブジェクト（以下，ドメインオブジェクトと称する）の側面から抽出し，それらを時間的，あるいは空間的に配置した，組織的かつ実装独立で

[1] 本論文でいう「概念モデル」とは，ビジネスの本質を抽出して構造化したグラフであり，業務プロセス自体やプロセスで扱う対象の因果関係や相互作用を説明するための構造図である．また，概念モデルと類似した意味を持つ用語として「ドメインモデル」があるが，ドメインモデルは，システムに関わるさまざまな実体（entity）とそれらの関係（relationship）を説明するシステムの概念モデルである．従って，ドメインモデルの場合には，その表現として専らクラス図や ER 図が採用されるが，概念モデルの場合には，その表現には，データフロー図，ユースケース図，アクティビティ図，フローチャート，ステートマシン図などの動的なモデルが加わる．

はあるが実装にストレートに変換可能なモデル」，と定義する．従って，本研究の概念モデルには，ユースケース以外にも日本語ビリヤードボールモデルと称する動的モデルと，存在従属クラス図と称する構造モデル（以下，ドメインモデルと称する）の両方が含まれる．

また，概念モデルは，設計用途だけでなく，関係者の意思疎通や合意形成に用いられる．このため，誰もが簡単に表記でき，読みやすく，直感的に理解しやすいものでなければならない．そこで，分析結果の表記法についても，研究対象としている．さらに，概念モデルは，要求を可視化し合意して，それで終わりではなく，要求分析以降の開発の作業分野においても有効活用できなければならない．先に,「実装独立ではあるが実装にストレートに変換可能なモデル」，と定義したのはそのためである．具体的には，本研究では，構築した概念モデルを入力として，システムの機能規模の見積もりを行ったり，システムユースケースを自動生成したり，あるいは，データベースのスキーマ定義とデータベースにネットワーク越しのアクセス手段を提供するサービスの仕様の自動生成も研究対象に含めている．

1.2 本研究の目的

本研究の目的は，情報システムの開発を対象に，日本語要求記述を出発点とし，より良い概念モデルが構築できるように，また構築した概念モデルが有効に活用できるように，そのための方法について提案可能なモデリング手法，モデル化のためのガイドライン，そして，モデル化対象に適した設計成果物の表記を確立することである．

本研究の特徴としては，(a) 概念モデルの構築の際に，行為文よりも状況描写文が頻出する日本語の特性を考慮に入れていること，(b) ドメインモデルとしての存在従属クラス図を，ドメインオブジェクトの存在に由来する時間的前後関係の制約を含意した動的モデルと位置づけていること，(c) 多次元尺度構成法を応用した定量的な概念モデルの妥当性の確認を含めていること，(d) 概念モデルの構築だけでなく，その有効活用も含めた包括的な提案していること，を挙げることができる．

1.3 本論文の構成

本論文は，緒論，本編 6 章，および結論の全 8 章で構成されている．

第 1 章では，緒論として，研究の背景と問題意識，本研究の目的，および本論文の構成を記している．

第 2 章では，業務についての記述（日本語要求記述）をもとに，概念モデルとしてのイベント応答モデルを導く方策について論じている．具体的には，行為者から飛来する，

業務プロセス起動の契機に対して，業務で行うべき処理や応答を，認知言語学の成果の一つである，ビリヤードボールモデルを援用して考案した様式に則って形式化する手法について論じている．これにより，日本語要求記述の中でも注目すべきは，動態プロセス（そのプロセスの結果，世界の状態が変化するプロセス）であることを明らかにし，業務プロセスを，ビジネスイベントを単位に，概念的に独立した単位として切り分け，分割されたプロセスについて，参与者（ドメインオブジェクト）と，それぞれの状態変化を抽出して，形式的に記述することを可能とした．

第 3 章では，システム化の適用領域を，ドメインオブジェクト間の存在従属性に着目して，モデル化する研究の成果について論じている．本章では，これを存在従属分析と呼び，その成果物を存在従属クラス図と称する，UML のクラス図表記をカスタマイズした表記にてモデル化する提案をしている．存在従属分析のガイドラインは，母語の文法に依存せずに適用できる上，この手法により，正規化理論を陽に使うことなしに，第 4 正規形以上の正規化レベルを有するドメインモデルが得られることを確認した．これにより，存在従属クラス図は，十分に正規化された実体関連図（ER 図）と等価である，と言えるようになった．

第 4 章では，ドメインモデルを責務（ドメインオブジェクト群，および，それらの属性群と操作群）の 2 次元空間布置と見做した上で，既存手法によって得られたドメインモデルと存在従属分析によって得られたドメインモデルを比較している．既存手法の代表として，ソフトシステムズ方法論に着目し，概念活動モデルから知識の責務と振る舞いの責務を抽出するとともに，責務間の関連度合いを数量化する基準を提案している．既存手法によって得られたドメインモデルと存在従属分析によって得られたそれは，矛盾しない構造を有することを確認した．

第 5 章では，存在従属クラス図を入力として，業務システムのユースケース群を自動生成するとともに，同時にクラス間の依存関係を利用して，生成されたユースケース間の依存関係についても自動生成する研究について論じている．その結果，使用性（ユーザビリティ）を向上させるユースケース以外は高い網羅性にて抽出できることを確認した．

第 6 章では，存在従属クラス図から，それを扱う業務システムの機能規模を測定する研究について論じている．本章で提案している測定手法は，業界で広く用いられている COSMIC 法よりも，簡便でかつ，開発プロセスの早い段階で適用可能でありながらも，その測定結果は COSMIC 法を用いた測定結果と比較しても差がほとんどないことを確認している．

第 7 章では，存在従属クラス図を入力として，データベースのスキーマを生成し，それらを業務リソースとして，リソースに対する包括的なアクセス手段を提供するサービスの仕様の生成について論じている．存在従属クラス図をテキストベースで表記可能な文法を有する DSL(Domain Specific Language) を定義し，DSL で記述したドメインモデルを試実装のコンパイラで翻訳することにより，その後の開発作業で必要となる設計成

果物，すなわち，データソースのスキーマ定義，定義されたデータソースにネットワーク越しにアクセスする手段を提供する Web サービスインターフェース群，およびそのドキュメントを自動生成することができる．このことにより，アプリケーション・インターフェース (Application Interface: API) の原型定義に規則性と網羅性がもたらされたことを報告している．

以上, 本研究では，日本語要求記述から概念モデルを構築し，得られた概念モデルを有効に活用する手法の提案を行った．本研究の成果が，我が国の情報システム開発現場の改善に多少なりとも貢献できることを願っている．

第2章　業務事象の整理

2.1　緒言

提案依頼書などに記載される膨大な日本語要求記述をユースケースモデルに変換するには，記述中からビジネスについての本質的な記述を見つけて体系的に表現することが求められる．しかし，それは業務の流れをどこで区切るかといった問題や業務フロー上の個々の作業のまとまりをどのように捉えるかといった問題があるため，概して分析者によるバラつきの出やすい作業である．

本章では，ビジネスの要となる参与者[1]に状態変化を引き起こすエネルギー伝播に着目し，それらの事象をラネカー [5] らのビリヤードボールモデルからヒントを得て策定した表記（日本語版ビリヤードボールモデル：Japanese Billiard-Ball Model：以下，JBBM）にて形式化することを提案する．これにより，一連のビジネス活動を概念的に自立した系列として切り出せるため，分析者によるバラつきを抑制でき，後続のモデルへの接続もスムーズになることが分かってきた．

昨今，経営ニーズの変化に機敏に対応できる情報システムが求められている．そのためには情報システムの柔軟性を向上させねばならない．柔軟性を高めるには，ビジネス活動の系列単位で簡単にサービスを追加，変更できる仕組みを考える必要がある．そのためにはビジネスの側面とアーキテクチャ設計の側面の両側からアプローチが必要であるが，本章ではビジネスの側面からのアプローチを考える．

情報システム開発における最上流工程である要求分析では，要求仕様記述の手段として，最初に自然言語が利用され，それをもとにユースケース図，ユースケース記述，ドメインモデル（分析クラス図，概念モデル）などを含むユースケースモデルに変換される．

しかしながら，要求記述からユースケースモデルへの変換時には大きく３つの問題があると著者は考えている．

１つ目は，要求記述の量的な問題である．通常，提案依頼書などで施主[2]とベンダ[3]との間で交わされる要求記述量は膨大である．そのため，本質的な記述を見極めなければ

[1]この場合，参与者といっても，人とは限らず，ビジネスで捕捉，蓄積，管理すべき重要なオブジェクト（エンティティ）である

[2]システム開発の依頼者，もしくは発注者．

[3]システム開発の請負者，もしくは受注者．

ならない.

２つ目は，要求記述をユースケースモデルに変換する方法論上の問題である．自然言語で記述された要求記述を形式化された UML などのモデルに変換するための方法論はすでに数多く考案されているが，それらのほとんどが英語圏で育まれたものであるため，認知構造が日本語の影響を受けている [4] 日本人にとっては使いにくい場合がある.

３つ目は，元の要求記述文とユースケースモデルとの間のトレーサビリティが保たれないことである．自然言語による要求記述に対して，ユースケース図では情報量が少なすぎ，アクティビティ図で表現されるようなユースケース記述やドメインモデルでは詳細過ぎる.

そこで，適切な粒度で要求記述の本質を捉えるモデルが望まれる．すなわち，要求されるビジネスプロセスはどのようなタイミングで誰が始めるのか，どのようなエネルギー（依頼メッセージ）が伝搬し，何がどのように変化するのか，そしてどこで一段落するのか，を把握できるモデルである.

本章では，要求記述の中から本質的な記述を見極め，日本語を母語とする人にとって使いやすく，かつ，もとの要求記述とユースケースモデルとのトレーサビリティを担保できる形式化を提案する.

その際注目すべきは要求記述には，記述者の認知構造が反映されていることである．人間は自分が認知したことによって，記述し表現する内容を変化させるため，記述者によって認知された事象が，どのように言語化（一種のモデル化に他ならない）されるかに注目することは認知言語学のテーマであるとともに，この知見をソフトウェア工学に取り込めば，要求記述の読み手側にも，書き手側にも一定の指針を与えるだろう.

以下，2.2 節では，要求分析にビリヤードボールモデルを持ち込む理由を説明する．2.3 節では，日本語による要求記述の特性を考慮したビリヤードボールモデルの作成法を提案する．2.4 節では提案手法の例題への適用を紹介する．そして，2.5 節は本章の結言である.

2.2 ビリヤードボールモデルの援用

2.2.1 認知言語学とオブジェクト指向

認知言語学は，近年になって注目されてきた学問分野であり，G. レイコフの認知意味論と R.W. ラネカーの認知文法を中心に発展を遂げてきた.

前者は，言語が人間の他の身体能力や認知能力から切り離すことができないものとみ

[4] 言語はその話者の認知の形成に差異的に関与する，とする仮説としてサピア＝ウォーフの仮説が知られている.

なし，物事を捉える際の心の作用の観点から言葉を分析・説明する理論であり，後者は，人間が持つ一般的な認知能力の反映として言語を捉える理論である [6]．それらの成果として，物事の類似関係に支えられる「メタファー」，物事の隣接関係に支えられる「メトニミー」，事象の時間的位置関係に支えられる「事象認知スキーマ」などの概念が知られている．

それらは，オブジェクト指向の考え方にも根底で通じていると考えられ，メタファーは構造モデリングにおける Is-a 関係，メトニミーは構造モデリングにおける Has-a 関連，そして，事象認知スキーマは振る舞いモデリングにおけるオブジェクトの相互作用分析にそれぞれ概念的に結び付いていると想像される．

本章では，この中でも事象認知スキーマに注目する．事象認知スキーマは，人間にとって事象認知は決して無秩序に行われるのではなく，一定の枠組みに当てはめる形で行われるという考え方を受けた概念である [7]．そのため，英語（およびそれ以外の言語）における発話者の主語，目的語，斜格の選択は無秩序ではないとされる [8] が，英語と日本語とでは異なるため配慮が必要である（2.3.2 節で述べる）．既によく知られた事象認知スキーマのモデルの一つとして，ラネカーらのビリヤードボールモデルがある [5]．

2.2.2　ビリヤードボールモデル

すべての言語現象が，その意味の値を捉えるために時間の軸，すなわち処理時間の経過に伴ってどのように展開するか，が重要だという点で,「動的」な概念化に基づく記述を必要とする．この概念化には注意の焦点が順次移行するプロセスが含まれる．ビリヤードボールモデル（以下，BBM）は，そのような事象を分かりやすく説明するのに適したモデルである [5]．

BBM のモデル要素は，空間，時間，参与者 ，エネルギーである．空間は発話者が切り出した事象であり，その中で参与者から参与者へのエネルギー伝搬を時間軸にそって左から右へ順番に表記していく．W．クロフトの書き方 [9] に従えば，たとえば,「太郎が斧で木を切った」という事象は，太郎から斧へ，斧から木へ，木から木へのエネルギー伝播の連鎖として，図 2.1 のように表わすことができる．

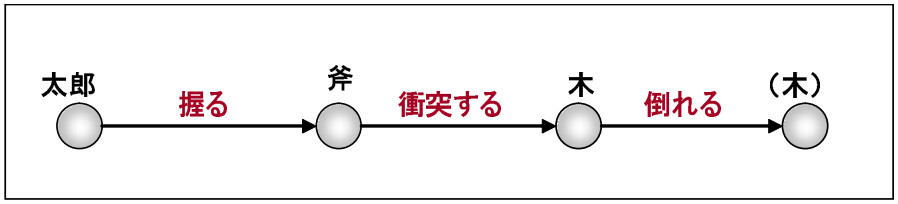

図 2.1:「太郎が斧で木を切った」の BBM（クロフト流の表記）

BBM の表記規則は，参与者は丸印で表し，エネルギーは発生元の参与者から受け手の

参与者への矢印で表す，と単純であるため，かなり自由に描くことができる．図 2.1 では，斧との衝突の結果，木に生じる状態変化もまたエネルギー伝搬，つまり木から木自体への再帰的なエネルギー推移として表している．木を複数回表記しているのはこの意味であるが，複数回表記する煩雑さを解消するには図 2.2 のように表記することもできる．

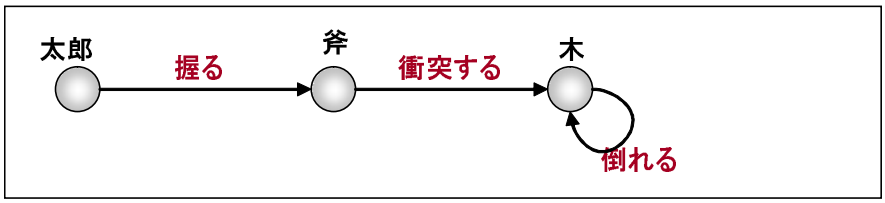

図 2.2:「太郎が斧で木を切った」の BBM（表記を圧縮）

また,「太郎は金の斧を花子に贈与した」のような二重目的語構文である場合も，図 2.3 のように表記することができる.「太郎」から発せられる「贈与する」というエネルギーは「金の斧」と「花子」に向けてそれぞれ伝搬する様子が表現できる．

事象を BBM で表現すると，最初に事象を引き起こす参与者が明確になり，太郎から斧へのエネルギーの伝搬のような，元の文章には記載されていないエネルギー伝搬が見つかる場合がある．そして，事象は規則的な $S + V + O$ の連鎖で表現されるため，何らかの事象の結果，状態が変化する参与者を見つけやすくなる．

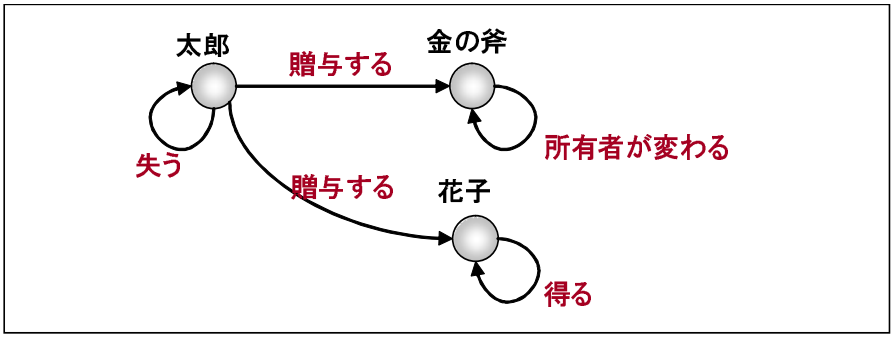

図 2.3: 二重目的語構文節の BBM（「太郎が花子に金の斧をあげる」）

2.2.3　事象のスキーマタイプと BBM

しかしながら，BBM はあらゆるタイプの事象表現に適しているわけではない．J. テイラーと瀬戸は文が表すプロセスを大きく 4 つのスキーマタイプに分類している [8]．すなわち，表 2.1 の第 2 列に示す，1）動態プロセス，2）状態プロセス，3）認知プロセス，4）複合プロセスである．

1）の動態プロセスはさらに,「そのプロセスの結果，世界の状況が変化する」サブタイプ（表 2.1 中 1-a）と,「ある状況へのエネルギーの注入はあるが，そのプロセスから状

表 2.1: テイラーと瀬戸によるプロセスタイプの分類（表中の例文は著者がアレンジした）

タイプ#	プロセスのタイプ	説明	例文#	英語の例文	日本語の例文
1-a	動態プロセス	そのプロセスの結果，世界の状態が変化するタイプ	1	The company listed.	会社は上場した．
			2	Taro orders goods.	太郎は商品を注文する．
			3	Taro gave Hanako the golden ax.	太郎は花子に金の斧を譲渡した．
1-b		ある状況へのエネルギーの注入はあるが，そのプロセスから状態変化が生じないタイプ	4	The telephone rang.	電話が鳴った．
2	状態プロセス	エネルギーの注入も状態変化も生じない．ある状況が単に存在する，あるいはそのまま継続するタイプ	5	Today's sales amounted to one million yen.	本日の売上高は100万円になった．
3	認知プロセス	心的プロセスであり知覚的プロセスでもあるタイプ	6	I heard the news.	私はそのニュースを耳にした．
4	複合プロセス	上記いずれかのタイプの組合せに分割できるタイプ	7	Taro returned the book to the library.	太郎は本を図書館へ返却した．

態変化が生じない」サブタイプ（表 2.1 中 1-b）に分類される．表 2.1 は，それらに例文を加えてまとめたものである．

例文１は，自動詞の例で，会社そのものの状態が未上場状態から上場状態へと変化している．例文２は，目的語を１つとる動詞の例で，新規注文が生まれ，商品がそれに引当てられる．例文３は，二重目的語をとる動詞の例で，物品の所有権が変化している．例文４は，例文１に似ているが，電話が鳴り終わったあと世界はまたもとの状態に戻るため，例文１とは区別されるべきものである．例文５は，金額という特性（属性）が主語である「本日の売上高」に存在しており，その値が 100 万円に達したことを表わしている．例文６は，例文２と文型は同じであるが，例文２ではエネルギーの伝搬が主語の外部に向かうプロセスであるのに対して，例文６ではエネルギーの伝搬が主語の内側に向かう内的なプロセスである．例文７は，分解すると３つの部分プロセスからなる [8]．すなわち,「太郎がその本に何かをする」と「その本が位置を変える」と「その本が新しい場所（図書館）に落ち着く」である．

ちなみに，中村の「もの・こと」分析によれば，仕事とは「行わなければならないことを，体や頭を使って行うこと」とされる．つまり，仕事は行為（activity）であり，行為を対象に働きかける（すなわちエネルギーを注ぐ）ことで,「はじめの状態」を「終わりの状態」に変化させることと定義している [10]．この定義にしたがえば，要求記述中の文のなかでも表 2.1 の 1-a のタイプの文節に注目すべきことが示唆される．BBM が適しているのはまさにこのタイプの事象であり，両者は一致している．

2.2.4 BBM に登場する参与者の役割

BBM において参与者は何らかの意味役割を演じる [5]．意味役割は動詞が主語や目的語に与える役割のことである.「θ 役割」,「深層格」など研究者毎にいくつかの呼称があるが，文献 [8] と文献 [11] に共通で紹介されている役割を抜き出すと，表 2.2 に示す９種類にまとめられる．

表 2.2 の意味役割は互いに排他的ではない．つまり，１つの参与者が複数の意味役割を担う場合がある．たとえば，行為者が行為の結果，受益者になったり，ある事象の被動者

表 2.2: 文献 [8] と文献 [11] に共通で紹介されている役割

役割名称	説明
行為者(agent)	変化を引き起こす意図的な存在
起因(author)	変化を引き起こす非意図的な事物
道具(instrument)	動作主の制御下にある事物
被動者(patient)	ある種の状態変化を被る対象
経験者(experiencer)	心的状態の変化を被る対象
受益者(beneficiary)	変化によって利益を被る人
主題(theme)	変化によって移動する事物
起点(source)	主題の移動の出発点
到達点(goal)	主題の移動の到達点

が，つぎの事象の行為者になったりする．また，役割はさらに細分化できる．たとえば，到達点を空間的到達点と比喩的到達点に分ける提案もある．

意味役割の種類は少なくないが，BBM にモデル化されるエネルギー伝搬の視点で参与者の役割を考えた場合は，役割の種類は制限できる．すなわち，エネルギーの意図的発生源である行為者，エネルギーの受け手である被動者，エネルギーの伝搬手段である道具，およびエネルギーに付随して授受される主題の４つの意味役割だけで網羅できるだろう．

たとえば,「太郎が斧で木を切った.」であれば，太郎 は「行為者」，斧は「道具」で，木は「被動者」である．また,「太郎は花子に金の斧を贈与した.」であれば，太郎 は「行為者」，花子は「被動者」で，金の斧は「主題」である．意味役割の導入によって，エネルギーから見た参与者の役割が鮮明になる．

2.3 日本語向きビリヤードボールモデル

2.3.1 概要

2.2 節では，状態変化プロセスについて BBM の概念メタファーを用いて事象をプロファイルすると，事象が空間のなかで，規則的な $S + V + O$ の連鎖で表現できることを確認した．さらに，意味役割を導入することによってエネルギー伝搬についての行為者，被動者，主題，道具が明確になることも示した．このように BBM は，事象についての記述を形式化する際に役立つとしても，日本語テキストから作成する場合は，相応の配慮が必要である．そこで，本節では，日本語記述文の事情に合わせた日本語向き BBM（JBBM と称する）を策定し提案する．

2.3.2　日本語についての配慮

英語の文の構造は，そもそも 2 つ参与者（行為者と被動者）を動詞でつなぐ構造になっており，それを左から順番に並べて記述しているため BBM を描くのには労を要しない．

一方，日本語の文の構造は事象を取り巻く参与者を先に述べておき，最後に動詞を持ってきて参与者の関係を一気にまとめ上げる構造である（図 2.4）．金谷はこれを盆栽構造と呼んだ [12]．結果として日本語では，参与者間のエネルギーの伝搬関係が読み取りにくくなる．しかし，この状況は状態変化プロセスを一定の様式で捉える枠組を与えることによって改善されると考える．

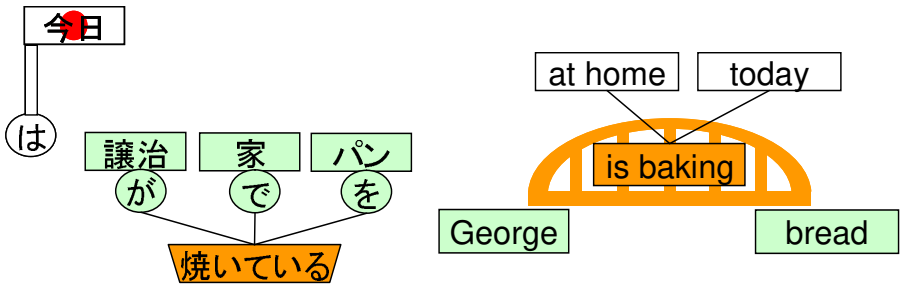

図 2.4: 日本語の盆栽構造（左）と英語の挿入構造（右）

2.3.3　JBBM 作成プロセス

提案手法では，図 2.5 に示す様式を利用して，要求記述を BBM に変換する．2.2 節で言及した通り，オリジナルの BBM では僅かなモデル要素と表記規則しかない．これは，$S + V + O$ が明確な英語には適しているが，日本人が事象を形式的に整理する用途には使いにくい．そこで，2.2.4 節で確認した意味役割等の欄を設けてビジネスを整然と記述できるように配慮したものである．本様式を仮に JBBM と命名する．JBBM の作成手順は以下のようになる．

【Step 1】状態変化プロセスの発見
日本語要求記述から表 2.1 の 1-a に示すタイプの文を見つける．見つけたらその行番号を「記述行番」欄に記入し，新たな事象 ID を採番して「事象 ID」欄に記入する．

【Step 2】トリガ事象を見つける
当該プロセスを始める契機となる別の事象がある場合には，その事象 ID を，請求締日の到来など業務カレンダー等で開始されるプロセスの場合は，その時期を，無い場合は「なし」，記載されていない場合や読み取れない場合には,「言及なし／不明」等を「トリガ事象」欄に記入する．

【Step 3】エネルギーの識別
当該状態変化プロセスにおけるエネルギーの伝搬を見つける．普通，エネルギー伝搬は

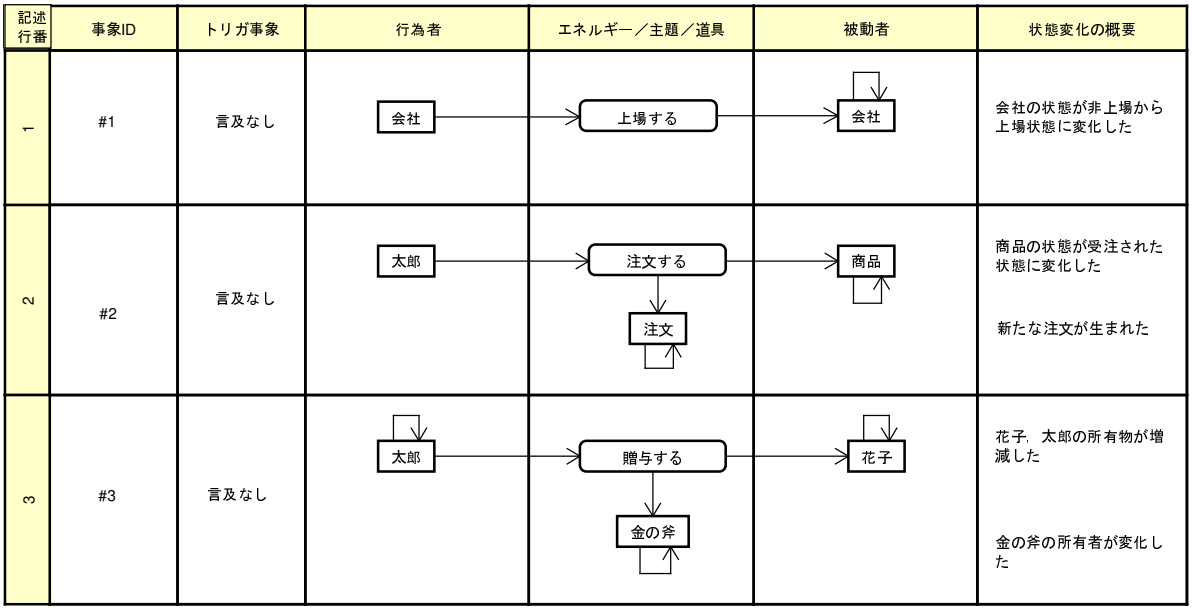

図 2.5: 提案手法の様式（JBBM） 表 2.1 中の例文 1 から 3 を JBBM で表現した例

動詞で記載されているはずであるが，日本語の場合，サ行変格活用動詞の語幹は名詞に見えるので注意する．見つかれば,「エネルギー／主題／道具」欄に当該エネルギー伝播をラウンドの四角で記入する．そして行為者，被動者，主題・道具が見つかれば，それらを名詞 (または名詞句) で，それぞれを該当する欄に記入する．なお，これらの参与者は矩形で表記する．

【Step 4】参与者の状態変化の識別

当該プロセスによって状態変化が考えられる参与者には，再帰矢印を描いて参与者自身の状態が変化することを示しておく．また,「状態変化の概要」欄にはその概要を記しておく．この欄は非常に重要であり，この欄の記載事項を見れば中村 [10] のいう仕事の「終わりの状態」が読み取れるようにしておく．

2.4 提案手法の例題への適用

2.4.1 JBBM の例題への適用

本節では，例題を通じて実際に日本語要求記述から JBBM を作成し，その後の成果物に繋いでいくプロセスを示す．例題は提案依頼書に記載された要求記述の一般的な例として，日経 IT プロフェッショナルに掲載された「RFP（Request For Proposal）サンプル」から引用した．図 2.6 の記述は，中古車販売業務システム化プロジェクトの提案依頼書「業務::中古車販売業務の流れ」[13] からの抜粋である．

図 2.6 の要求記述に 2.3 節で紹介した手順を適用する．１行目の文は，この文単体で見

【中古車受注プロセス】
1:お客様は, 中古車の購入を希望します.
2:販売担当は, 在庫管理担当に在庫を問い合わせます.
3:在庫管理担当は, 在庫を伝えます.
4:販売担当は, 在庫の中からお客様の購入希望に合う中古車の見積を作成します.
5:お客様は, 中古車を注文します.
6:販売担当は, お客様が顧客登録されているかどうかを判断します.
7:顧客登録されていなければ, 販売担当は, お客様を登録します.
8:販売担当は, 中古車を受注します.
9:在庫管理担当は, 在庫を引き当てます.

図 2.6: 日本語要求記述の例：「業務::中古車販売業務の流れ」[13] から抜粋

た場合は，行為者である「お客様」の認知プロセスとも考えられるが，２行目ではその結果を受けて，販売担当が在庫担当に在庫を問い合わせている．したがって，行為者「お客様」，エネルギー「伝える」，主題「希望中古車」，被動者「販売担当」による状態変化プロセスと判断できる．このプロセスによって被動者である「販売担当」の状態が「お客様の希望中古車を把握していない状態」から「お客様の希望中古車を把握している状態」に変化する．

２行目と３行目の文は，販売担当が在庫管理担当に在庫を問い合わせると，その回答が返ってくることを述べているため，２つの文で１つの事象を表していると判断する．トリガ事象は陽に記述がないが，１行目のプロセスの完了がトリガになっていると考えられる．行為者「販売担当」，エネルギー「問い合わせる」，主題「在庫中古車」，被動者「在庫管理担当」である．そして次の瞬間には「在庫管理担当」が，今度は行為者の役割で,「販売担当」に「在庫中古車」を「伝える」．その結果,「販売担当」の状態が「在庫中古車を把握している状態」に変化する．

４行目の文のプロセスタイプは，典型的な状態変化プロセスである．行為者「販売担当」，エネルギー「作成する」，主題は「希望中古車」および「在庫中古車」，被動者は「見積」で,「見積」の状態が「新規作成された状態」に変化する．

５行目と８行目の文は，同じ１つの事象について主語を変えて述べているため，１つの事象としてモデル化する．この事象のタイプも，典型的な状態変化プロセスである．行為者「お客様」，エネルギー「注文する」，主題は「在庫中古車」と「注文」．被動者は「販売担当」で，主題である「注文」が「新規作成済み」に変化する．

７行目の文には,「顧客登録されていなければ」という条件が記述されているが，JBBM では無視する．なぜならば，JBBM の段階で条件分岐までをモデル化すると不必要に詳細なモデルが出来てしまうからである．以降の文についても同様にモデル化を進めていく．

図 2.6 上記の分析結果を JBBM の様式にまとめた結果である．日本語の要求記述中の状態変化プロセスを抽出し，エネルギー伝搬を中心とした４つの意味役割で中古車受注プロセスを形式的に整理することに成功した．

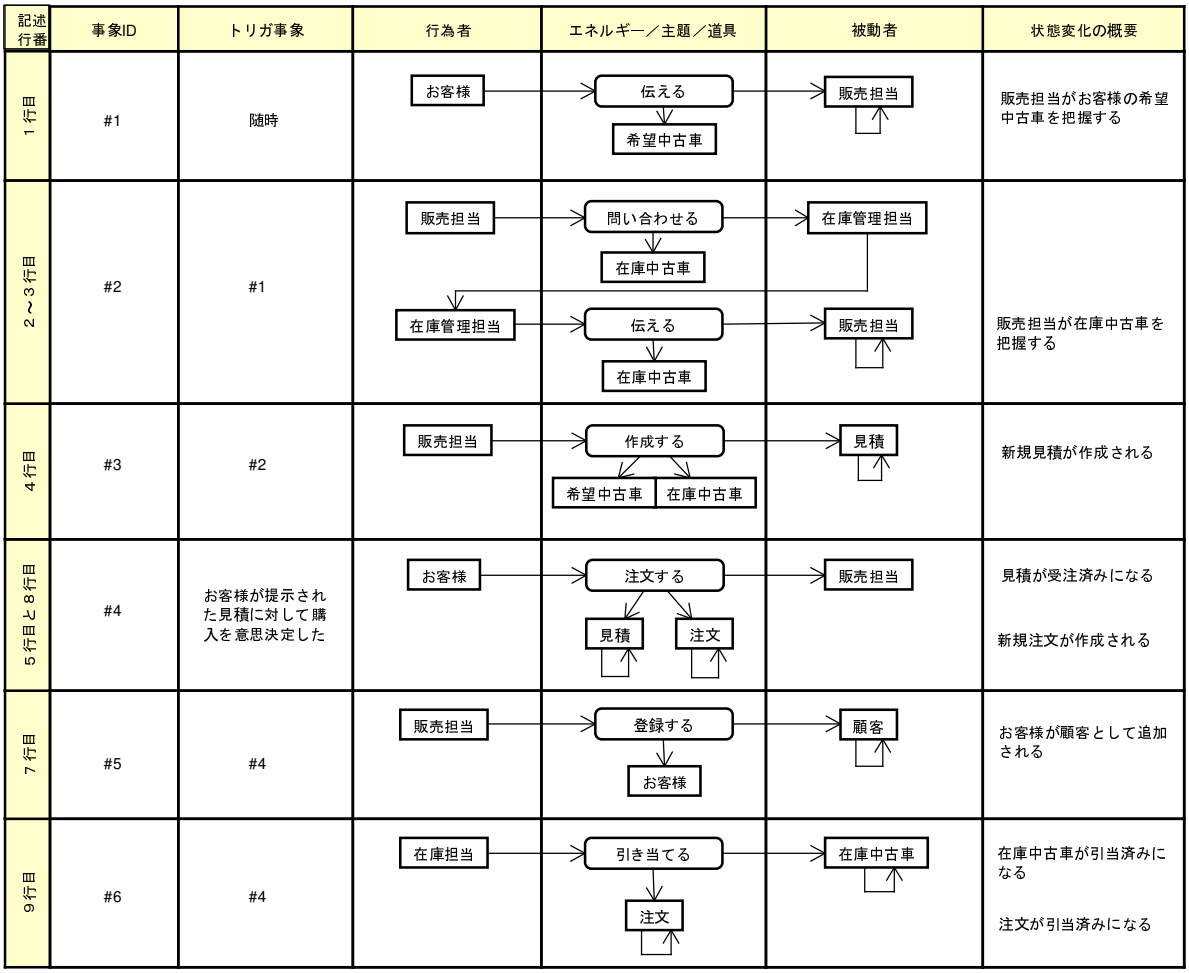

図 2.7: 要求記述の例題（2.6）から作成した JBBM

2.4.2　JBBM からユースケースモデルへの接続

現在のところ JBBM の用途は，要求記述とユースケースモデルの橋渡しを行うためのモデルであると考えている．図 2.8 は JBBM からユースケースモデルへの接続について概観した図である．周知の通りユースケースモデルの要素には，ユースケース図，ユースケース記述，ドメインモデル（概念クラス図）などがある．

(1)　JBBM からユースケース図への接続

ユースケース図のアクターは，JBBM 中の行為者がアクター候補となるが，たとえば，請求締日の到来など，時間的な事象によって起動されるプロセスの場合は，トリガ事象欄に記載した事象がアクターになる場合もある．また，ユースケースは，JBBM 中のエネルギー欄に記載した動詞と，その動詞によって状態が変化する参与者を目的語として「目的語（を）＋動詞」で命名したものがユースケース候補となる．

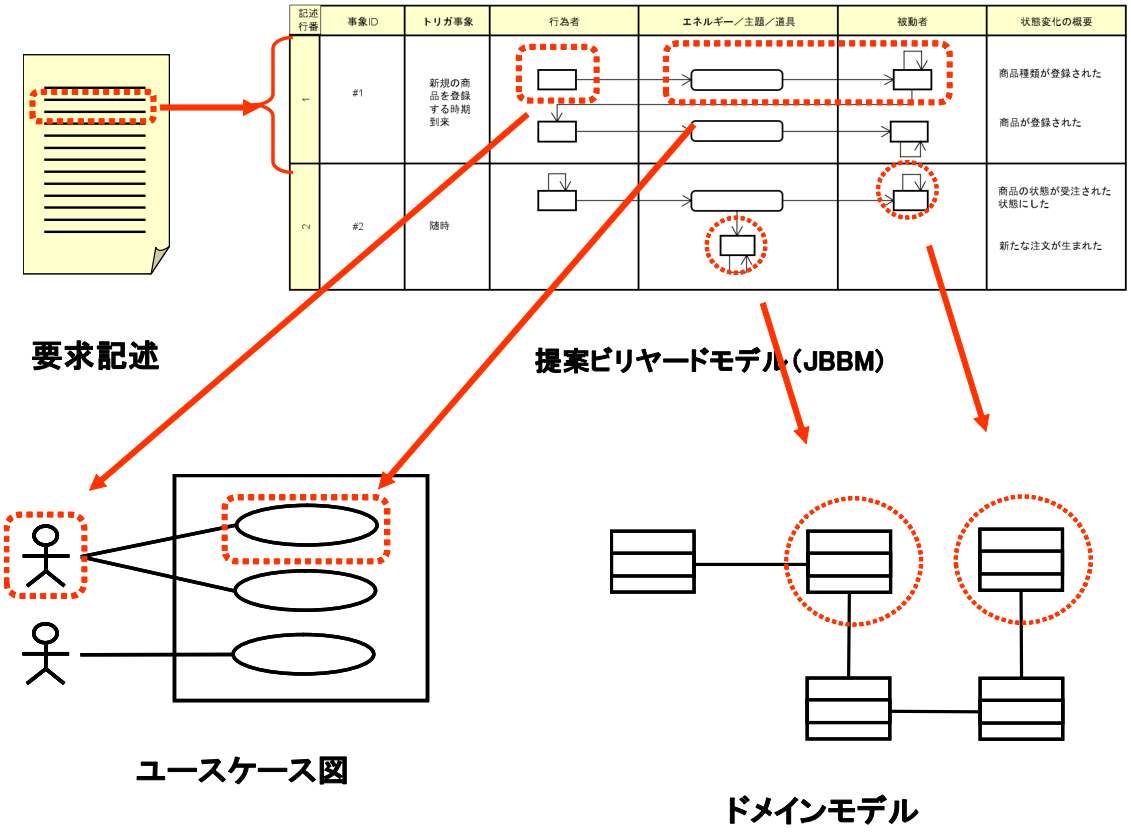

図 2.8: JBBM からユースケースモデルへの接続イメージ

中古車販売管理の例題では，アクター候補として「お客様」,「販売担当」,「在庫担当」が識別された．また，ユースケース候補として「希望中古車を伝える」,「在庫中古車を注文する」をはじめとする 7 つのユースケース候補が識別された（図 2.9）．ただし，この段階で識別されているアクターとユースケースはあくまでも候補であり，後続の分析によってアクター候補が機械に置き換えられたり，複数のユースケース候補が 1 つのユースケースに統合されたりする可能性がある．中古車販売管理の例題では，たとえば，在庫担当というアクターは後で機械（システムタスク）に置き換えられる可能性がある．

(2)　JBBM からユースケース記述への接続

ユースケース記述の作成では，JBBM 中の状態変化に注目する．具体的には，状態変化の概要欄の記載事項を中村 [10] のいう「終わりの状態」とみなし，JBBM で明らかになったトリガの発生から「終わりの状態」に至るまでのアクター（行為者）とシステムとの対話の流れを基本系列として記述していくことになる．ただし，実際にはアクターの特性によって，アクターに求める入力アクションや画面遷移など，ユーザビリティに関わる要素も含めて対話を組み立てていく必要があるため，別途ガイドラインが必要である．

(3)　JBBM からドメインモデルへの接続

ドメインモデルは問題領域固有のエンティティとそれらの間の関係を表したクラス図である．エンティティは業務において，その状態の進捗を捕捉，蓄積，加工すべき重要な管理対象であるため，通常は永続化対象のクラスが該当する．したがって，JBBM 中の参与者のうち，状態が変化する参与者はそのままクラス候補となる．たとえば，図 2.7 において状態変化を示す再帰的な矢印が表記されている参与者は,「販売担当」,「見積」,「注文」,「顧客」,「在庫中古車」の５つであり，これらはすべてドメインモデルのクラス候補となる．その際もしも参与者の名称が有標 のままで残っている場合は，一般化して無標 にする必要がある．クラス間の関連は，JBBM における参与者間のエネルギーの伝播関係から導かれる．中古車販売管理の例題では，図 2.7 の３行目を例にとると「販売担当」が「在庫中古車」について「作成する」というエネルギーを注入した結果生まれた「見積」から,「販売担当」と「見積」，および「在庫中古車」と「見積」の間にそれぞれ関連が導かれる．他の行についても同様に関連を導き出すと図 2.10 に示すドメインモデルが得られる．ただし，識別子，属性，多重度については別途検討が必要である（第 3 章で検討する）.

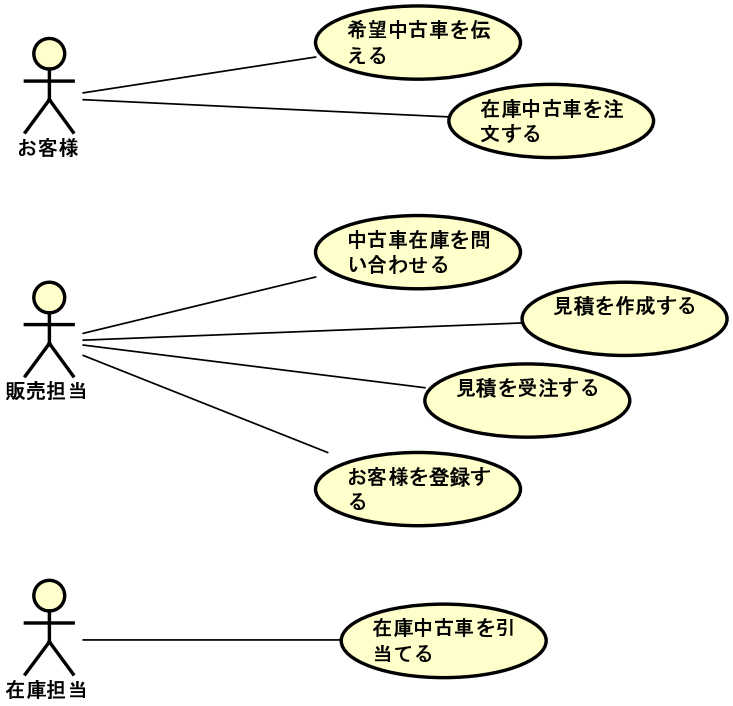

図 2.9: 図 2.7 の JBBM から導出したユースケース図

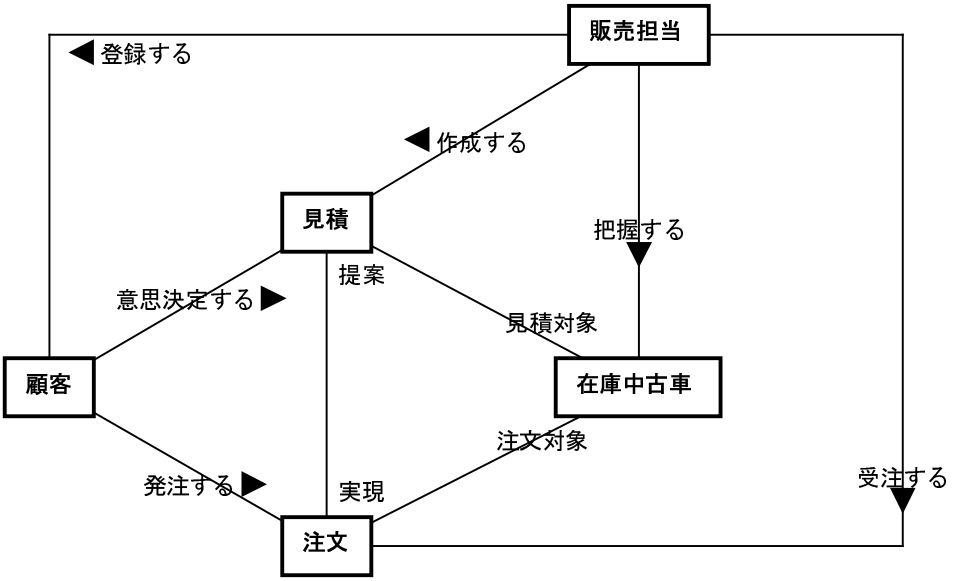

図 2.10: 図 2.7 の JBBM から導出した概念モデル

2.4.3 JBBM の導入効果

以上の検討結果を踏まえて，著者が JBBM を考案する際に気付いた期待効果についてまとめておく．

イ）BBM は事象に対する人間の認知プロセスをそのままモデル化したものであるため，理解しやすく，実践しやすい．その特性を日本人にも活かせるように JBBM の様式を検討した．

ロ）参与者（名詞）→ エネルギー（動詞）→ 参与者（名詞）という一定の形式の連鎖で事象を記述するため，参与者間の因果関係が明確になる．特に行為の主体たる主語が省略されがちな日本語[5] では記述の曖昧性を抑制する効果が高い．また，日本語に限らず要求記述に登場する語を名詞として扱うか動詞として扱うか，について迷う場面があるが，それらを記述からそのまま抜き出すのではなくエネルギーの伝搬として識別するため迷いが解消する．

ハ）JBBM の 1 行は，必ず，エネルギーの発生源である参与者がエネルギーを発した時点から始まり，参与者によるエネルギーの伝搬が一段落する時点で終わる．そのため，一連のビジネス活動を時系列的な観点からも概念的に自立した系列として切り出すのに有効である．

ニ）参与者（名詞）の間に，必ずエネルギー（動詞）が挿入されるため，事象を成り立

[5]正確には，日本語では,「誰が何をどうした」を描写する行為文よりも，その結果として,「何がどういう状況にある」を描写する存在文の方が好んで（あるいは高頻度で）用いられる [12][14]，ため，行為の主体を述べるべき機会が少ない．

たせるために必要なエネルギー伝搬とその対象を見落とすことが少ない．特に，日本語の場合は，盆栽型構造で事象が記述されるが，JBBM を作成する過程においてそれらは，英語と同じ構造に変換される．要求記述について JBBM を先に作成してしまえば，著者が併せて提案している，日本語仕様記述から概念クラス図への変換法 [15][16] とも親和性が高く，概念クラス図の作成もストレートフォワードな作業になる．

2.5 結言

本章では，認知言語学の知見を援用しながら，日本語要求記述から JBBM を作成し，それを経由してユースケースモデルに繋げるアイデアを示した．概念モデルが本質的な構造モデルであるならば，JBBM は本質的な振る舞いモデルとして認知されていくことを望んでいる．

今後の課題は，実用化に向けて様式も含めた JBBM 作成プロセスそのものと，ユースケースモデルへの接続プロセスを，とりわけアジャイル開発現場での適用を通じてモデルを蓄積し，改良していくことである．

JBBM 作成プロセスの改良のひとつに，格助詞のコアイメージの積極的な活用が考えられる．要求記述に登場する格助詞に対して，そのコアイメージを取り出すことで，エネルギーに伝搬に関わる参与者の役割をより正確に識別できる可能性がある．著者はすでに，日本語仕様記述を概念クラス図に変換するメソッドを提案しており，そこでは重文・複文が多いとされるビジネス要求記述を単文化する際に参照できる，12 のパターン・テンプレートを提示している（表 2.3）[16]．これは，岡等の日本語格助詞のコアイメージ研究 [17] を取りまとめた，森山 [18] の格助詞に対するコアイメージ区分に従って作成したものである．このテンプレートに照らして単文化を行うと，単文パターンに応じた概念クラス図の部分が求まり，それらを結合すると記述ドメインの概念クラス図を得ることができる．

しかし，日本語の格助詞の用法は至って多義的であり，「てにをは」は合致していても，文章のコアイメージは複数存在することが知られている [6]．そのため，意味役割の正確な識別を目的とした格助詞のコアイメージの使用には，さらなる研究が必要である．また，JBBM からユースケースモデルの接続プロセスについても，ユースケース記述への接続や多重度の決定を含むドメインモデルへの接続について検討の余地を多く残している．継続的に JBBM を実際のビジネスモデリングシーンに適用しながら効果を検証し，JBBM の改良に取り組んでゆきたい．

[6] 興味深いことに，森山 [19] は，格助詞の「ガ」と「ニ」についてはその用法を「プロセス的用法」と「存在論的用法」に分けてそれぞれのコアイメージを説明している．

表 2.3: 認知言語学と関数従属性から導かれたクラス図作成ガイドライン

項番	区分	ガイドラインの内容	事例および補足説明
1	ドメインモデル全体	可算名詞がクラス名，不可算名詞が属性名，属性値，状態動詞が関連，そして，動作動詞が操作に該当する．日本語仕様記述を読み取る際には，この4種類の区別を意識する必要がある	例：自動車，債務等は可算名詞，砂，朱色等は不可算名詞，「歌う」は動作動詞，「雇用している」は状態動詞．日本語の状態動詞は「ある」「いる」しか無いが，この2 つが付いていても，状態動詞であることが釈然としない場合が多い．文脈を考慮して判断すべきである
2		クラス図は，時間的推移を捨象して，ビジネスの途中で，一定期間に渡って成立する状態を表現する．クラス図には，「時間的経過」の情報はない．時間的経過は，相互作用図などの動的モデルの担当である	関係性が成立する期間とそうでない期間が存在する場合には，当該関係性は「関連」とする．関係が当該クラスのインスタンスが存在する期間中は時間的に変化しない関係は，クラス中の属性として表現するか，又は「合成集約(has-a)」関連として表現される
3		日本語は「存在文」を多用する言語であり，英語は「行為文」中心である．行為文（する型）にしないと，関数従属性が表現に現れ難いので，日本語仕様文は，単文化する際に，日本語で行為文に書き換え，主語を明確にするべきである．とりわけ，日本語では少ない，非生物主語を多用することになる	「机には4本の足がある」⇒「机は4本の足を持つ．」クラス図は英語の認知構造であり，特に，「関連」はエネルギーの伝搬（関数従属性）を意味するので，日本語の存在文では，関数従属性を取り出せない
4	エンティティ・クラス	クラスと属性の関係は，当該クラスのインスタンスが生成されるとともに属性値が生まれ，インスタンスがこの世に存在する限り属性値が存在し，そのインスタンスがこの世から消えるとともに，属性値も消える，そのようなライフサイクル一致の関係である．結果，必然的にクラスのすべての属性はクラスに関数従属することになるため，第三正規形が保たれる	「飲み物」の「値段」は，たとえ「メニュー」に書いてあったとしても，値段はメニューの属性ではなく，飲み物の属性とするべきである（但し，複数の店を想定する場合には，値段は，その提供者（店）と提供物（飲み物）の間にある「提供」の属性となる）
5		操作も属性と類似した要素であり，所属するクラスの識別子に対して，推移的でない関数従属性を満足させるべきである．	「検索する」という操作は，データがある限りにおいて検索可能であるが，ユーザの誕生・死亡と検索は無関係である．従って，検索操作（メソッド）は，被検索データにつくべきである
6		関連とは関数従属性であり，「このクラスが決まれば，他方のクラスが決まるか否か」が関連を付ける必要性を判断する唯一の根拠である．関連につながる一方のクラスが主語であると思ってはならない	「商品」と「メーカー」の間には明らかに関数従属性があるが，「商品をメーカーに発注する」であり，何れも主語ではない
7		クラスの名称は，「無標」「有標」の観点から見た「無標」の名詞で表現する．	特定の企業向けのアプリケーションでは，その日本語仕様に現れる「会社」をクラスとして明示する必要はない
8		対象ビジネス中にインスタンス一個だけあって，当該ビジネスにおいて変化しない場合には，概念クラス図でそれをクラス化する必要はない	プリウスでは，1台1台を区別して表すクラスと，トヨタ社内の車種を表すクラスが設けられるべきである．しかし，チョロQでは，1台1台に管理番号を打つ必要はない
9		クラスには「種類」を表すクラスと，個別のインスタンスを表すクラスがある．	プリウスでは，1 台1 台を区別して表すクラスと，トヨタ社内の車種を表すクラスが設けられるべきである．しかし，チョロQ では，1 台1 台に管理番号を打つ必要はない
10	関連	動作動詞は，結果として，対象物の間に永続する関係性を含意することがある．この場合には，動詞が動作動詞であっても，その永続的関係性を関連等として，表現するべきである	「チョコレートを花子に送る」は，結果としてチョコレートは，花子の所有物として永続的関係性を含意する．英語の動作動詞では，しばしば，このような静的関係が含意されているためである
11		欧米人の抽象化とは，「属性値が同じものを集める」という意味がある．「一般的な大きな概念ともっと具体的な概念との関係」とだけ捉えてはならない欧米人の抽象化とは，「属性値が同じものを集める」という意味がある	「〜と見なす」，「〜 と定義する」はis-a 関係の可能性が高い．is-a 関係とは，「〜 と定義するための特別の関連」くらいに思った方が良い
12		「もの{こと{もの」パターンは，本表の項番6, 項番10 である．関連のいずれの側も主語ではない．動作動詞（行為）の結果，スタティックな関数従属性が生成される（多対多）．このため「関連」は必要である．但し，このままでは，(1) RDB に翻訳できない，(2) 関連に持たせるべき属性を書くべきクラスがない，と言った問題があり，関連クラスの追加が必要であることが多い．これが「もの－こと－もの」パターンである	「会社はAをBに発注する」では，主語「会社」は暗黙の了解である．一方，「Bに」は所格として必須である．AとBの間には関数従属性が生まれるから関連が生まれる．但し，発注に伴う「発注ID」「発注日」などを記録するために，第3正規形を構成する関連クラス「発注」が必要となる

第3章　存在従属分析によるドメインモデルの構築

3.1　緒言

今日，ビジネスアプリケーション開発では経営環境の急速な変化に対応するために空前のアジリティが求められている．同時に，多くの組織で開発しているシステムは，昨今のビッグデータブームからも明らかなように，よりデータ指向的になってきている [20]．長期的な傾向としては，ビジネスの深層部でより一層複雑化する問題に対してソフトウェアを適用する方向にあるとされる [21]．

そのため，ドメインモデルに複雑なアプリケーションデザインの基礎を求めるドメイン駆動設計 [22] がアジャイル開発の中で採用されるケースが増加している．ドメイン駆動設計でいうドメインモデルとは，アプリケーションの問題領域（ドメイン）の管理対象を端的に記述し，ドメインの専門家と開発技術者との意思疎通を促進し，かつ，ドメインの論理要件[1]を満たして，データベース設計のモデルへの変換もストレートフォワードに行えるようなモデルを指す [22]．

近年，ビジネスアプリケーションの世界では，ドメインモデルの構築にはオブジェクト指向アプローチ（Object Oriented Approach：以下，OOA）と UML 表記を採用し，その実装には Java のようなオブジェクト指向プログラミング言語（OOPL）と Oracle のような関係データベース管理システム（RDBMS）を用いた開発が一つの典型的なスタイルとなっている [23]．

しかしながら，OOA の発想だけでは，論理要件に対して頑健なドメインモデルを構築することは難しく，また UML のクラス図表記もそのままでは論理要件を表記し難いという問題がある．なぜならば，OOA ではどのようなクラスのインスタンスであっても，その識別子として属性とは別に単一属性のオブジェクト ID（OID）を仮定するためであり，クラス図には分かりきった OID を含めて主キーを明示的に表記しない慣習があるからである[2]．

[1]本章ではこの用語をドメインで捕捉，蓄積，管理すべき重要なデータの無矛盾性，整合性，一貫性に関する要求の意味で用いる．

[2]ある属性が主キーであることを明示的に表記するにはわざわざステレオタイプ≪ PK ≫などを付加する必要がある [24]．

このような識別子の取扱い方では，クラス間の結び付きの意味が曖昧になり，業務ドメインにおいて重要なデータの論理要件を満足にモデルに表現することはできない．結局，モデルに表現されない要件は正しく実装されないか，あるいは忘れ去られる運命にある．そのような理由から，かつての DOA の手法を見直す動きもある [20] が，直感的な分りやすさと収束の速いトップダウンの実体主導型（3.2.2 節で述べる）が身上の OOA はアジャイル開発に適しており，ここに急遽 DOA のようなデータ項目主導型の正規化理論[3] を持ち込むことは困難であると考えられる．

そこで，本章では，OOA とクラス図をベースに存在従属性に着目したドメインモデルの構築手法と表記法を提案する．この手法は，クラスの識別子とクラス間の関連をインスタンスにとって生まれながらの存在従属性と後天的な参照を峻別して見直すものである．提案手法を適用すれば正規化理論を表に出さずにドメイン構造の解明を容易にするため，結果として論理要件に頑健なドメインモデルを迅速かつ正確に構築できる．ただし，存在従属性の概念を紹介した論文 [25][26] やこの概念をモデリングに導入する提案 [27] は過去にも存在するため，本章ではこの概念をオブジェクトの属性や識別子の決定に用いることを，その表記と併せて提案する．

以下，3.2 節では，本提案に関係する概念の定義と関連研究の概要を述べる．3.3 節では，本提案の手法を説明する．3.4 節では，提案手法と正規化レベルの対応について議論する．3.5 節では，存在従属クラス図の性質について検討する．3.6 節では，存在従属性の UML クラス図での表記について議論する．3.7 節では，本提案手法を例題に適用した結果を評価する．3.8 節は本章の結言である．

3.2 概念の定義と関連研究

3.2.1 ドメインモデルとドメイン駆動設計

通常，ビジネスシステムは複数の問題領域から構成される．たとえば，在庫管理や生産管理などは典型的な問題領域である．この問題領域をドメインと呼ぶ．ビジネスシステム全体では大きすぎて分析対象としては適さないため，通常は全体をドメインに分けて個別に分析が行われる．

ドメイン分割された領域ごとにドメインモデルが作成される．このようなドメインモデルと呼ばれるモデルは以前からも用いられてきた．それらは，概念モデル，概念データモデル，ドメインのクラス図などとも呼ばれ，ドメインの静的なデータ構造を表し，それを操作可能な単位の集まりとして組織化するものである．当初は開発技術者が業務を理解するために用語辞書を補足する目的で作成されることが多かったが,「エリック・エ

[3] そもそも正規化理論で用いられる関数従属性はデータ項目間の関係であって，エンティティ間の関係ではない [25].

ヴァンスのドメイン駆動設計」[22] が好評を博して以来，ドメインモデルへの要求は高度化している．

ドメインモデルに対する情報は，問題記述やそのドメインの専門家の知識，実世界の一般知識より得られる．もしも開発者がその領域の専門家でなかったら，専門家から情報を得ながらモデルに対するチェックを繰り返さなければならない．このとき，ドメインモデルは，ドメインの専門家とソフトウェア開発の専門家とのコミュニケーションを促進するものでなければならない．したがって，ドメイン駆動設計におけるドメインモデルはドメインの専門家と開発の専門家の意思疎通のための共通語，すなわちユビキタス言語 [22] として機能することが求められるといえよう．

3.2.2 ドメインモデルの構築

従来からドメインモデルの構築には 2 つの方法が存在する．第 1 の方法は，データ項目主導型である．そのアプリケーションに関連するデータ項目を洗い出し，関数従属関係を満たすように属性の集合へと統合あるいは分割していく．第 2 の方法は，実体（エンティティ）主導型である．そのアプリケーションが対象とするドメインに存在する意味のある実体を見つけて，それを記述する [28].

OOA は，実体主導型の方法に分類される．そして，実体（クラス）と属性，全体と部分，クラスとそのインスタンスといった，幼児が最初に自分をとりまく世界を理解するときに獲得してしまうような概念に基礎をおいている [29] ため，非常に直感的に理解しやすいという優れた特徴を持っている．このことは，OOA の重要な資産であると同時に大きな負債でもあると著者は考える．なぜなら，このことが原因で多くの OOA の教科書や方法論が，このモデリングプロセスが極めて直感的で容易であり，あたかも説明するに値しないかの如くこのプロセスに関する説明を省略してきたため，初学者にとって実に迷いや混乱が多いアプローチともなっているからである．表 3.1 は DOA と OOA の特徴を著者が大局的に対比したものである．

3.2.3 オブジェクト指向と正規化理論

表 3.1 は DOA には正規化と呼ばれる集合論に裏打ちされた設計理論が存在することを示す．しかしながら，OOA にはそれに相当するような方法は未だ存在しない．それでは，OOA にも正規化理論を導入すればよいと考えられるが，前述の通り OOA の識別子の考え方は正規化理論に馴染まない．OOA における識別子の考え方は，先にオブジェクトを識別し，すべてのオブジェクトには生まれながらにして，OID と呼ばれる，単独の識別子が自動的に付与されるとされる（以下，OID 方式と表現する）．一方，DOA においては，アウトプットに求められる属性項目を，正規化理論を用いて，結合従属性を失わな

表 3.1: DOA と OOA の大局的な対比

項目	DOA	OOA
着眼点	属性（項目）主導型	実体主導型
アプローチ	ボトムアップ中心	トップダウン中心
モデリングプロセス	正規化	直感方式
識別子	複合主キー前提	オブジェクトIDと呼ばれる単独識別子を常に仮定

いように分解あるいは統合する．したがって，DOA における識別子の考え方には，複合主キーが自然かつ頻繁に登場する．たとえば，第 1 正規形から第 2 正規形へ変換する際には，識別子に部分関数従属しているデータ項目を分離するが，そもそも部分関数従属という概念は，複合主キーが前提である．OOA の識別子の考え方と，DOA の識別子の考え方は根本的に異なる．

ここで，ユーザの要求を充足するアプリケーションが提供できるのであれば，いずれの識別子の考え方に基づいていてもよいが，オブジェクトの永続化の実現手段として，90% 以上 RDBMS が用いられている現状 [20] においては，事情が異なる．

OOA の識別子の発想では，整合性要件を充足するアプリケーションの構築が難しい．なぜならば，OOA の OID 方式には，複合識別子は登場しない．クラスがどのようなタイプのものであれ，すべてのインスタンスは単一属性の OID で識別されることになっている．このような識別子の付与を行った場合，たとえ優れた RDBMS を永続化層で使用していたとしても，整合性制約をアプリケーション側で実装しなければならないモデルになる．

さらに，困ったことに，前者のアプローチで作成されたモデルと後者のアプローチで作成されたモデルは，一見非常によく似ている．そして，クラス図には伝統的に主キーが表記されないことが事態をより深刻にしている．データの整合性要件（業務の論理要件）の観点からは，OOA のモデルは脆弱になりがちである．このことは，ドメインモデルの作成とデータモデルの再作成という重複する努力を排除できない原因の一つとなっている．

3.2.4 存在従属性

存在従属性の概念は P．チェンが用いている [30][25]．あるオブジェクトが，別のオブジェクトの先立つ存在を前提として存在し得るとき，前者のオブジェクトは後者のオブジェ

クトに存在従属するという．そして前者のオブジェクトを従属クラス（dependent class）のオブジェクト[4]，後者のオブジェクトを独立クラス（master class）のオブジェクト[5]と呼ぶ [27]．この存在従属という概念を用いることによって，たとえば，ある区間は，ある２つの地点の間に存在従属する．月々のローンの返済は，過去の購買という事象に存在従属する，などと表現できる．UML の提案者たちの OOA の文献ではわずかに [31] だけが存在従属性について言及している．

3.2.5　存在従属性と属性

存在従属性は，オブジェクト間だけの関係ではない．オブジェクトと属性の間にも存在従属性が認められる．すなわち，属性とは，あるオブジェクトが生まれるとともにそのオブジェクトの値として生まれ，そのオブジェクトがこの世に存在する限り，そのオブジェクトに付随し，そのオブジェクトがこの世から消えるとともに当該属性も消えるデータ項目のことである．したがって，属性群はオブジェクトに存在従属するといえる．言い換えれば,「オブジェクトに存在従属するデータ項目こそが属性」という観点でそれらを定義すると，属性はオブジェクトに完全関数従属するとともに，候補キーではない属性から候補キーを構成する属性への関数従属性をも排除することができる．クラスが持つべき性質で必要なのは,「すべての事実は，キーだけに関係する事実である [32]」を満たすため，これだけでも少なくともボイス・コッド正規形が達成される（このことに関しては 3.4 節で確認する）．これまで見てきたように存在従属性に着目してオブジェクト指向分析を実施するとボイス・コッド正規形以上の頑健さを備えたドメインモデルが構築できる．3.3 節では OOA の単純さ，直感的な分りやすさを犠牲にすることなしに，論理要件に対して頑健なドメインモデルを構築する手法とその表記法を提案する．

3.3　エンティティの存在従属分析

3.3.1　ガイドライン

提案の核心は極めてシンプルである．具体的には存在従属性の概念を導入したドメインモデリングのガイドラインを次のように定める．

イ) あるデータ項目をあるクラスの属性とするには，その値がそのクラスのインスタンスに存在従属する場合に限る．

[4]弱実体（weak entity）とも呼ばれる．
[5]強実体（strong entity）もしくは，正実体（regular entity）とも呼ばれる．

ロ) そのインスタンスが，独立して存在可能なクラスの ID には「必ず」単一属性の ID を与える．

ハ) そのインスタンスが，独立して存在できない（存在従属な）クラスの ID には,「決して」単一属性の ID は与えず,「必ず」その存在根拠（前提）となるクラスの ID を含む ID を与える．結果，存在従属なクラスの識別子は複合主キーとなる．

ニ) そのインスタンスが,（リソースではなく）イベント[6] の場合は，上記のルールに加えて，そのクラスの ID に「必ず」時点をあらわす時刻または版（バージョン）のシリアル値を含める．

ホ) 存在従属以外のクラス間の関係は，次の汎化関係と単なる参照関係の 2 種類のみとする．

i. 汎化関係の場合：サブクラスの ID は「必ず」スーパークラスと同じ ID を共有する．たとえば,「プレミアム会員」のスーパークラスが「会員」クラスで，その ID が「会員 ID」であった場合は,「プレミアム会員」クラスの ID も「会員 ID」となる．

ii. 単なる参照関係の場合：参照元のインスタンスと参照先のインスタンスのそれぞれのライフサイクルは独立している．このような場合は，表記では参照元のクラスから参照先のクラスへの依存関係を定義し，RDB では参照元は参照先の ID を外部キーとして保持するものとする．

以下では，このガイドラインを用いた手法について説明する．例題として，専門学校などで，１週間の時間割の策定を支援するアプリケーションを考える．

このアプリケーションには，さまざまな制約がある．制約とは,「実習科目に座学教室が割り当てられないこと」など，必ず真にならなければならない条件である [32]．ドメインの専門家（たとえば学校の時間割策定担当者）は，このような成立して当然の条件群は，通常わざわざ口にしないと考えられるが，それでも制約は論理要件として厳然と存在するので，ドメインモデルではそれらを表現しなければならない．ある学級[7] の月曜日の１時限目の授業を考える．学級と曜日と時限が決まれば，授業が決まる．おのおのの授業には，講師，教室，そして教科が必要である．しかし，教科によって担当できる講師や教室の要件（座学教室か実習教室か）に制約がある．また，講師も非常勤の場合は，出講（登壇）可能な曜日と時間帯に制約があるのが普通である．さらに，教科もその学級で１週間で実施すべき教科が過不足なく網羅されなければならない．

こうなってくると OOA の発想だけでは，論理要件に頑健なドメインモデルを構築することはもはや難しいはずである．けれども，OOA の発想に存在従属の概念を追加するだ

[6] 佐藤 [33] は，エンティティを時点が帰属するイベントと帰属しないリソースに類別している．
[7] 「クラス」と呼びたいが，OOA の「クラス」との混乱を避けるため「学級」と呼んでいる．

けで，この問題は劇的に考えやすくなる．

以下では，存在従属の概念を用いた分析（存在従属分析と命名する）手順とモデルの表記について説明する．

3.3.2 分析手順とモデルの表記

存在従属分析は図 3.1 に示す Step 1〜Step 4 の 4 つの手順で構成される．以下では各ステップにおいて行うべきことを説明する．

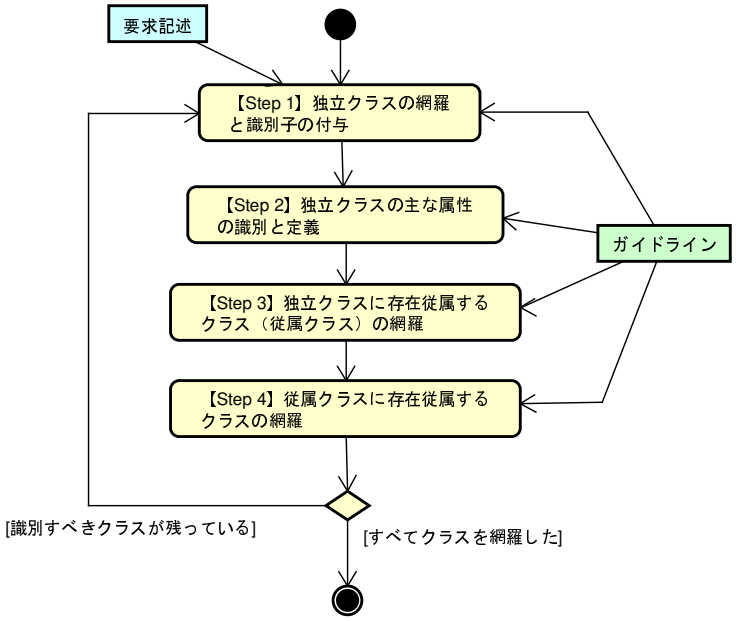

図 3.1: 存在従属分析手順

【Step 1】当該ドメインにおける独立クラスを網羅し，それぞれに識別子を付与する

まずは独立クラスを識別する．独立クラスとは，当該ドメインにおいてそのインスタンスが単独で存在できるクラスである．この問題の場合，学級，教科，講師，教室，曜日，時限が独立クラスである．ここで或いは，学級は，その学級が置かれる学校に存在従属するような従属クラスではないかと考える方もおられるかも知れないが，ドメインがその学校一校に限られるのであれば学校そのものをモデル化する必要はない．独立クラスには，前述のガイドライン（ロ）に従って，その識別子として単独属性の主キーを付与しなければならない．例題では，学級 ID，教科 ID，講師 ID，教室 ID，曜日 ID，時限 ID をそれぞれ対応する独立クラスの識別子として付与する．この付与法は OID 方式と同様である．図 3.2 は Step1 実施後のドメインモデルである．図 3.2 では，モデルを簡潔に

保つために主キー属性を示す目的でステレオタイプは用いず，属性名の後に「ID」を追加してそれを示すことにした．

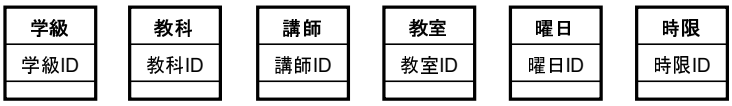

図 3.2: 例題ドメインの独立クラス群

【Step 2】独立クラスのインスタンスに存在従属するデータ項目（すなわち属性）を記述する

つぎに，独立クラスのインスタンスに存在従属するデータ項目（すなわち属性）を記述する．データ項目はすべて網羅する必要はなく，業務要求に照らして主要なものを 2〜3 個記述するだけでよい（クラスが識別された後で充実させればよい）．こうすることによって，そのクラスの存在意義を具体的に把握しやすくなる．ここで大切なのは，上述のガイドライン（イ）にしたがって，データ項目は当該インスタンスに存在従属するものだけに限定することである．データ項目のインスタンスに対する存在従属性を保つことにより，そのデータ項目は文字通り「属性」となるため，正規化理論でいうボイス・コッドの正規形が自然と満たされることになる．

例題では，たとえば，教室クラスに，座実区分が置かれている．その教室が座学教室なのか，実習教室なのかは教室に存在従属するデータ項目と考えられるからである．図 3.3 は Step 2 実施後のドメインモデルである．

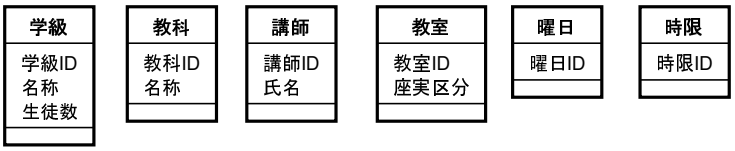

図 3.3: 属性が追加された例題ドメインの独立クラス群

【Step 3】当該ドメインの独立クラスに存在従属するクラスを網羅する

今度は，従属クラスについて考えていく．従属クラスとは，当該ドメインにおいて，そのクラスのインスタンスが単独で存在することができないクラスであり，必ず別のインスタンスの先立つ存在（生成）が前提である．この問題の場合，講師の「担当できる教科」は，講師と教科に存在従属する，「講師の出講できる時間帯」は，講師と曜日と時限に存在従属する，などと考えながら従属クラスを発見していくことができる．

図 3.4 は Step 3 実施後の例題のドメインモデルである．図 3.4 では，前述のガイドライン（ハ）にしたがって，各クラスの識別子としてその存在根拠（前提）となるクラスの ID を含む ID を与えている．その結果，従属クラスの識別子は複合キーとなっている．たとえば，担当可能クラスの識別子は，教科の ID と講師の ID の連結した複合主キーである．こうすることによって，担当可能クラスのインスタンスは必然的に，実存する講師と実存する教科の組み合わせに限定される．

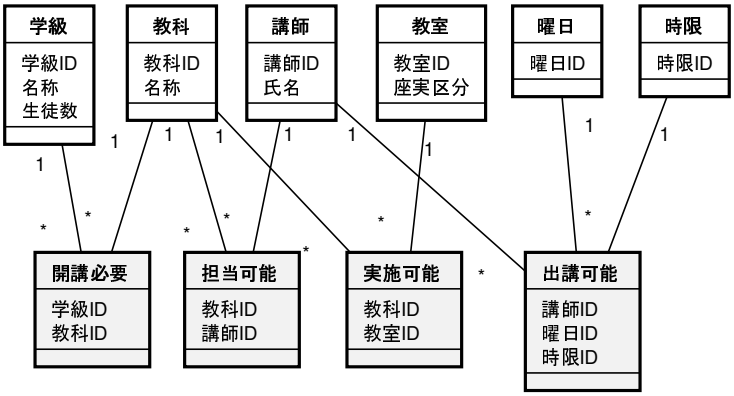

図 3.4: Step 3 実施後の例題のドメインモデル

ここで，表記法についても検討したい．図 3.4 では，従属クラスを淡い灰色で表記し，おのおのの独立クラスとの間に関連を定義しているが，その関連が存在従属であることをモデルの読み手に強く印象づけたい．なぜならば，関連の意味が存在従属であることが正しく伝われば，従属クラスの複合識別子をわざわざ表記する必要がなくなり，モデルが簡潔になるからである．同時に，従属クラスからみた独立クラスの多重度は必ず 1，従属側は多になっているため，多重度もわざわざ表記する必要はなくなる．そこで，図 3.4 のモデルは図 3.5 または図 3.6 のように表記できることも併せて提案する．図 3.6 では，上述のガイドライン（ロ）が徹底されるのであれば，独立クラスの識別子でさえも省略可能であることを示唆している．

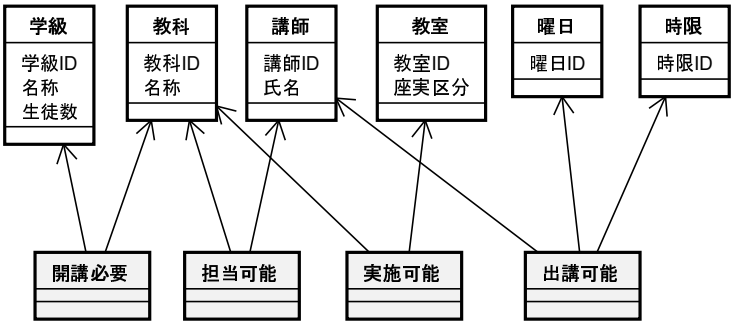

図 3.5: 図 3.4 の提案記法による表記（ID 明記）

ここで，図 3.5 と図 3.6 では関連の種類が存在従属性であることを明示的に表現するために誘導可能性を使用している．ドメインモデルは，ユースケース実現のモデリングを

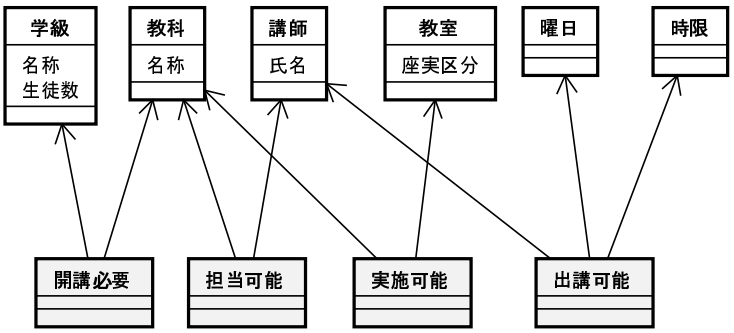

図 3.6: 図 3.4 の提案記法による表記（ID 表記省略）

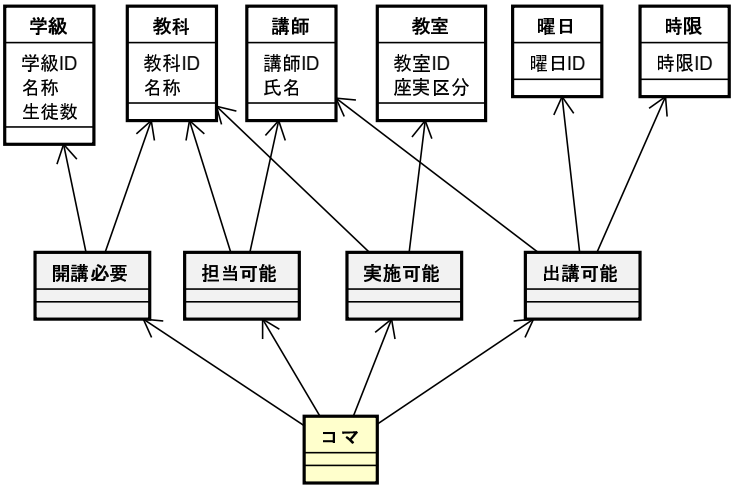

図 3.7: 例題ドメインの Step 4 実施後のモデル

実施する以前のモデルであるため，誘導可能性がモデルに登場することはないため，このモデル要素を流用することにした．また，誘導可能性を示す関連（矢線）は，3.1 節で述べた通り RDB への実装を前提としている以上，妥当であると考える．ただし，この表記法の提案に至る経緯については 3.7 節で議論する．

【Step 4】当該ドメインにおける従属クラスに存在従属するクラスを識別する

さらに，当該ドメインにおける従属クラスに存在従属するクラスを識別する．例題では，時間割のそれぞれのコマがそれに該当する．図 3.7 は「コマ」クラスをドメインモデルに追加したものである．図 3.7 のモデルでは，時間割の各コマが満たすべき論理要件が正確に反映されている．すなわち，時間割コマのすべてのインスタンスは，ある学級において開講が必要な教科について，その教科が担当可能でかつその時間帯に出講可能な講師と，その教科を実施可能な教室の先立つ存在を前提として存在できることを示して

いる[8]．これは，時間割策定支援アプリケーションの論理要件そのものである上，このモデルは第 4，第 5 正規形が満たされていると期待される．提案手法と正規化レベルの対応については 3.4 節で論じる．

3.4　提案手法と正規化レベルの対応

表 3.2 は提案手法が正規化にどのように寄与しているかについてまとめたものである．表中の正規化レベルごとの概要については渡辺 [34] を参照した．ただし，存在従属クラス図における正規化レベルは，当該クラス図を関係モデルに変換した後の状態にて議論するものとする．そして，存在従属クラス図を関係モデルに変換する手順は次の通りとする：

(1) 図上のすべてのクラスと 1 対 1 で関係変数を用意し，各クラスを関係変数に対応付ける．もちろん，クラス名はそのまま関係変数名として採用する

(2) 図上のすべてのクラスのすべての属性を，対応する関係変数の属性に対応付ける．このとき，その属性が保持し得る値が，スカラ値でない場合は，あらたに関係変数を用意して，属性群を当該関係変数に移動するとともに，当該関係変数から元の関係変数に結合従属性のための外部キーを定義する．なお，便宜上，数値，文字列，日付および時刻の値は，実装上では構造を持っていたとしてもスカラ値であるとみなす

(3) 図上のすべてのクラス間の関係（属性，存在従属関係，単なる参照関係，汎化関係）について 3.3.1 節に示したガイドラインに従って，参照される関係変数の主キー属性と参照する側の関係変数の外部キー属性に対応付ける

提案手法のガイドラインを厳格に適用するならば，理論的には 3.2.5 節で述べた通り，ドメインに登場する概念を構成するデータ項目群は，ある概念に存在従属すると認められるものだけがその属性となるため，おのおのの属性値がそのインスタンスに完全関数従属するとともに，部分関数従属や推移関数従属するようなテータ項目は最初から属性にはならない．同時に，たとえば製造番号（シリアル番号）のように製品個体の属性とすべきデータ項目が製品種類の属性とされたり，逆に型番のような製品種類の属性とすべきデータ項目が製品個体の属性にされてしまうような誤りも回避される．これにより，自明でない多値従属性も排除される．

[8]結果，論理的に「コマ」クラスの主キーは学級 ID，教科 ID，講師 ID，教室 ID，曜日 ID，時限 ID からなる複合主キーである．

表 3.2: 提案手法の正規化レベルへの寄与

正規化レベル	概要	提案手法における対応
第1	非キー属性のうち繰り返しデータ項目などの集合要素を分離	対応なし．しかし，繰り返す属性は，発見に労を要しないため，発見次第直ちに別の実体として分離すれば問題なし
第2	非キー属性のうち，主キーの一部だけに関数従属（部分関数従属）するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
第3	非キー属性のうち，主キーから推移的に関数従属するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
ボイス・コッド	非キー属性のうち，候補キーに対して部分あるいは推移関数従属するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
第4	関係から自明でない多値従属性のある関係を取り出して分離	このような関係は従属クラスとして識別されるため，当初から分離される
第5	関係のうち，既約でない属性を 取り出して分離	このような関係は従属クラスとして識別されるため，当初から分離されるはずであるが，どの程度まで分割するかは業務要件に強く依存する問題であるため，提案手法ではさらに分離すべき属性が残存しないことを保証できない

3.4.1 提案手法とボイス・コッドの正規形

詳しく見てみよう．複数の独立クラスに存在従属するクラスの場合，まず，A，B 2つのクラスに存在従属するようなクラス C が識別された場合，C の識別子は，3.3.1 節のガイドラインに従えば，基本的に A の識別子（これを a とする）と B の識別子（これを b とする）の複合識別子 $\{a, b\}$ である[9] が，C の属性には，この複合識別子に存在従属するデータ項目のみが含まれるため，これを仮に c とした場合，$\{a, b\} \to c$ のたった 1 種類の関数従属性は成立するが，その他の関数従属性は成立しない．

まず，$a \to c$ の関数従属性は成立しない．なぜならば，$a \to c$ が成立する場合は 3.3.1 節のガイドライン（イ）によって，データ項目 c は C の属性ではなく先に A または，A だけに存在従属するクラス，あるいは，それらいずれかの参照先のクラス属性にされているべきだからである．同じ理由で，$b \to c$，$a \to b$，$b \to a$，$c \to a$，$c \to b$，といった関数従属性も成立しない．

つぎに $\{a, c\} \to b$ も成立しない．なぜならば，このような関数従属性の成立は b は B の属性という当初の前提と矛盾するためである．同じ理由で，$\{b, c\} \to a$ といった関数従属性も成立しない．

以上から，$\{a, b\} \to c$ のたった 1 種類の関数従属性だけが成立することが確認されたため，提案手法はボイス・コッドの正規形を満たすことが確認された．

[9]実際には，クラス C の識別子に時点属性が加わる場合があるが，それは関数従属性を限定する方向にしか作用しないため，ここでの議論では無視できる．

3.4.2 提案手法と第 4 正規形

つぎに，第 4 正規形を検証する．そのためには，3 つ以上のクラスに存在従属するようなクラスについて，その識別子を構成する属性間に，自明でない多値従属性が存在しないことを確認すればよい．もちろん，それ以外のクラスについては，前節で述べたボイス・コッドの正規形の確認が完了していることが前提である．

まず，3 つのクラスに存在従属するようなクラスの識別子について検証する．たとえば，A，B，C 3 つのクラスに存在従属するようなクラス D が識別された場合，D の識別子は，3.3.1 節のガイドラインに従えば，基本的に $\{a, b, c\}$ であり?? [10] ，D の属性には，この複合識別子に存在従属するデータ項目のみが含まれるため，これを仮に d とした場合，$\{a, b, c\} \rightarrow d$ のたった 1 種類の関数従属性だけが成立することを確認する必要がある．検証の流れは，上記 3.4.1 節の手順と同様であるため，追加で検証すべきは，D の識別子 $\{a, b, c\}$ 内部において，$a \twoheadrightarrow b$，かつ $a \twoheadrightarrow c$（ただし，b と c は独立）の多値従属性が成立しないことである．ここで，もしも，$a \twoheadrightarrow b$ が成立するならば，b は B の正しい属性ではなく，A または，A だけに存在従属するクラス，あるいは，それらいずれかの参照先のクラス属性であったということになる．したがって，$a \twoheadrightarrow b$ は成立しない（$a \twoheadrightarrow c$ についても同様）．

あとは，実例は少ないと思われるが 4 つ以上のクラスに存在従属するようなクラスが識別された場合についても同様の手順で確認していくことができる．C.Date[32] によれば，少なくとも第 4 正規形に関しては，ちょうど 2 つの射影への無損失分解を繰り返せば十分である, とされる．

以上から，提案手法は第 4 正規形を満たすことが確認された．

3.4.3 提案手法と第 5 正規形

渡辺は，第 5 正規形は努力目標ではなく，このレベルでないデータモデルは使い物にならないと述べている [34] ため，第 5 正規形に関しても検討する必要がある．しかしながら第 5 正規形については，提案手法を適用しても，さらなる結合従属性を含んでいるクラスが残っている可能性は否定できない．なぜなら，この問題は正規化だけではなく業務要件に強く依存する問題でもある [11] からである．さらに分解可能なクラスを発見し，第 5 正規形にもっていくためガイドラインについては今後の検討課題としたい．

[10] 実際には，クラス D の識別子に時点属性が加わる場合があるが，それは関数従属性を限定する方向にしか作用しないため，ここでの議論では無視できる．

[11] たとえば，製品を部品レベルにまで分解して管理すべきかどうかは，販売ドメインと製造ドメインで異なる．

3.5 存在従属クラス図の性質

前節では，業務ドメインで扱うべき実体について，周到な存在従属性のチェックを行えば，正規化理論を陽に適用することなしに，第 4 正規形以上の正規化レベルを有するクラス図が得られることを確認したが，ここでは，それを踏まえて存在従属関連の性質について考察する．3.2.4 節で述べた存在従属性の定義に照らして，存在従属関連は次の性質を持っている．これらの性質は，存在従属クラス図を入力として，以降の章で述べるような，さまざまな設計成果物を出力できる可能性をもたらす．

性質 3.5.1（存在に関する性質） クラス A からクラス B に向けて存在従属関連が定義されている場合，クラス A のインスタンスが存在できるためには，少なくとも，クラス B のインスタンスが先に存在しているか，同時に生成される必要がある．

性質 3.5.1 の証明 存在従属関連では，クラス B のインスタンスが存在しないと，クラス A のインスタンスは生成できないため，クラス B のインスタンスが先に存在しているか，同時に生成される必要がある． □

性質 3.5.2（削除に関する性質） クラス A からクラス B に向けて存在従属関連が定義されている場合，クラス B のインスタンスは，それに存在従属している A のインスタンスが存在している間は，削除することができない．

性質 3.5.2 の証明 存在従属関連では，クラス B のインスタンスが存在しないと，クラス A のインスタンスは存在できないため，クラス A のインスタンスが存在している間に，その従属先のクラス B のインスタンスを削除した場合は，性質 3.5.1 に矛盾が生じる． □

性質 3.5.3（参照の性質） クラス A からクラス B に向けて存在従属関連が定義されている場合，クラス A のインスタンスにクラス B のインスタンスへの参照を持たせれば，その参照の値はスカラ値であり，かつ非 NULL の定数である．

参照の性質 3.5.3 の証明 存在従属関連では，クラス B のインスタンスが存在しないと，クラス A のインスタンスは生成できないため，クラス A のインスタンスにクラス B のインスタンスへの参照を持たせれば，その値は必ず 1 つだけ存在するため，スカラかつ非 NULL であり，クラス A のインスタンス生成後は，クラス B の従属先のインスタンスへの参照の値は，NULL 値にしたり，クラス B の別のインスタンスへの参照の値に書き換えできないため定数である． □

性質 3.5.4（非循環の性質） クラスレベルで存在従属関連のみで結ばれたクラス構造から，インスタンスレベルで閉路（サイクル）のあるオブジェクト構造を構築できない．

性質 3.5.4 の証明 インスタンスレベルでループになるような存在従属リンクを作ろうとすると，ループ先にある従属先のインスタンスが未存在の状態が発生する．これは，存

在従属関連の性質 3.5.1 に矛盾する．たとえば，x は y に存在従属し，y は x に存在従属するようなリンクは生成不可能である．同様に，x は y に存在従属し，y は z に存在従属し，z は x に存在従属するようなリンクも生成不可能である． □

3.6 表記法についての議論

3.3 節では，モデル化のステップとともにクラス図上で存在従属性を明確に表現するために誘導可能性の表記を導入する提案を行った．しかしながら，UML のモデル要素を本来の意味のまま用いることによって，存在従属関係を表現できるのであれば，それに越したことはない．本節ではその可能性について，若干の検討を加えておく [12]．

3.6.1 存在従属性をコンポジションで表記する

まず，存在従属性の表現のためにコンポジションを用いることを検討する．コンポジションは，たとえば，注文クラスと注文明細クラスの関係を表現する場合によく用いられる（図 3.8）．この例では確かに，注文明細は注文に存在従属するし，注文が全体側で，注文明細は他の注文と共有できない部分側のインスタンスである．このような見出し－明細パターンに見られる存在従属性を表現する場合はコンポジションが相応しい．

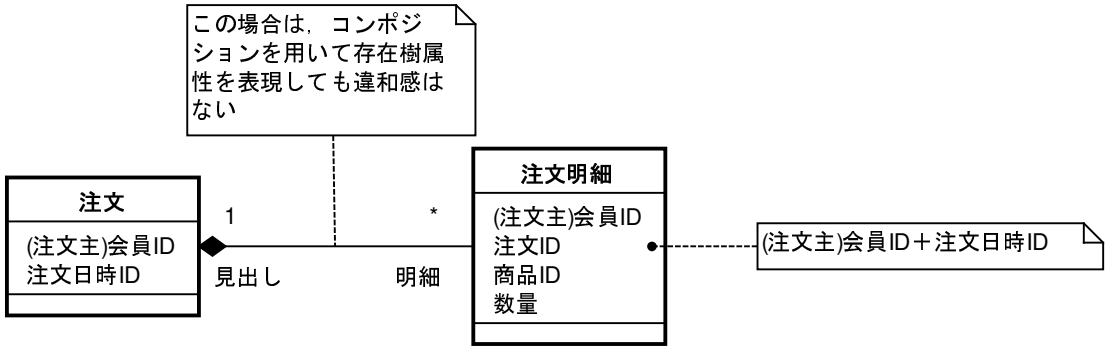

図 3.8: コンポジションを用いた存在従属性の表記（適切な例）

しかしながら，別の例，たとえば，オークションサイトの入札は，あきらかに出品に存在従属する（存在しない出品に対する入札は不可能であるため）が，この関係を図 3.9 のように，コンポジションを用いて表記した場合は，入札は出品の部分であるとは認めにくいため違和感が残る．コンポジションは，全体と部分の関係（しかも全体側にとって部分側を共有できない）という意味 [35] の方が強調されてしまうように感じられる．したがって，コンポジションを存在従属の表記に使用する方法は採用できないと判断した．

[12]ただし，ここではステレオタイプについては，多用するとモデルが煩雑になるため検討の対象から外している．

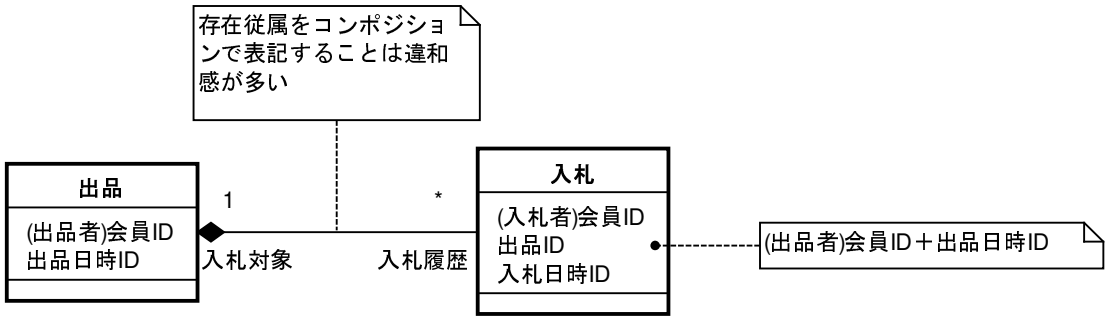

図 3.9: コンポジションを用いた存在従属性の表記（不適切な例）

3.6.2 存在従属性を関連クラスで表記する

つぎに，存在従属性を関連クラスにて表記することを検討する．2 つの独立クラスの間に存在従属する概念を関連クラスとして表記することで存在従属性がストレートに表現できれば好都合である．しかしながら，この方法にも問題がある．なぜならば，関連クラスは関連そのものである [35] ため，関連クラスには追加で独自の識別子を持たせることはできないという制約が存在するからである．この制約は，関連のインスタンスであるリンクは，自分自身の識別子を持てないため，その両端に接続されるインスタンスの組だけで識別されねばならないことからも理解される．すなわち，リンクの両端のインスタンスを 1 つずつ選んだ場合に，その間のリンクが 2 本以上存在するようなケースでは，それらを関連または関連クラスとしてモデル化することは厳密にいえば UML の文法に違反することになる．

例として,「会員がホテルを予約する」ドメインを考える．予約を会員とホテルの間に存在従属するクラスとしてモデル化することは自然であるとしても，その状況を関連クラスとして表記することはこの例ではできない．この例をオブジェクト図で表現すれは，図 3.10 のようになる．図 3.10 は，リンクの両端のインスタンスを 1 つずつ定めても，それらの間のリンクが 2 本以上存在するケースがあることを示している．このような場合，予約のインスタンスの識別を会員の識別子とホテルの識別子の複合キーだけで行うことはできないから，予約のインスタンスには日付などの追加の識別子が必要となる．したがって，予約は関連クラスとしてではなく，はじめから普通のクラスとしてモデル化しなければならない（図 3.11 の左側のクラス図は厳密には誤り）．このような例があるため，2 つの独立クラスの間に存在従属するクラスを関連クラスとして表記する方法も採用すべきではないと判断した．

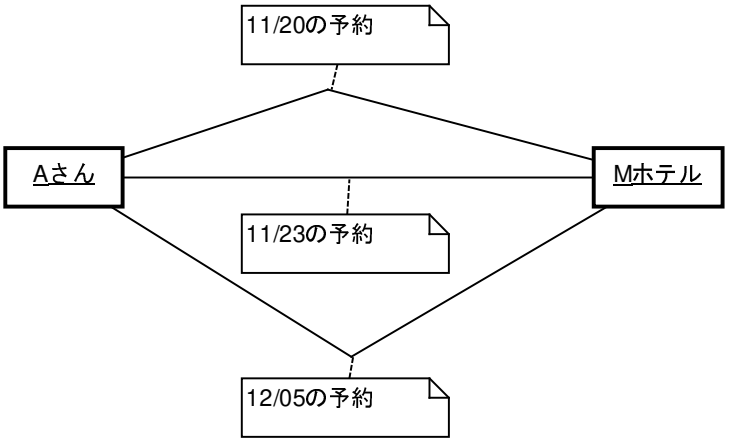

図 3.10: ホテル予約ドメインのオブジェクト図

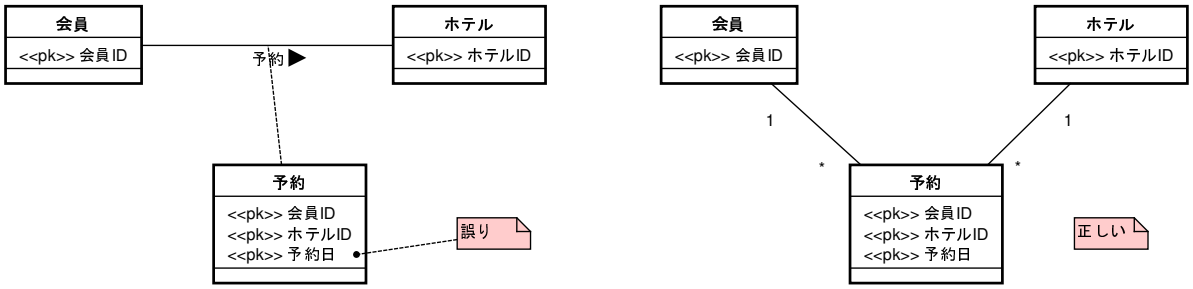

図 3.11: 関連クラスを用いた存在従属性の表記（不適切なケースがある）

3.6.3　存在従属性を多重度 1 の強調によって表記する

さらに，多重度がきっかり 1 であることを明確に示すことによって，存在従属性を表現することを検討する．たとえば，多重度が 1 以上 1 以下であることを強調すべく「1..1」のような多重度の表記を行うことが考えられる．しかし，結論から言えば，あるクラスから見た別のクラスの多重度が厳密に 1 であるからといって，前者のクラスは後者のクラスに存在従属しているとは限らない場合がある．図 3.12 はそのような場合の例で，このモデルは，従業員が部署に存在従属することを表しているのではなく,「従業員たるものは常時どこか 1 つの部署に所属しなければならない」という業務ルールを表わしている．また,「1..1」のような表記は，モデリングツールによっては受け付けられない．したがって，この方法も存在従属性の表現としては採用し難いと判断した．以上のような検討経緯から，クラス図上で存在従属性を明示的に表現するために誘導可能性を用いることに辿り着いた．

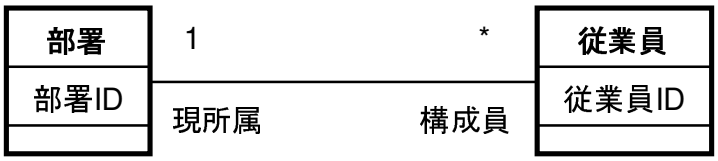

図 3.12: 存在従属ではない関連（参照関係）の例

3.7 データモデルによる検証

前節までで，存在従属性の概念を用いたドメインモデルの作成について，やや極端な例題を用いてそのガイドラインと手順，およびモデルの表記について説明した．

独立クラスと従属クラスを意識することで，ドメインの理解が促進される上，その論理要件を満たす理想的な識別子についての考慮も無意識に完了できることが明らかとなった．本節では，次の例題 3.7.1 を用いて提案手法を検証する．

3.7.1 例題 3.7.1 の問題記述

A社では，清掃当番表を作成して運用することにした．従業員３人ないし４人で清掃チームを編成し，清掃時間帯は毎営業日（月〜金）の終業時刻前 10 分間と決めた．また，清掃エリアを分割し，曜日ごとにチームが清掃するエリアのローテションを行う．そして，清掃作業の完了ごとに，清掃を実施した事実の有無を記録することにした．表 3.3 は清掃当番表の例である．

表 3.3: 例題 3.7.1 の清掃当番表

曜日 / チーム	月	火	水	木	金
A	事務所	会議室	研究室	廊下	事務所
B	廊下	事務所	会議室	研究室	廊下
C	研究室	廊下	事務所	会議室	研究室
D	会議室	研究室	廊下	事務所	会議室

3.7.2 ドメインモデルの構築と実験

例題 3.7.1 について，提案手法を適用してドメインモデルを作成した．清掃当番表の割当をバージョン管理するために，ガイドラインの（ニ）と，割当からエリアを参照するためにガイドラインの（ホ）-ii. を適用した．図 3.13 は，成果物を UML 表記に忠実に表記したものである．また，図 3.14 は，同じ成果物を提案の表記法で表記したものである．

これらを比較すれば，ドメインの専門家と開発技術者の対話を促進するのは図 3.14 のほうであると考えられる．

さらに，図 3.15 は，図 3.14 のドメインモデルの頑健性を検証するために作成したデータモデルの例である．ドメインモデルから直接 MySQL を使って実装し，試しにいくつかの具体値を用いてデータの追加，更新，削除を行ってみた．

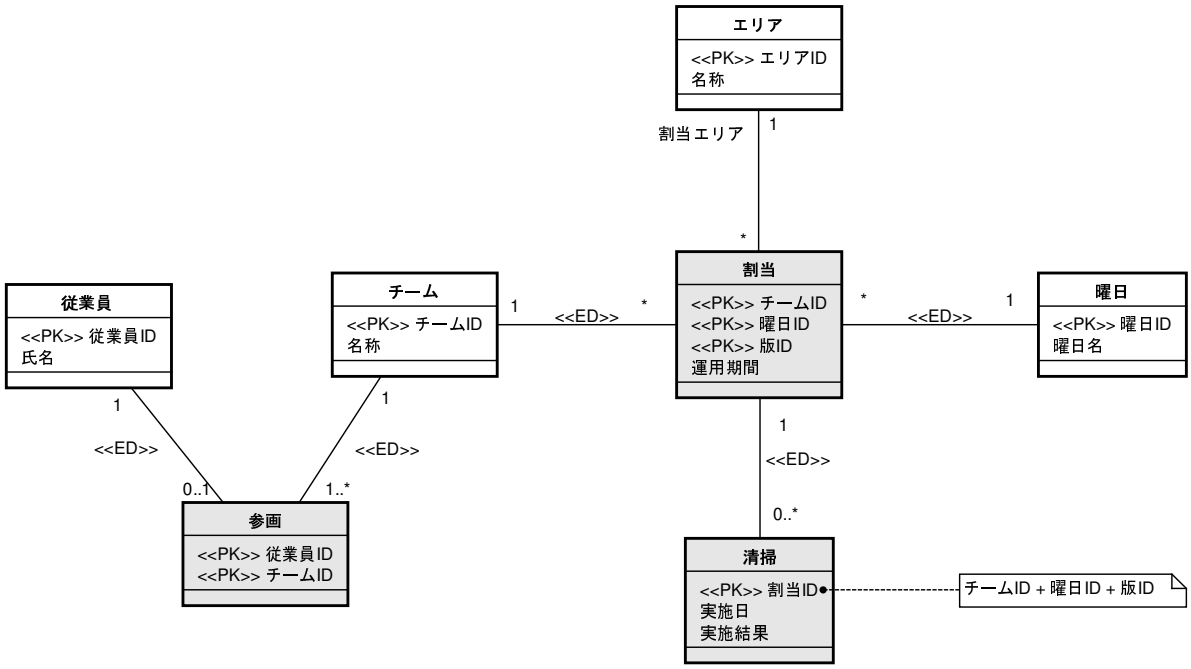

図 3.13: 例題 3.7.1 のモデル（クラス図の表記法に忠実な表記）

3.7.3 試行結果

試行の結果，このモデルは，次の当り前の業務要件を愚直にクリアした．

1. 割り当ての登録時には同じチーム，同じ曜日に異なったエリアの割当て登録ができないこと
2. 存在しない曜日や存在しないチームに対する割当て登録ができないこと
3. 割当てが存在するのに，チームや曜日が不用意に削除されないこと
4. すでに存在する割当ての，チームや曜日の変更を許さないこと
5. 将来，割当てが変更されたとしても，過去の実績に影響を与えないこと

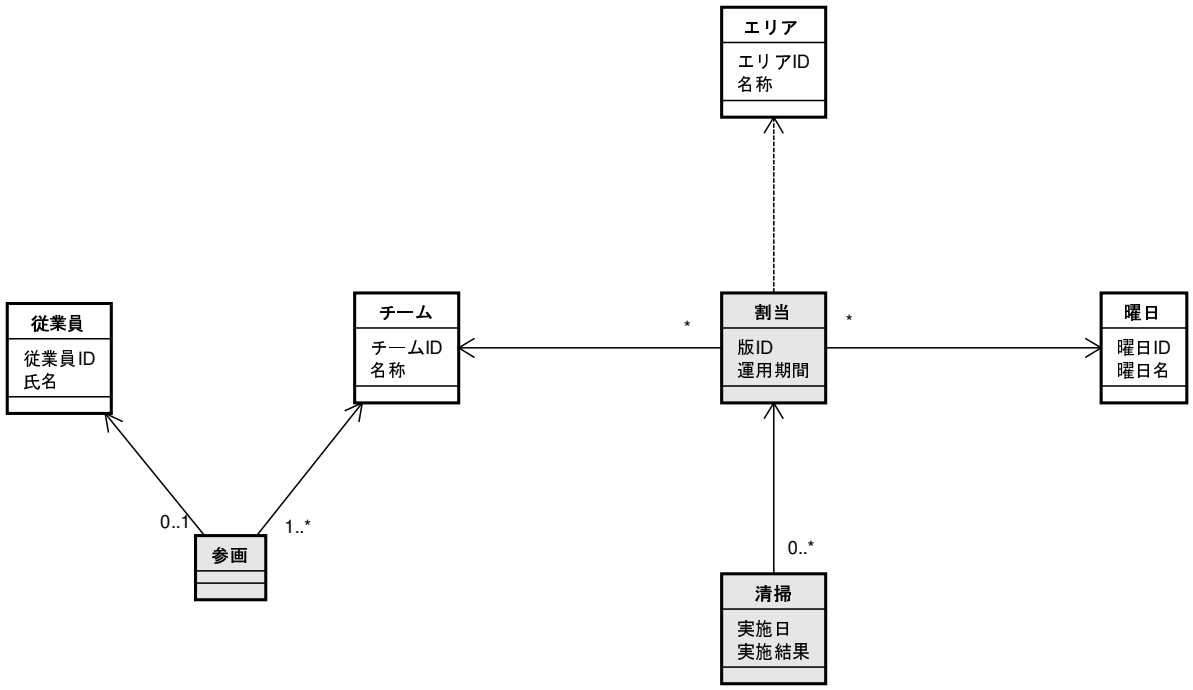

図 3.14: 例題 3.7.1 のモデル（提案の表記法による表記）

3.8 結言

本章では，ドメインモデル作成のために存在従属性に着目したモデリング手法と表記法を提案した．ヤコブソン [36] は，モデル化技法は，利用するのに簡単であり，少ない概念で構成され，簡単に学べて，強力なものでなければならないと主張しているが，提案手法はそのような状態を目指した．

ドメインモデルがドメインの専門家と開発の技術者が共通かつ厳密に意思疎通可能なユビキタス言語として機能するためには，そのガイドラインにおいても表記においてもノイズを排除した究極のシンプルさと，論理要件に対する頑健さを兼ね備えることが望まれる．

存在従属性の概念とそれに関わる表記もすでに存在しているが，存在従属性をオブジェクト指向アプローチにおいて正規化理論に相当するものと位置づけたこと，そして表記を世界標準の UML のクラス図の上で提案したところに本提案のオリジナリティがある．存在従属性の概念は理解しやすくドメイン中で見極めやすいため，提案手法により，第 4 正規形以上の正規化レベルを持ったドメインモデルが容易に構築できる．これは RDBMS を単なる永続化層としてではなく，データの整合性を保つための砦として使用できることを意味する．

著者は，予てから日本語問題記述からドメインモデルを工学的に導く研究を進めている [3][37]．それらの概略は，ひとつは，英語，すなわち，主語が目的語を動詞で制御する

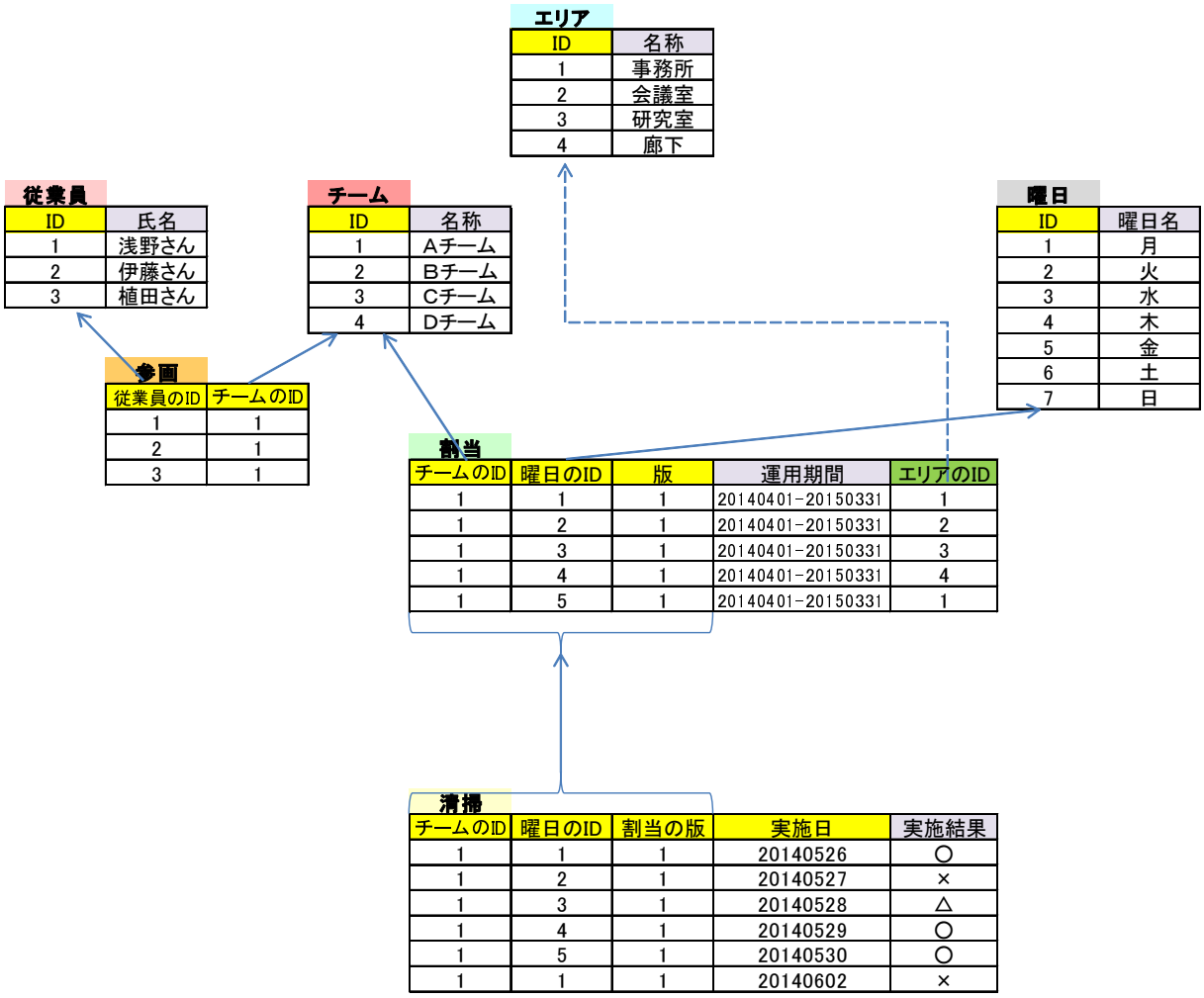

図 3.15: 検証のために用いた例題 3.7.1 のデータモデル

行為文を中心した言語の観点からドメインモデルを理解しようとするものであり，もう一つは，前者で得られた主語や目的語から知識の責務（オブジェクトや属性の候補），動作動詞から振る舞いの責務（操作の候補）を抽出し，責務間の依存度合いを数量化して MDS による責務の空間布置の観点からドメインモデルを導こうとするものである．とりわけ，責務間の依存度合いの数量化に際して今回の提案を内容を加えることで，モデル化のための精度が高められると期待している．

提案手法がオブジェクト指向手法において正規化と似た役割を果たし，ドメインエキスパートとシステムエンジニアが共同で参加するモデリング作業の一助となれば幸いである．

第4章　MDSによるモデルの妥当性検証

4.1　緒言

第3章では，存在従属分析を紹介したが，その手法で得られたモデルが，伝統的な手法で得られた概念モデルと空間的な一致を示すものかどうかを本章では確認しておきたい．

伝統的な手法として，本章では，ソフトシステムズ方法論に着目する．ソフトシステムズ方法論に着目した理由は，今日のシステム開発の現場では，多様化，複雑化する要求を合意形成し，機敏かつ柔軟に実現していくために，モデルとコンポーネントを中心とした開発プロセスが採用されているからである．

たとえば，最上流工程ではソフトシステムズ方法論（Soft Systems Methodology，以下SSM）など適用して，関係者間で目的についての合意を行い，それら実現のための分析・設計はオブジェクト指向アプローチ（以下，OOA）を採用する，といったプロセスがそのプロトタイプとして考えられる．そのため，SSMで得られた最上流のモデルをOOAのモデルと連動させることに大きな期待が寄せられている．しかしながら，SSMの概念活動モデルのような最上流工程の成果物とOOAのクラス図のような上中流工程の成果物を連動するための方法は確立されていないのが現状である．

一方，多次元尺度構成法（Multidimensional Scaling，以下MDS）は，心理学の分野で発展し，データの潜在的構造を幾何学的に表現するために非常に有効な手法となっている．近年マーケティングや行動科学などの多く分野で利用されており，最近ではビッグデータブームで再び脚光を浴びている．過去にMDSをOOAの構造パターンの評価に利用する提案[38]はあったが，SSMのモデルとOOAのモデルを連動させるために利用したという例はないようである．そこで，本章では，SSMの関連システムの根底定義（Root Definition）から知識の責務と振る舞いの責務を抽出し，それらの間の相互の関連度合いを簡便に数値化する基準を試案する．そして，責務間の関連度合いの行列を非計量MDSにて解析し，責務の空間配置をオブジェクトの責務配分に援用することで，SSMのモデルとOOAのモデルを連動する手法を提案しつつ，存在従属分析で得られたドメインモデルと比較する．

以下，4.2節では，提案手法の関連研究の概要について説明した後，4.3節では，例題を用いて提案手法を説明する，4.5節では別の例題についての提案手法の適用結果について報告する．4.6節は本章の結言である．

4.2 関連研究の概観

4.2.1 ソフトシステムズ方法論

SSM は，問題が明確に定義されていない状況において，ステークホルダの世界観のアコモデーション [1] を探り，問題の解決に導くプロセスである [39][40]

SSM は，7 つのステージから成るプロセスで構成されており，その第 4 ステージでは，複数の候補から選択された関連システム（目的を持った人間活動システム）の根底定義から概念活動モデル（Conceptual Model：目的を実現するために必要な要素活動）を作成する．そして，それらを以降のシステム構築作業のモデルに結び付ける．

4.2.2 オブジェクト指向方法論と責務配分

一方，OOA の本質は，あるオブジェクトに求められる振る舞いの責務に対して，その責務遂行に必要な知識の責務を凝集（カプセル化）することにより，モジュールとしてのオブジェクトの強度を高めつつ，他のオブジェクトへの依存性（結合度）を弱めて，保守性，再利用性，拡張性の高いシステム構築に資することである．

そのため，オブジェクトの識別とオブジェクトへの適切な責務の配分は重大な関心事であり，従来から多くの原則やガイドラインが提案されてきた．たとえば，ヤコブソンの BCE[36]， ワーフスブラックのロールステレオタイプ [41]， ラーマンの GRASP[42]，マーチンの SOLID 原則 [43] などが挙げられる．

しかし，上記のガイドラインは，あくまでも責務分割のための一般的な基準やガイドラインを提供してくれるだけであり，利用者の個別の問題に即したヒントを提供してくれるわけではない．小さな例ではあるが，ビリヤードゲームのシミュレータを OOA で開発することを想像した場合，ボールと何かの衝突判定と，衝突後のエネルギーの交換を，それぞれどのクラスの責務にすべきかについて上記ガイドラインだけで判断することは容易ではない．

結果として，責務配分の妥当性が評価可能となるタイミングは，ユースケース実現の構造モデリングと振る舞いモデリングが一通り完了した後になってしまうという問題がある．できれば，もっと早い段階で，オブジェクトへの責務配分のアウトラインを把握したい．

[1] 異なる視点・価値観を持つ人同士が，議論を重ねて互いの視点・価値観を共存させること

4.2.3 多次元尺度構成法と対象の空間布置

多次元尺度構成法（Multi-Dimentional Scaling : MDS）は，計量 MDS と非計量 MDS に大別される．計量 MDS は，間隔尺度以上のデータについて，データの中に潜む構造をできるだけ少数次元の空間で幾何学的に表現する手法であり，非計量 MDS は計量 MDS を順序尺度のデータに対しても扱えるように拡張したものである [44]．MDS をイメージ的に理解するために北海道の地図を例として用いる．地図から都市間の距離を求めるのは定規で測れば容易である．しかし，逆に都市間の距離関係のみが与えられている場合，その情報だけから都市を平面上に布置して地図を復元することは容易ではない．

表 4.1: 北海道の都市間の直線距離：文献 [44] より引用

	札幌	旭川	稚内	釧路	帯広	室蘭	函館	小樽
札幌								
旭川	115							
稚内	274	202						
釧路	249	188	358					
帯広	152	118	313	98				
室蘭	88	200	360	290	195			
函館	152	263	413	330	240	64		
小樽	30	130	266	277	180	98	159	

図 4.1: MDS による空間布置に白地図を重ねたもの

MDS はこのような問題を解析的に解く手法である．表 4.1 は北海道の 8 都市間の直線距離を測定したものである．図 4.1 は表 4.1 のデータを MDS にかけて X-Y 平面上にプロットしたものである．白地図を重ね合わせると，位置関係は実際の地図上の都市とほぼ等

しく復元されていることがわかる．MDS は本来，既知の構造を可視化するのに用いることはないが，このようなモノとモノの一対間の距離的な関係（近似性でも非近似性でもいずれも扱える）を読み込んで，モノをできるだけ少数の次元（可読性からいえば 3 次元までが望ましい）空間に配置する手法である．

Kurskal の方法 [45] は，非計量 MDS の一種で，空間布置の適合度の指標としてストレスと呼ばれる値を用いる．ストレス値と空間布置適合度の評価の対応は概ね表 4.2 の通りであるとされる [45][46].

表 4.2: ストレス値と適合度の評価

ストレス値	評価
0.2	悪い(poor)
0.1	まずまず(fair)
0.05	よい(good)
0.025	すばらしい(excellent)
0	完璧(perfect)

4.3 MDS の適用手法

4.3.1 提案手法の概略

以上のような状況と成果を踏まえ，本節では，MDS を SSM と OOA のモデル連携に用いる方法を提案するとともに，存在従属分析で得られた概念モデルと MDS で得られた空間布置を比較する．

具体的には，関連システムの根底定義から概念活動モデルを構築する．そして概念活動モデルの各ノードから知識の責務と振る舞いの責務を抽出し，それらの間の相互の関連度合いを簡便に数値化する基準を試案する．そして，責務間の関連度合いの行列を非計量 MDS にて解析し，責務の空間配置を試みる．

本提案の全体像を図 4.2 に示す．図 4.2 の通り，提案手法では，SSM のシステムの根底定義から出発して，概念活動モデルから責務を抽出し，それらの間の関連度合を求めて，責務の空間布置を得る．これを後続の OOA のクラス図作成等の参考にしようというアイデアである．以下，提案手法のプロセスについて例題を用いて説明する．

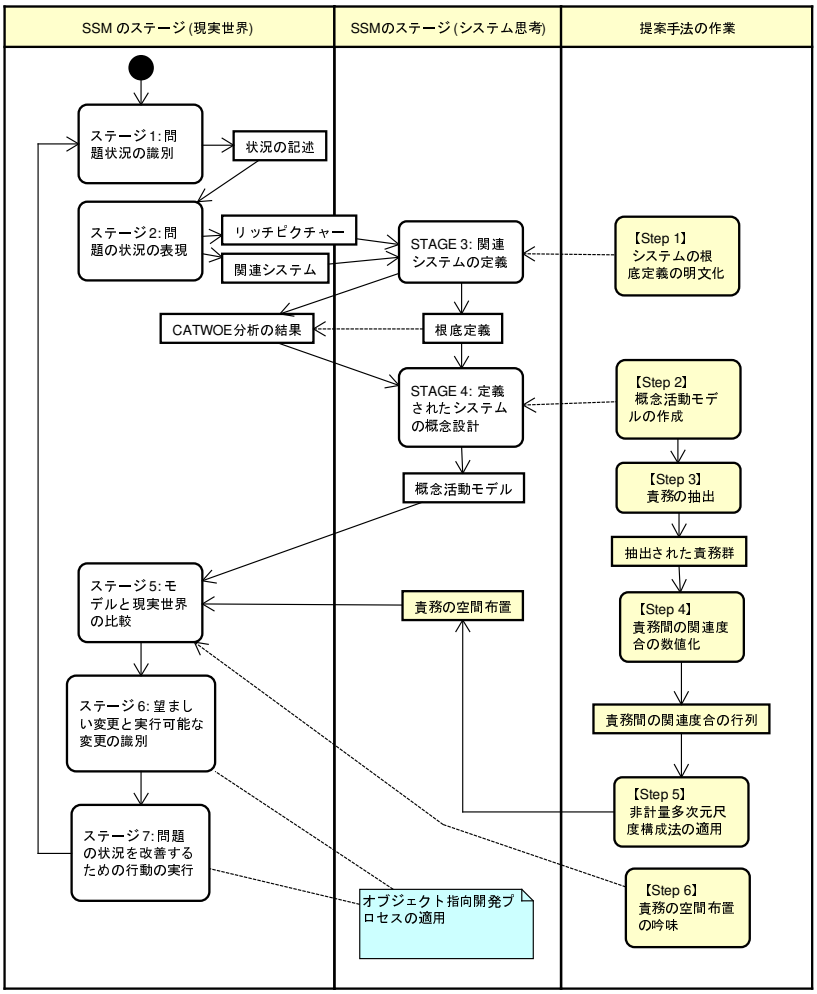

図 4.2: SSM のプロセスと提案手法

4.3.2 提案手法のプロセス

【Step 1】システムの根底定義を記述する

Step 1 は伝統的な SSM のステージ 3 をシステム利用者の視点で拡張する．具体的にはアクターを明確にしながらコンピュータ化すべき関連システムの根底定義を定義する．システムへの要求は，突き詰めれば,「いつ」,「誰が」,「何のために」利用するシステムなのかを定義することである．たとえば，ユースケースモデルでは,「いつ」はビジネスイベント,「誰が」はアクター,「何のために」はユースケースに対応する．また，SSM においては,「(Z) ～するために, (Y) ～することによって, (X) ～するシステム」の形式で表現されたものは関連システムの根底定義と呼ばれる．根底定義を用いて「想いの活動」を表現しておくことは，SSM の学問的基準である回復可能性を担保するために重要であるとされる [39]

提案手法では，SSM の要求定義の形式として，SSM の根底定義に「誰が (W)」の情

報を追加した形式を採用する．すなわち,「(W) ～が, (Z) ～するために, (Y) ～することによって, (X) ～するシステム」という形式を用いる．ここでの (W) はステージ 3 の CATOWE 分析 [2] で導かれた「C」の Customers が該当する．行為者としての主語を明確に掲げることで，概念活動モデルの各ノードを行為文として切り出しやすくする狙いがある．

たとえば,

【例 1】 「子供を入塾させようと考えている親 (W) が，塾を選ぶために (Z)，塾，教科，開講場所，月謝，特徴，講師などを検索することによって (Y)，比較検討するシステム (X)」

【例 2】 「ビリヤード愛好家が (W)，ゲームの腕を上げるために (Z)，コンピュータ上でゲームの動きをシミュレートすることによって (Y)，ボールの打ち方を学ぶシステム (X)」,

などが SSM で用いられる根底定義の例である．

【Step 2】概念活動モデルへ展開する

Step 2 は伝統的な SSM のステージ 4 をシステム利用者の視点で拡張する．すなわち，ステージ 3 で明らかになった関連システムのうち，コンピュータシステム化したい根底定義を選んで，概念活動モデルへと展開する．システムの根底定義は，システムへの機能的な要求を凝縮している．しかしながら，このような簡潔な定義からシステムの構成要素が持つべき責務をいきなり抽出することは情報不足のためできない．そこで，関係者（システム開発の依頼者と提供者など）間でモデリングセッションを行いつつ，根底定義から概念活動モデルに展開する．

概念活動モデルは，根底定義で表現された手段 (Y) を稼働させるのに必要な活動を 7 ± 2 個程度の動詞 (V) ＋目的語 (O) の形式で表現し，おのおのの活動を論理的依存性に照らして矢印（目的→手段）で結合したものである [40]．図 4.3 は，例 1 の根底定義を概念活動モデルに展開した例である．

[2] C：Customers：プロジェクトによって影響を受ける人，A：Actors：プロジェクトを遂行する人，T：Transformation Process：プロジェクト中の変換プロセス，W：World View：プロジェクト定義の背景にある世界観，O：Owners：このプロジェクトの主体，E：Environmental Constraints：および，プロジェクトに影響を与える外的環境要因，について分析する方法

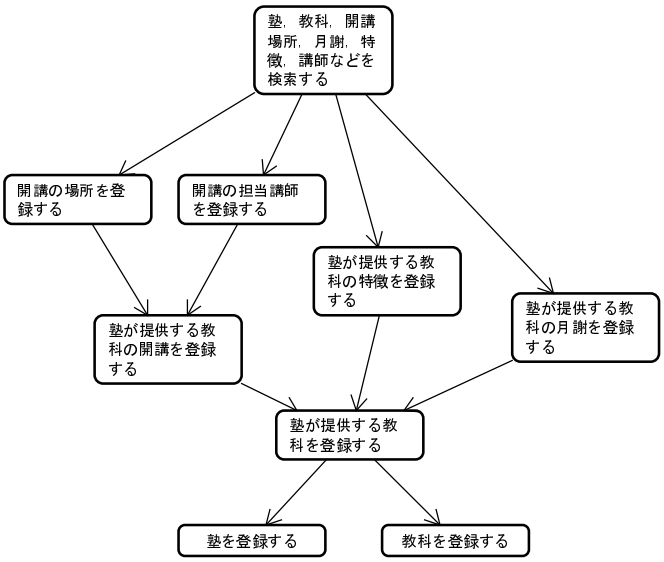

図 4.3: 例 1 の根底定義から展開した概念活動モデル

【Step 3】概念活動モデルから責務を抽出する

Step 3 は伝統的な SSM にはない，本提案独自のステップである．概念活動モデル上のおのおののノードから可算名詞[3] を知識の責務，動作動詞を振る舞いの責務として抽出する．この作業は，日本語の要求記述文からオブジェクトを抽出するのに比べて格段に容易である．なぜならば，日本語の要求記述文は形式化されておらず，また，日本語の特徴から存在文が多いのに対して，概念活動モデルへ展開後は「V（動作動詞）＋ O（名詞）」の形に整形されているからである [47].

著者は先行研究において，英語の品詞とクラス図の要素の対応関係を明らかにしている [3]．この研究では，要求記述に出現する可算名詞がクラスまたは属性，不可算名詞[4] が属性または属性値，動作名詞が操作，状態動詞が関連にそれぞれ対応することを明らかにした．ここでは，それらの知見を適用する．可算名詞については，クラスか属性かを俄かに判断することは難しい．しかし本節の手法では後で MDS による責務の空間布置を確認するため，この段階では単純に名詞を知識の責務，動詞を振る舞いの責務とみなしてよい．そのため，オブジェクトにすべきか属性にすべきかの判断にこの段階で迷う必要はない．表 4.3 に，例 1 の概念活動モデルから抽出された知識の責務と振る舞いの責務を示す.

[3]正確には，名詞の可算用法である．
[4]正確には，名詞の不可算用法である．

表 4.3: 例 1 の概念活動モデルから抽出された知識の責務と振る舞いの責務

抽出された知識の責務	抽出された振る舞いの責務
塾	塾を登録する
教科	教科を登録する
塾・教科の提供	提供を登録する
提供の月謝	月謝を登録する
提供の特徴	特徴を登録する
提供の開講	開講を登録する
開講の担当講師	開講の担当講師を登録する
開講の場所	開講の場所を登録する
	塾情報を検索する

【Step 4】責務間の関連度合を見積もる

本手法では，後で MDS を利用するため，Step 3 で抽出した責務（それらはオブジェクトの候補，属性の候補，および操作の候補であるが，すべてひっくるめて）おのおのの間の関連度合を数値化する必要がある．

関連度合の見積もりは，手作業で行うならば，できるだけ簡便な方法が望ましい．そこで，著者は，以下のような基準で数値化することにした．基準は前述の情報エキスパートの原則 [42] を満足すべく試行錯誤の末，辿りついたものである．数値は，関連度合の強さに応じて順序尺度を満たしていればよいため，0（最弱）から 4（最強）の 5 段階の整数値を用いることにした．ただし，4（最強）は同一の責務（そのもの）間の関連度合であり，対角成分の既定値となる．また 0 は関連度合が最弱であることを意味する．なお，以降の責務間の関連度合いを示す表では，見やすさのため，対角成分と 0 の表示はサプレスしてある．

（ア）知識の責務同士の関連度合の数値化基準

この基準は，ある知識の責務にライフサイクルが依存するする知識の責務は，ライフサイクルの依存性がないその他の知識の責務よりも近い位置に配置されるべきであるという考え方に基づいている。具体的には，知識の責務間の関連度合の数値化では，その根拠を概念活動モデルのノードに記載された日本語の格助詞「の」に求めることにした．格助詞「の」の用法は多様であるが，概念活動モデルのノード上に記載される「の」は自ずと名詞と名詞を結び付ける「の」に限定されてくる．そしてそれらは，所属や所有を表現しているため，格助詞「の」で接続された名詞（すなわち知識の責務）間にはライフサイクルに関して何らかの依存性があり，それらの知識の責務間の関連度合は強いと考える．

たとえば,「開講の担当講師」であれば，講師は開講までに決定される必要があり,「塾の（による），教科の，提供」であれば，提供を登録する前に，塾と教科の双方が登録済

みである必要がある．

採用した基準を表 4.4 に示す．そして，表 4.5 はこの基準に従って知識の責務間の関連度合を求めた結果である．

表 4.4: 知識の責務同士の関連度合の数値化基準

関連度合値	意味	例
3	両者は所属，所有を表す格助詞「の」で結ばれていて，所属先あるいは所有者が一つの場合	開講の担当講師
2	両者は所属，所有を表す格助詞「の」で結ばれているが，所属先あるいは所有者が2つある場合	塾・教科の提供，（塾の提供，教科の提供に分解できる）
1	両者は所属，所有を表す格助詞「の」で結ばれているが，所属先あるいは所有者が3つある場合	商品の在庫，倉庫の在庫，期末の在庫
0	上記以外の場合	

表 4.5: 例 1 の知識の責務間の関連度合

知識 ＼ 知識	塾	教科	塾・教科の提供	提供の月謝	提供の特徴	提供の開講	開講の担当講師	開講の場所
塾								
教科								
塾・教科の提供	2	2						
提供の月謝			3					
提供の特徴			3					
提供の開講			3					
開講の担当講師						3		
開講の場所						3		

（イ）振る舞いと知識の責務間の関連度合の数値化基準

この基準は，ある知識の責務に対して，その状態遷移も含めてライフサイクルを制御する振る舞いの責務は，ライフサイクルを制御しない他の振る舞いの責務よりも近い位置に配置されるべきであるという考え方に基づいている．そこで，これらの責務間の関連度合の根拠を CRUD に求めることにした．CRUD とは，永続化オブジェクトを扱うソフトウェアが持つべき 4 つの基本機能である，Create（生成），Read（読み取り），Update（更新），Delete（削除）の頭文字を並べたものであり，ライフサイクルの制御の度合は $C \geqq D > U > R$ の順になる．

たとえば,「教科を登録する」という振る舞いの責務は,「教科」という新しい知識の責務を Create する．また,「開講の場所を登録する」という振る舞いの責務は,「開講の場所」という新しい知識の責務を Create するとともに,「開講」という既存の知識の責務について，その場所という知識の責務を Update すると考える．

採用した基準を表 4.6，例題への適用結果を表 4.7 に示す．

（ウ）振る舞いの責務同士の関連度合の数値化基準

この基準は，ある振る舞いの責務に対して，それを直接的に利用する振る舞いの責務は，それを利用しない振る舞いの責務や間接的に利用する振る舞いの責務よりも近い位

表 4.6: 振る舞いと知識の責務間の関連度合の数値化基準

関連度合値	意味	例
C or D=3	当該振る舞いの責務は，当該知識の責務を生成（Create）または削除（Delete）する場合	教科を登録する と 教科（教科を登録すると教科がCreateされる）
U=2	当該振る舞いの責務は，当該知識の責務を更新（Update）する場合	開講の場所を登録する と 開講（開講の場所がCreateされるとともに開講がUpdateされる）
R=1	当該振る舞いの責務は，当該知識の責務を参照（Read）する	提供を登録する と 塾および教科（塾と教科がReadされ，提供がCreateされる）
0	上記以外の場合	

表 4.7: 例 1 の振る舞いと知識の責務間の関連度合

振る舞い ＼ 知識	塾	教科	塾・教科の提供	提供の月謝	提供の特徴	提供の開講	開講の担当講師	開講の場所
塾を登録する	C							
教科を登録する		C						
提供を登録する	R	R	C					
月謝を登録する	R	R	U	C				
特徴を登録する	R	R	U		C			
開講を登録する	R	R	R			C		
開講の担当講師を登録する	R	R	R			U	C	
開講の場所を登録する	R	R	R			U		C
塾情報を検索する	R	R	R	R	R	R	R	R

置に配置されるべきであるという考え方に基づいている．そこで，振る舞いの責務同士の関連度合の数値化では，その根拠を概念活動モデルにおける活動ノード間の接続に求めることにした．たとえば，図 4.2 では,「塾が提供する教科を登録する」のノードが,「塾を登録する」と「教科を登録する」を直接的に利用しているため値 1 を与える．採用した基準を表 4.8，例 1 について求めた振る舞いの責務間の関連度合を表 4.9 に示す.

表 4.8: 振る舞いの責務同士の関連度合の数値化基準

関連度合値	意味	例
1	一方の振る舞いの責務ともう一方の振る舞いの責務は概念活動モデル上で直接矢印で結ばれている場合	塾が提供する教科を登録する と 塾を登録する 塾が提供する教科を登録する と 教科を登録する
0	上記以外の場合	

【Step 5】MDS を適用する

責務間の関連度合が数値化され，表 4.11 のような正方行列の下三角行列が得られたならば，これを MDS の入力とする．本手法で用いる責務間の関連度合の値は，より近いと考えられる責務間ほど相対的に大きな値が与えられたデータであるため順序尺度である．したがって，MDS のなかでも非計量 MDS を選択する必要がある．また，これらは責務間の近似性データであるとして処理を行う.

MDS を実行するには，SPSS[48] のような統計パッケージを使ったり，R 言語 [49] のような統計解析向けのオープンソースプロダクトを利用するなどの方法がある．しかし，今回はクラスカルの方法を文献 [45] の実装をもとに PHP 言語で実装した手組のプログラムを用いることにした．手組のプログラムであればその中身がよく分かっているので，将

表 4.9: 例 1 の振る舞いの責務間の関連度合

振る舞い ＼ 振る舞い	塾を登録する	教科を登録する	提供を登録する	月謝を登録する	特徴を登録する	開講を登録する	開講の担当講師を登録する	開講の場所を登録する	塾，教科，開講場所，月謝，特徴，講師などを検索する
塾を登録する									
教科を登録する									
提供を登録する	1	1							
月謝を登録する			1						
特徴を登録する			1						
開講を登録する			1						
開講の担当講師を登録する						1			
開講の場所を登録する						1			
塾情報を検索する				1	1		1	1	

表 4.10: 例 1 に関わる責務間の関連度合（表 4.5,4.7,4.9 の結合イメージ）

責務 ＼ 責務	塾	教科	塾･教科の提供	提供の月謝	提供の特徴	提供の開講	開講の担当講師	開講の場所	塾を登録する	教科を登録する	提供を登録する	月謝を登録する	特徴を登録する	開講を登録する	開講の担当講師を登録する	開講の場所を登録する	塾情報を検索する
塾																	
教科																	
塾･教科の提供	2	2															
提供の月謝			3														
提供の特徴			3														
提供の開講			3														
開講の担当講師						3											
開講の場所						3											
塾を登録する	3																
教科を登録する		3															
提供を登録する	1	1	3						1	1							
月謝を登録する	1	1	2	3							1						
特徴を登録する	1	1	2		3						1						
開講を登録する	1	1	1			3					1						
開講の担当講師を登録する	1	1	1			2	3							1			
開講の場所を登録する	1	1	1			2		3						1			
塾情報を検索する	1	1	1	1	1	1	1	1				1	1		1	1	

来の研究において概念活動モデルエディタに MDS を組み込む場合に有利であると考えたためである．表 4.11 のデータを MDS の入力として処理した結果，図 4.4 に示す責務の空間布置を得た．

4.4　責務の空間布置の評価についての議論

図 4.4 を眺めると，塾と教科とが明確に分離され，それらを結んだ線のちょうど中間あたりに提供の月謝や提供の特徴といった責務が配置されているのがわかる．ストレス値にいては，0.188 と高いものの，同じ例題について，モデラーが存在従属分析でクラス図を作成した場合（図 4.5）とよく似た構造を得ることに成功している．

ここで，責務の空間布置を読む際の留意事項について述べる．空間布置には関連は表示されないため，読み手は関連線を補って読む必要がある．ここでの関連は，システムの根底定義に由来する．そもそも概念活動モデルは，一つの関連システムの根底定義（目的）のために，それを実現すべく展開した 目的←手段 グラフであるため，おのおののノード上に記載される知識の責務（属性）は，いずれもが目的の実現のために必要なものである．

保守性，理解性，再利用性に目をつぶれば，それらは１つのクラスの属性としてまとめたとしても動作するシステムを構築することができる．このことから，これら全ての知

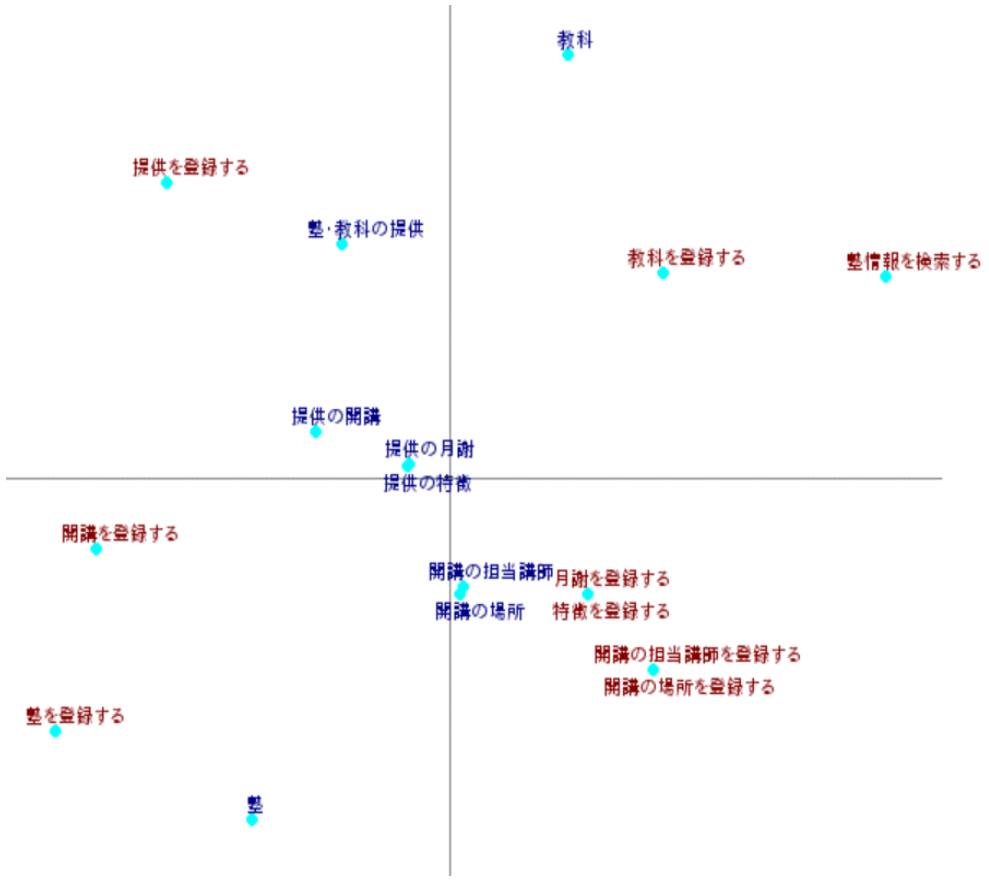

図 4.4: MDS で得られた例 1 の責務の空間布置 (繰り返し計算回数 44 回でストレス値は 0.188 に収束)

識の責務間には本来，直接的あるいは間接的に関連が存在していると考える必要がある．したがって MDS の空間布置を眺めて，近傍にまとまっている責務群をクラスとして取り出したならば，取り出したクラス間にも関連があるものと考えなければならない．もともと，ソフトウェア工学は,「本質的に関連性が強い要素同士を凝集しておけば，将来の改造に対して安定である」との視点に立って責務の分配を設計してきた [16] が，MDS の出力はその視点を補うものである．

4.5　他の例題への適用

本節では，4.3.2 節で示した 2 つ目の例題に対して提案手法を適用する．すなわち,「ビリヤード愛好家が（A），ゲームの腕を上げるために（Z），コンピュータ上でゲームの動きをシミュレートすることによって（Y），ボールの打ち方を学ぶシステム（X）」を根底定義とした例である．1 つ目の例題よりも若干複雑な例を用いて責務数を増やした場合のストレス値の変化を観測するとともに，空間布置を参考に作成した分析クラス図をもとに設計，実装までを行い提案手法の妥当性を確認する．

根底定義から，概念活動モデルを展開すると図 4.6 を得る．概念活動モデル上の名詞か

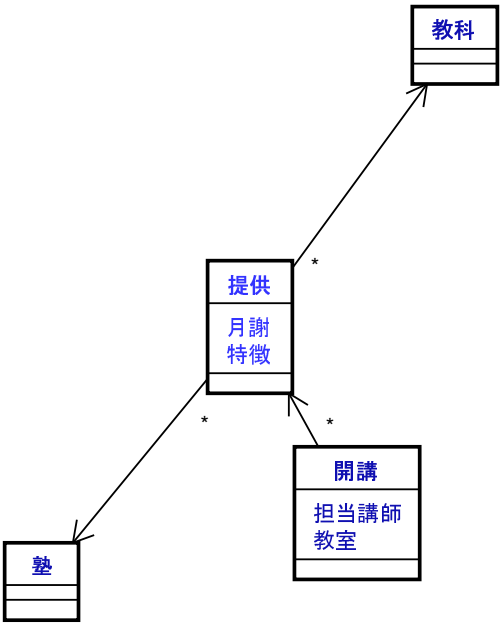

図 4.5: 例 1 についてモデラーが存在従属分析法で作成したクラス図

ら知識の責務，動作動詞から振る舞いの責務を抽出し，おのおのの責務間の関連度合を見積もって表にまとめると表 4.11 を得る．そして，表 4.11 のデータを MDS にかけると図 4.7 を得る．その際のストレス値は 0.092 に改善された．これは責務数が増え責務間の情報量が増えたためと考えられる．

表 4.11: 例 2 の概念活動モデルから求めた責務間の関連度合

責務 ＼ 責務	テーブル	テーブルのクッション	テーブルのポケット	テーブルのボール	クッションの位置	ポケットの位置	ボールの位置	ボールの速度と質量	ボールの転がり摩擦	テーブル上のボールの初期位置を決める	クッションの位置を調べる	ボールの位置を調べる	ポケットの位置を調べる	ボールの速度と質量を調べる	ボールの転がり摩擦を調べる	ボールをテーブルに並べる	ボールとクッションの接触を検知する	ボールを移動する	ボール同士の接触を検知する	ボールとポケットの接触を検知する	ボールを突く	ボールとクッションの衝突を再現する	ボール同士の衝突を再現する	ボールがポケットに落ちることを再現する	コンピュータ上でゲームの動きを再現する
テーブル																									
テーブルのクッション	3																								
テーブルのポケット	3																								
テーブルのボール	3	2	2																						
クッションの位置	2	3																							
ポケットの位置	2		3																						
ボールの位置	2			3																					
ボールの速度と質量				3																					
ボールの転がり摩擦				3																					
テーブル上のボールの初期位置を決める	1			2			2	2																	
クッションの位置を調べる	1	1			1																				
ボールの位置を調べる				1			1																		
ポケットの位置を調べる			1			1																			
ボールの速度と質量を調べる				1				1																	
ボールの転がり摩擦を調べる				1					1																
ボールをテーブルに並べる	1			1			2	2		1															
ボールとクッションの接触を検知する		1		1	1		1																		
ボールを移動する		1	1	1	1	1	1	2	1			1		1	1										
ボール同士の接触を検知する				1			1					1													
ボールとポケットの接触を検知する			1	1		1	1					1	1												
ボールを突く				2			2	2								1									
ボールとクッションの衝突を再現する		1			1		2	2			1	1													
ボール同士の衝突を再現する				2			2	2	1																
ボールがポケットに落ちることを再現する			1	2		1	2	2												1					
コンピュータ上でゲームの動きを再現する	1	1	1	1	1	1	1	1	1												1	1	1	1	

図 4.7 の空間布置を眺めると，ボールの転がり摩擦が，ボールよりもむしろテーブル寄りに配置されていたり，ボール同士の接触を検知する責務はテーブル寄り，ボールと何かの衝突を再現する責務はボール寄りに布置されているのが興味深い．図 4.8 のクラス図は図 4.7 の空間布置を見ながら作成したものである．多少の主観もあるが，クラス図作成時には図 4.7 の情報が大いに参考になった．

さらに，図 4.8 のクラス群を Model-View-Controller アーキテクチャパターン [50] の Model とし，View と Controller のクラス群を追加して設計モデルを作成した（図 4.9）．そして，Java 言語にて実装を試みた結果も良好であった．とくに MDS の布置によって分

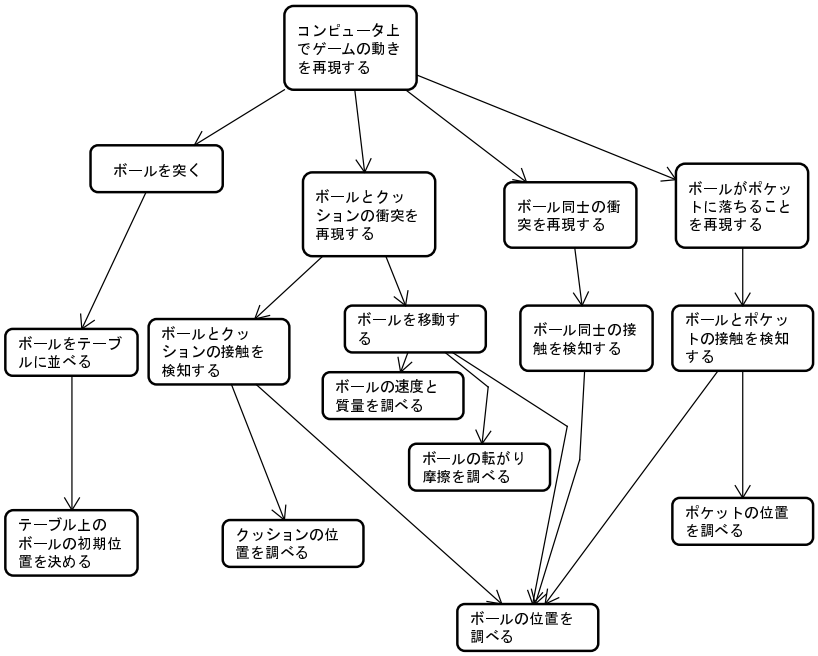

図 4.6: 例 2 の概念活動モデル

析段階のモデルから接触の検知と衝突の再現を分離できていたことが，ボールクラスの責務を軽減し，後続の詳細設計および実装作業を容易にした．

以上の適用実験から，本提案が示した基準による責務の空間配置は，機能要求に照らして真に機能的なものか否かは不明であるが，少なくとも高凝集と疎結合を目指して責務のカプセル化を行うオブジェクト指向の本質とは整合しており，OOA の責務分配を支援するものと考えられる。

4.6 結言

本章では，最上流工程の方法論として SSM，中上流工程の方法論として OOA を仮定し，システムの根底定義から概念活動モデルを経由して抽出した責務間の関連度合を数値化し，MDS を用いて空間布置したものをクラス図作成の参考にするアイデアを示した．そして，得られた知識の責務の空間布置は，存在従属分析により得られたクラス図のそれと矛盾しなかった．このことから，第 3 章で紹介した手法と，本章で紹介した手法は，相互補完的に使えるのでは，と考えられる．

著者は，オブジェクト指向といえば，先にオブジェクトを見つけることに気を奪われがちであったが，逆に，先に責務を見つけて，空間にばら撒けば，関連度合に応じて責務のグループができ，それを眺めてクラス図を考えるアプローチも有効であると考える．また，SSM の成果物を OOA の責務分配設計に繋げる方法を提示した点にオリジナリティがある．

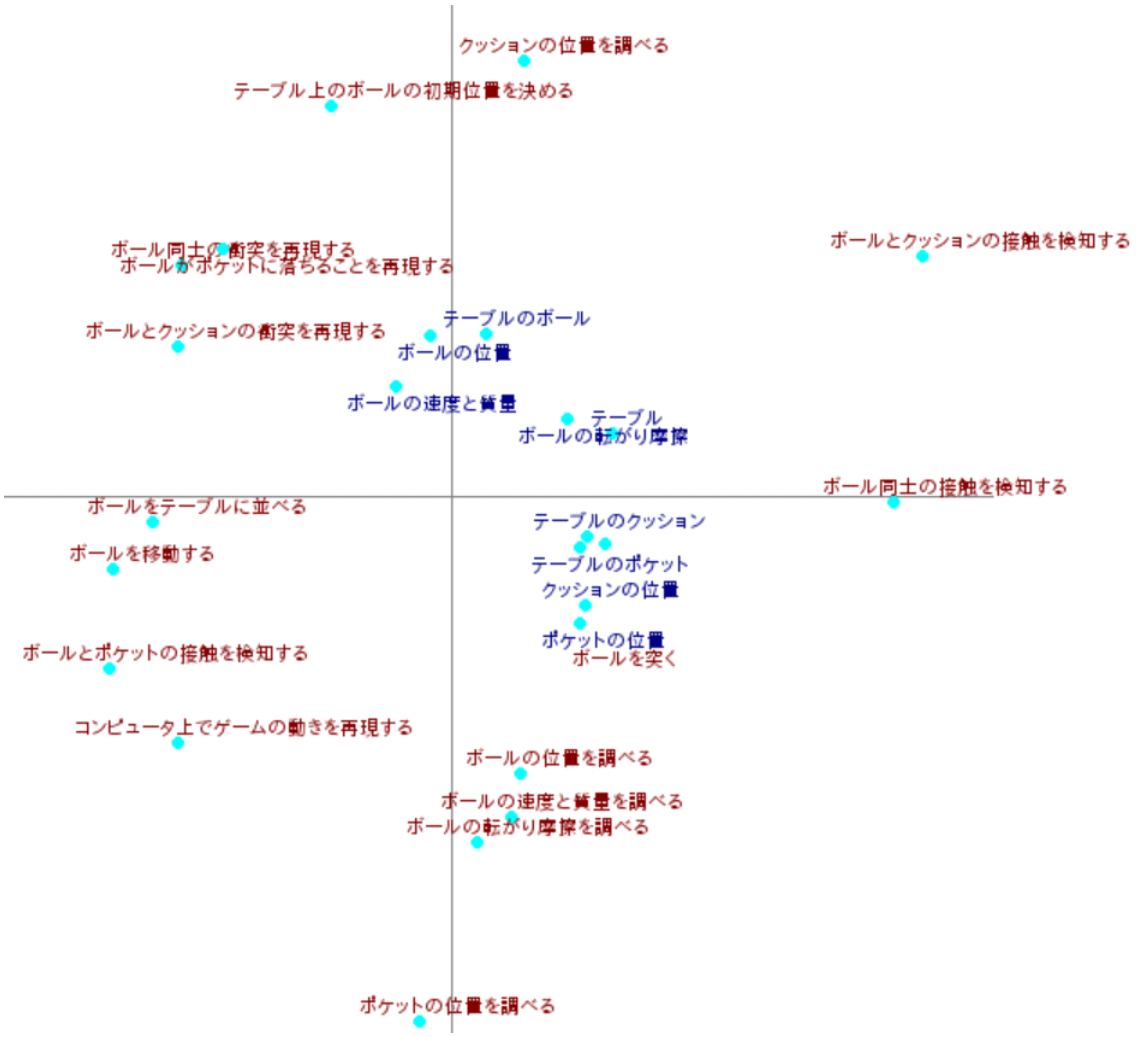

図 4.7: 例 2 の責務の空間布置モデル

ただし，提案手法を手作業で実施するのは非常に煩雑であるため，将来は MDS を組み込んだ概念活動モデルエディタを実装し，提案手法の責務の抽出からそれらの空間布置までを自動化することを目指したい．自動化が前提であれば，責務間の関連度合をさらにきめ細かいものにすることができる．たとえば，概念活動モデル上のあるノードから別のノードへ至るまでに経由する矢印の本数を責務間の関連度合に加味するなどが考えられる．実装すれば，クラス図を提案してくれるインテリジェントなモデリングツールが実現できる可能性がある．引き続きこれらのテーマについて取り組んでゆきたい．

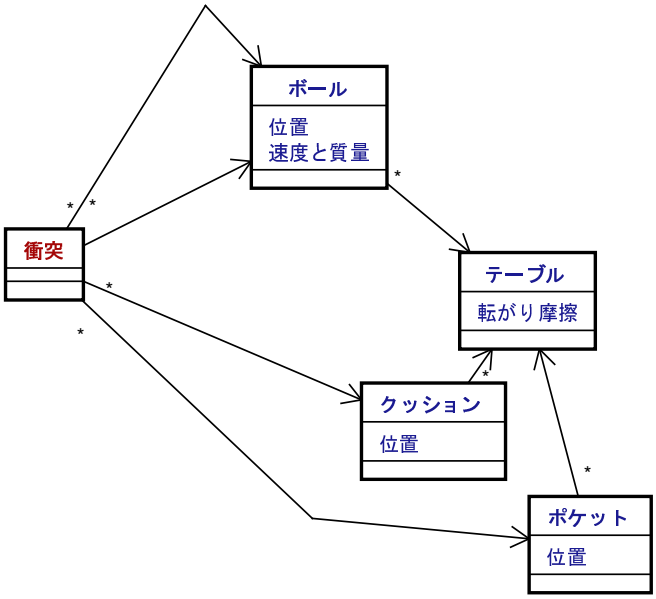

図 4.8: 例 2 についてモデラーが存在従属分析法で作成したクラス図

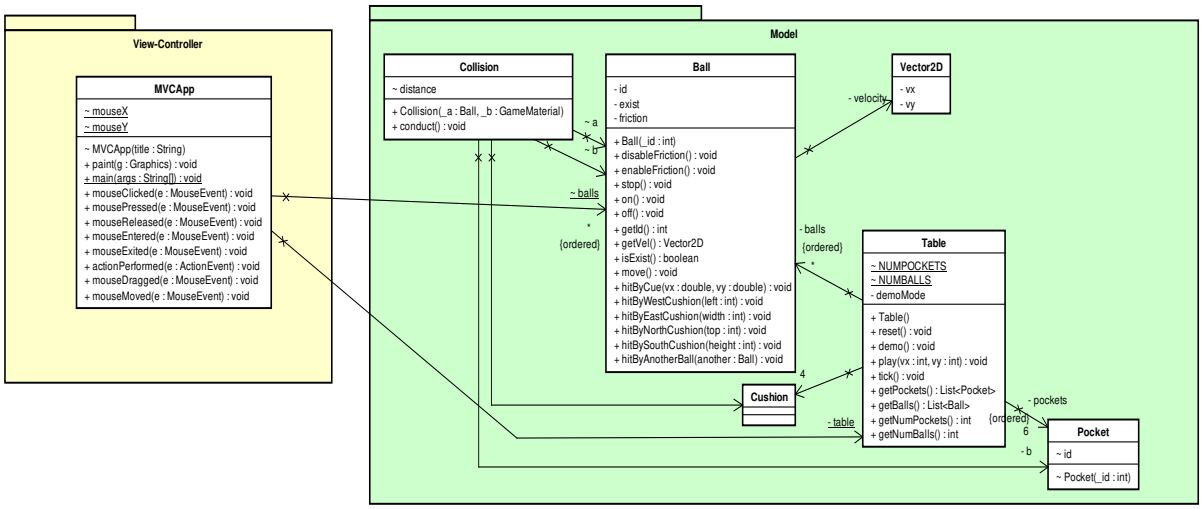

図 4.9: 例 2 のアーキテクチャ設計例

第5章　ドメインモデルからのユースケース自動生成

5.1　緒言

近年，業務システムは，OOSE(Object-Oriented Software Engineering)[36] やICONIX[51] に由来する，ユースケース駆動開発手法に基づいたソフトウェア開発が行われている．このユースケース駆動では，自然言語による要求記述から直接，ユースケース図を作成することになる (図 5.1 参照).

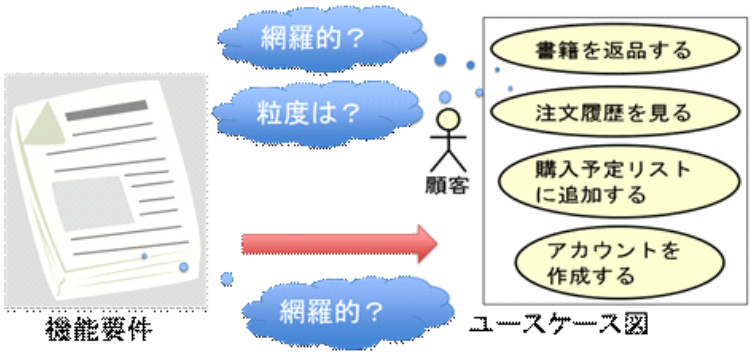

図 5.1: ユースケース駆動開発手法の問題点

しかし，この従来のユースケース駆動方式には，2 点の問題がある．まず，第 1 点は，要求記述が網羅的である保証がない点である．ユースケース駆動では，要求記述が網羅的であると想定しており，要求記述からそのままユースケースを作成する．そのため，要求記述の漏れに気が付きにくく，ユースケースを網羅的に識別できる保証がない問題点がある．2 点目は，ユースケースの粒度は，担当者の主観に任されており，曖昧性を残している点である．ICONIX ではユースケースの粒度について，ユースケース記述の基本系列と代替系列がそれぞれ 10 センテンス以上になればユースケースを分割せよ，とのガイドラインがある．しかし実際には，図 5.2 に示すように，10 センテンス以内であってもユースケースを分割する必要がある場合があり，ユースケース分割の判断基準が明確ではない[1].

[1] この問題は，ICONIX の問題というより，ユースケースの粒度の問題である．ICONIX は一つの例として，ご理解をいただきたい．

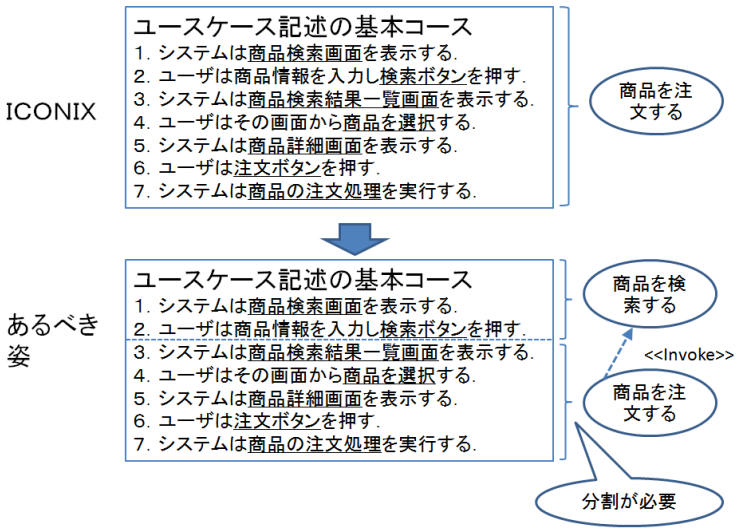

図 5.2: 従来手法でのユースケースの粒度の曖昧性

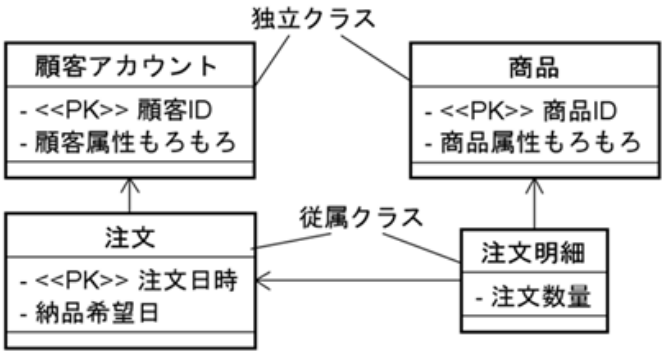

図 5.3: 存在従属クラス図の例

この問題を解決するため，本章では，存在従属性を併記したクラス図 (以下，存在従属クラス図)[52][4] に基づいて，ユースケース図を自動的に生成する手法を提案する [2].

あるクラスのインスタンスが，別のクラスのインスタンスの先立つ存在を前提として存在するとき，前者は後者に存在従属する．この定義から，インスタンスの時間的な処理手順について幾つかの制約を導くことができる．例えば，あるクラスのインスタンスは，従属先（親）クラスのインスタンスが生成された後でないと生成できない，といった制約がある．この制約に着目すれば，時間的な処理手順を含むユースケースを自動的に生成できる．

[2]RUP(ラショナル統一プロセス) 等のすでに確立した既存のユースケース駆動手法であつかう「ユースケース」は永続化データの CRUD だけではなく，システム間の連携や保守・管理等の幅広い範疇を含む．これに対して，本章のユースケース図自動生成アルゴリズムが扱い得るのは，クラス図（本章で扱う存在従属クラス図は，もともと，多対多の関連を持たない．また，操作（operation）も無く，主キーもクラス毎に明示されている．したがって，そのまま RDB のテーブルに変換できる）として抽出された各クラスの CRUD を行うユースケースのみである．ユースケースの粒度としても，その範囲のみを扱う．その意味では，扱えるユースケースの範囲に限定がある．後述の 5.4.2 は，その実使用上でのカバー率を評価することを目的としている．

提案手法では，最初に，自然言語による要求記述から，永続化するデータ[3] の存在従属クラス図を作成する．次に，その存在従属クラス図を基にユースケース図を自動生成する．最終的に，自動生成したユースケース図を見ながら，必要に応じて要求記述を修正したり，自動生成されたユースケース図を修正したりする（バージョン管理が必須である）．

提案手法では，ユースケース図を自動生成するため，ユースケースの粒度が均一になると考える．また，インスタンスを CRUD(Create, Read, Update, Delete) するユースケース及び，そのインスタンスの時間的な処理手順を含むユースケースを自動的に生成できるため，自然言語による要求記述漏れが発見しやすいと考える．

以下，5.2 節では，存在従属性について説明する．5.3 節では，提案手法について述べる．5.4 節では，評価実験について述べ，5.5 節は関連研究と考察を示し，5.6 節は本章の結言である．

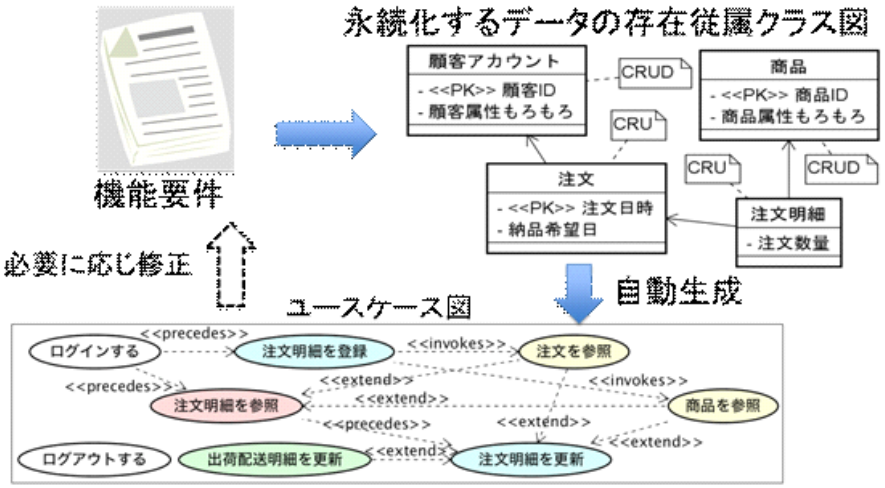

図 5.4: 提案手法のサイクル

5.2　存在従属について

5.2.1　存在従属クラス図

提案手法では，永続化するデータのクラス図を作成するとき，クラス間の関連に対して，存在従属性というクラス間の関係性を記述する．存在従属性を併記したクラス図を存在従属クラス図という．存在従属性の概念は，Chen が用いている [30][25]．また，井田

[3]Boundary, Control, Entity のうちの Entity である．

らは，存在従属性に着目したモデリング手法，及び存在従属クラス図の簡易表記法を提案している [52].

存在従属関係にあるクラスの種類には，独立クラス (Master class) と，従属クラス (Dependent class) が定義されている．独立クラスのインスタンスは単独で存在でき，どのクラスにも依存しない．一方，従属クラスのインスタンスは，別のクラスのインスタンスの存在を前提として存在する．このようにインスタンスが持つライフサイクルの関係性，つまりクラス間の関連に対して存在従属関係を記述することで，インスタンスの半順序的な処理手順を含意できる．

存在従属クラス図の具体例を図 5.3 に示す [4] 注文管理システムにおいて，永続化するデータのクラス図の一部である．顧客アカウントクラスと商品クラスそれぞれのインスタンスは，システムのリソースであり単独で存在できるため，独立クラスである．顧客アカウントと，商品クラスの情報つまりインスタンスが存在しなければ，顧客が商品を注文できないことは自明である．そのため，注文クラスは顧客アカウントに存在従属する．同様に，注文明細クラスは，注文クラスと商品クラス両方に存在従属する．特に注文明細クラスのインスタンスの生存期間は注文クラスのインスタンスのそれと等しいため，注文明細クラスは注文クラスに属性的に従属していると考えられる．

存在従属の定義から，インスタンスの時間的な処理手順について 4 つの制約を導くことができる．これらの制約は，3.5 節で説明した存在従属関連の性質から導かれる．

(1)**【生成制約】**　：従属クラスのインスタンスは，従属先クラスのインスタンスが作成された後でなければ生成できない．

(2)**【削除制約】**　：従属先クラスのインスタンスが削除されたら，従属クラスのインスタンスを削除しなければならない．

(3)**【更新制約】**　：従属先クラスのインスタンスが更新されたら，従属クラスのインスタンスは，従属先クラスの更新の影響を受けて更新される可能性がある．例えば，出荷配送明細クラスが，注文明細クラスに存在従属しており，注文明細クラスに注文数量，出荷配送明細クラスに出荷数量という属性をそれぞれ持っているとする．注文数量を増減させるように注文明細を更新した場合，出荷数量が変更される場合がある．

(4)**【参照制約】**　：従属クラスのインスタンスを参照する場合，従属先クラスのインスタンスを参照する可能性がある．従属クラスは，従属先クラスに情報の依存をしているためである．

[4]著者の存在従属アプローチでは，オブジェクトとして対象世界を捉えていても，無条件にすべてのクラスのインスタンスに，クラス毎に独自の ID(主キー）を付与することは考えない．独自の ID が付与されるのは，独立クラスのみである．図 5.3 に示すように，下流側のクラスは，存在従属の上流側にあるクラスの ID と（必要であれば）シーケンス番号やタイムスタンプを追加して主キーを作る．

表 5.1: 存在従属クラス図の関連と意味

関連のタイプ	インスタンス同士のライフサイクル制約	識別子	表記	提案アルゴリズムでの扱い
存在従属関連	従属元（親）クラスのインスタンスは従属先（子）のクラスのインスタンスよりも先に存在していなければならない．また，親クラスのインスタンスは子クラスのインスタンスよりも先に消滅してはならない．また，子クラスのインスタンスは，親クラスのインスタンスをそのライフサイクル中に挿げ替えできない．したがって，親クラスのインスタンスと子クラスのインスタンスのライフサイクルを比較した場合は，必ず前者の親クラスのインスタンスのライフサイクルの方が長い期間となる	子クラスのインスタンスは親クラスのインスタンスの識別子を，自分自身の識別子の一部（主キー兼外部キー）として持つ．もしも，子クラスがイベント（時点が帰属するクラス）の場合は，その識別子として必ずインスタンスの生成時点を表す時刻（タイムスタンプ）を識別子に加える	(UML クラス図の) 誘導可能性を持つ関連	被従属クラスに帰属するユースケースから従属クラスに帰属するユースケースに向けての≪ extend ≫依存関係を生成する．従属クラスに帰属するユースケースから被従属クラスに帰属するユースケースに向けての≪invokes ≫依存関係を生成する
属性従属関連（強い存在従属関連）	あるクラスのインスタンスの存在期間とそのクラスに含まれる属性の値の存在期間は一致していなければならない．この関係をここでは属性従属と呼ぶ．本来はあるクラスの属性でありながら，その項目が繰り返したり，構造を持つ場合は，注文と注文明細のようにもとのクラスから分離される	属性値は，当該クラスのインスタンスの識別子を，自分自身の識別子（主キー兼外部キー）として持つ．つまり，少なくとも，クラス図は第 3 正規系でなければならない	同上，または，コンポジション	
参照関係	参照先クラスのインスタンスのライフサイクルはそれを参照する（参照元の）クラスのインスタンスのライフサイクルとは独立である．参照元のインスタンスが参照先のインスタンスよりも先に存在する場合や，参照先のインスタンスが参照元のインスタンスよりも先に消滅する場合は，参照元のインスタンスにおいて，参照先のインスタンスが未定（null）である状態が許される．また，参照先のインスタンスは，参照元の都合でライフサイクル中に自由に変更することができる	参照元のクラスのインスタンスは参照先のクラスのインスタンスの識別子を，自分自身の属性の一部（外部キー）として持つ	(UML クラス図の) 依存関係	被参照クラスに帰属するユースケースから参照クラスに帰属するユースケースに向けて≪ extend ≫依存関係を生成する．参照クラスに帰属するユースケースから被参照クラスに帰属するユースケースに向けての≪precedes ≫依存関係を生成する
汎化関係	スーパークラスのインスタンスのライフサイクルとそのサブクラスのインスタンスのライフサイクルとは全く独立である	サブクラスの識別子はスーパークラスのエンティティの識別子と同じ属性である	(UML クラス図の) 汎化関係	サブクラスに帰属するユースケースからスーパークラスに帰属するユースケースに向けての≪ extend ≫依存関係を生成する

5.2.2　存在従属クラスにおける「関連」

存在従属クラス図の理解を容易とするため，存在従属クラス図における，1) 「関連」の種別，2) ライフサイクルの関係，3) 識別子，および，4) クラス図上 の表記と提案アルゴリズム上の扱いを表 5.1 に整理しておく．存在従属クラス図の詳細については，本論文の第 3 章や文献 [52] を参照されたい．

5.2.3　独立クラス・従属クラスの識別方法

本節では，日本語で記載されて業務要件記述から，永続化するデータをどのように選択するのか，そして独立クラスと従属クラスへの分類をどの時点でどのように行うのか，について述べる．前者は，著者の先行研究の文献 [53] に，後者については文献 [52] に，詳しく述べられているため，そちらを参照されたいが，ここでは，それらの概略を述べる．

まず，永続化するデータの選択は，得られた日本語業務要件記述を動詞が 1 つの行為文にリライトし，可算名詞[5]，動作動詞，状態動詞を抽出することによって行う [53]．リ

[5]正確には，名詞の可算用法である．

ライトを行うのは，日本語要求記述はその殆どが存在文であり，主語から目的語に対してのエネルギー伝搬を明確に捕捉できないためである．

たとえば,「顧客から舞い込んだ注文の商品の倉庫保管の在庫を引当てる」という業務記述は,「顧客は商品を注文する」,「商品は倉庫に保管されている」「従業員は，商品の在庫を引当てる」といった行為文にリライトする．そして，リライトした文の可算名詞はクラス候補に，動作動詞は操作の候補に，状態動詞は関連または，関連クラスの候補とする．ただし，動作動詞の結果を保存する場合は，動詞を名詞化したクラスを識別する．この例では，クラスの候補として，顧客，注文，商品，従業員，倉庫が，操作の候補として，注文，引当が，関連クラスの候補として保管，在庫がそれぞれ識別される．これらは要求記述のドメインにおける重要な管理対象であるため，すべて永続化するデータ（エンティティ）の候補となる．

次に，独立クラスと従属クラスへの分類をどの時点でどのように行うのか，についてである．文献 [52] に従うと，まず，エンティティの候補から，他のインスタンスの先立つ存在を前提とせず，単独で存在可能なエンティティ（独立クラス）を識別する．この例では，顧客，商品，倉庫が該当する．それができれば，それぞれに単一属性の ID（identifier）と主な属性を付与する．

つぎに，残ったクラス候補を独立クラスに存在従属させる．たとえば，注文は，注文主である顧客と注文対象である商品の先立つ存在を前提とするため，顧客と商品に存在従属させる．そして，注文の識別子を，注文主である顧客の ID，注文が発生した時点，注文対象の商品 ID の複合識別子にすることで，この事象「注文」の位置付け（従属関係）を明確にする．同様に，保管は保管主である倉庫と保管対象である商品の先立つ存在を前提とするため，倉庫と商品に存在従属させる．保管の識別子は，当然，倉庫 ID と商品 ID の複合識別子となる．これを図式化すれば存在従属クラス図が得られる．

なお，存在従属クラスの表記は UML クラス図のそれを流用しているため，モデル要素の意味は原則 UML クラス図のそれと同じである．しかし，モデル要素の意味を拡張しているものについては表 5.1 に示す．表 5.1 には UML クラス図と重複する表記が登場するが，その表記の意味は，存在従属クラス図としてクラス図を作成する場合には，表 5.1 に記載したモデル要素の意味を優先させるものとする．

5.3　ユースケースの自動生成

5.3.1　提案手法の概要

まず，機能要件から，永続化するデータの存在従属クラス図を作成する (図 5.4 参照) [6]. このように，設計の最初に存在従属クラス図を作成する点が，ユースケース駆動と異なる点である．業務システムの基本要素は，管理すべきデータ（Entity）であるため，まず存在従属クラス図を作成する．存在従属性の概念は理解しやすく，機能要件の中で見極めやすいため，第 4 正規形以上の正規化レベルを持ったクラス図を容易に構築できるとされる [52].

次に，存在従属クラス図からユースケース図を自動生成する手法を提案する [53][56]. 存在従属クラス図にはインスタンスの時間的な処理手順が含意されているため，半順序的な処理手順を含んだユースケース図が自動生成できる．その後，自動生成されたユースケース図を見ながら，必要に応じて機能要件を修正する．機能要件が変更されたら，必要に応じてクラス図の修正及びユースケース図の再自動生成を行う.

文献 [53] にも示した様に，自動化を行わなくても，提案のアルゴリズムを追ってゆけば，手作業でも生成は可能であるが，効率的では無いと考えて，本章では，自動生成を用いている．ただし，アルゴリズムの実装の詳細については，本論文では触れず，アルゴリズムの基本的な考え方を示す.

5.3.2　ユースケース図自動生成アルゴリズム

図 5.5 に，ユースケース図自動生成アルゴリズムの大まかな流れを示す．以下では，アルゴリズムの中味について記述する.

(1)　存在従属クラス図の準備

自動生成アルゴリズムの適用以前に，まず，存在従属クラス図を作成する．業務システムの基本要素は管理すべきデータであるためである．ただし，存在従属クラス図の各

[6]仕様記述から存在従属クラス図を作成する方法論は，基本的には，従来のクラス図作成と変わらない. なお，図 5.3 で示された主キーの設計法は，椿・渡辺によるデータモデリングの手法 [54][55] の結果とよく似ている．椿・渡辺のアプローチは，存在従属とは異なる発想により作成されるものであるが，結果として生成されるデータモデルは，著者の存在従属クラス図 [52] と極めて似ている．ただし，存在従属クラス図の主キーとして，通常のオブジェクト指向で見られる「オブジェクト ID」をそれぞれのクラスに付与するアプローチもあり得る．この図 5.3 の主キー構造は，本章で開示されたアルゴリズムが必要とするものではない．主キーの形式の選定は，存在従属クラス図の作成を行うドメイン分析者の技術的選択の範囲である．そして，主キーとして，どちらを選択しても，本提案の自動生成されるユースケースの態様には影響はない.

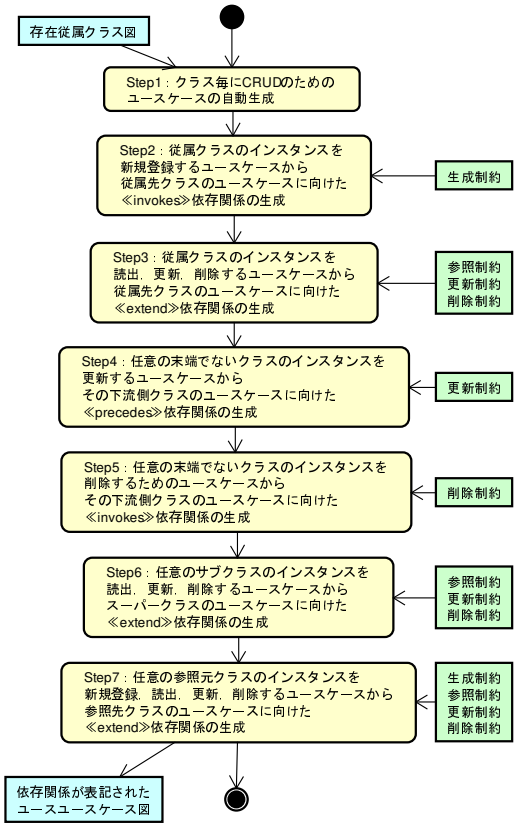

図 5.5: 自動生成アルゴリズムの流れ

クラスに CRUD のうち，どの機能が求められているのかを設定する必要がある．業務システムの機能は，本質的に，ドメインオブジェクトをデータベースに CRUD(Create:登録，Read:参照，Update:更新，Delete:削除) する機能であるためである．具体的には，図 5.6(左側) のように，注文クラスに CRU，注文明細クラスに CRU，商品クラスに CRUD を許可するように設定する [7].

なお，存在従属クラス図では，関連（クラス間のリンク）としては，1) 存在従属関係，2) Is-a 関係，3) 従属関係を持たない単なる参照，の 3 種類のみを考察の対象とする．良く知られた Has-a 関係は，存在従属関係の一種であるが，英語の状態動詞 have に相当するリンクに過ぎず，特別扱いをする必然性を感じられないからである．ただし，Is-a 関係では，インヘリタンス階層をなす最上位のクラスにのみ CRUD を定義する．継承関係の下位階層のクラスは，最上位クラスの CRUD をそのまま引き継いでいるとする．

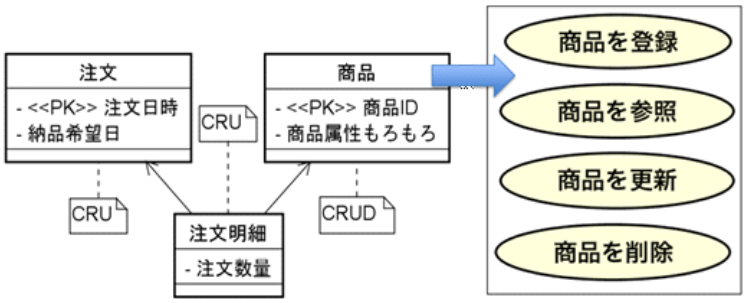

図 5.6: CRUD に応じたクラス図

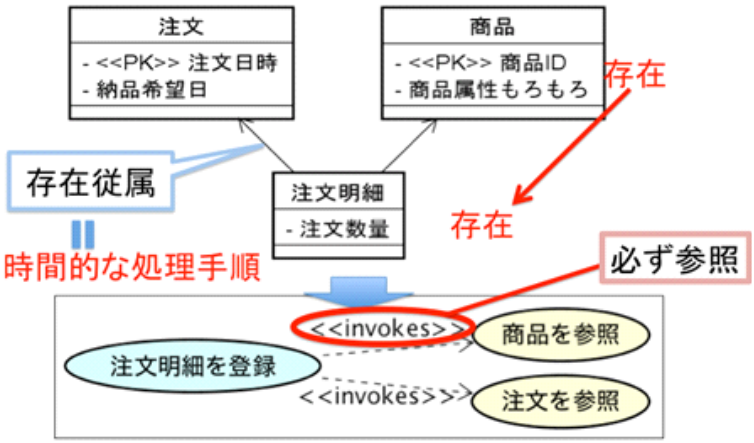

図 5.7: Create に関するユースケース自動生成

(2)　ユースケース自動生成

以下に自動生成アルゴリズムの骨子を示す．以下のアルゴリズムを，対象とする全てのクラスに対して実行する．異なるクラスからスタートした処理において，結果的にあるクラスに同一ユースケースを要求することは有り得るが，この場合，指定されたユースケースを重複して生成しない様に注意する必要がある．各クラスを処理してゆく際のクラスの選択順番は，結果には影響しない．また，CRUD の一部が欠落していた場合には，結果として，当該クラスから自動生成されるユースケースの個数は減少する．

【Step 1】 CRUD の許可設定に応じて，各クラスに対して「当該クラスを登録」「当該クラスを参照」「当該クラスを更新」「当該クラスを削除」のユースケースを自動生成する．

CRUD の設定により，最大 4 種類のユースケースを当該クラスに作成する必要がある．この場合，ユースケース名は,「当該クラス (の名称) を登録/参照/更新/削除す

[7]著者の経験によれば，存在従属ではない参照関係は，あまり多数は出現しない．ほとんどが，存在従属関係のリンクである．

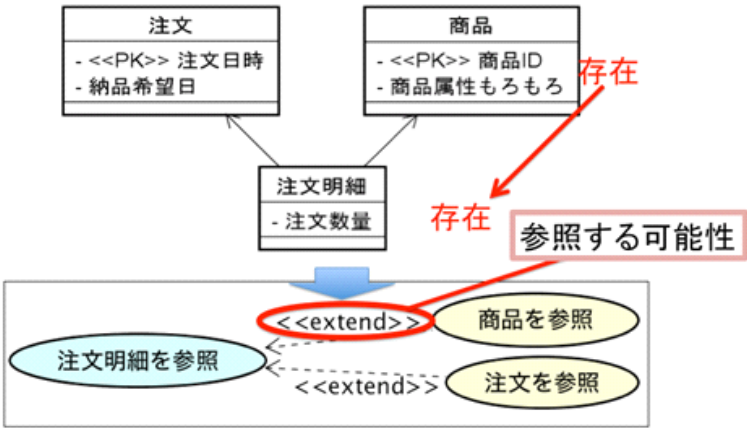

図 5.8: Read に関するユースケース自動生成

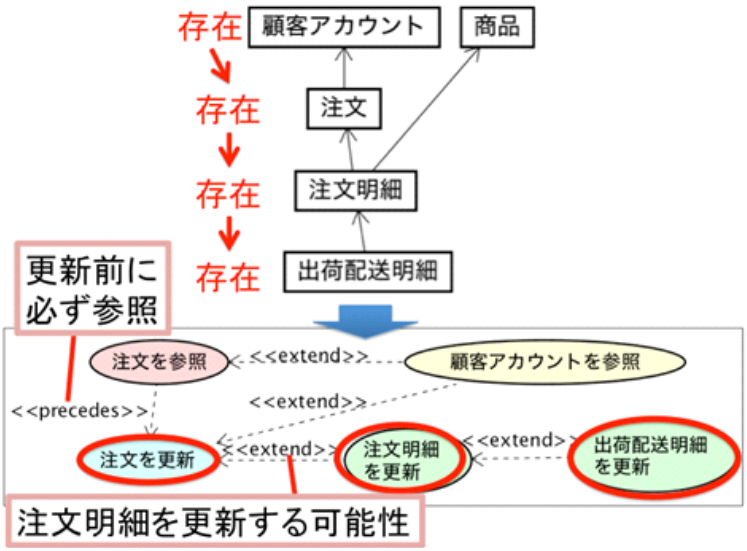

図 5.9: Update に関するユースケース自動生成

る[8]」となる．具体例を図 5.6 の右側に示す．左側の存在従属クラス図では，商品クラスには CRUD を要求するように設定しているため,「商品を登録」「商品を参照」「商品を更新」「商品を削除」それぞれのユースケースを自動的に生成する．注文クラス，注文明細クラスについても同様に，CRU が要求されているため，CRU の各ユースケースを生成する．具体的な，CRUD の各処理に関するユースケースについては，以下のステップで対応する．

【Step 2】　「従属クラスを登録」ユースケースから「従属先のクラスを参照」ユースケースを呼び出す (invokes).

まず最初にクラスの登録（生成）を考える．上流側のクラスを必要とする従属クラスでは，上流側クラスの存在を確認することなく，勝手に生成することはできない．

[8]この場合，ターゲットクラスの名称ではなく,「ロール名」を用いることも考えられる．ロール名のほうが，より分かり易い場合も想定されるが，本章では触れない．

具体例を図 5.7 に示す．このようにユースケースを生成することで，従属クラスを登録する場合，従属先のクラスを必ず参照しなければならない．これは，5.2.1 節で述べた「生成制約」のためである．関連のステレオタイプは≪ invokes ≫とした．

【Step 3】　「従属クラスを参照」「従属クラスを更新」「従属クラスを削除」ユースケースそれぞれに対して,「従属先のクラスを参照」ユースケースを拡張する (extend).
次に，従属クラスを，参照，更新，削除する場合について考える．ただし，この Step 3 で着目しているのは，存在従属の上流側のクラスのみである．下流側クラスについては,「更新」「削除」の場合のみ問題となる．これらは，Step 4，Step 5 で扱う．具体例を図 5.8 に示す．従属クラスを参照・更新・削除する場合，従属先のクラスを参照する可能性がある．つまり，当該クラスを見つけ出すために，上流のクラス（正確にはインスタンス）を探す必要が「場合によっては」生じる．ステレオタイプは，オプションを示す≪ extend ≫を用いることにした．これは，5.2.1 節で述べた【参照制約】のために生じる処理である．

【Step 4】　任意のクラスを更新する場合，そのクラスの下流側に繋がっているクラス全てについて，更新するユースケースを連鎖的に拡張する．
次に，更新について考える．この場合，下流側への影響が心配なので，独立クラス，従属クラスの区別はない．具体例を図 5.9 に示す．これは，5.2.1 節で述べた【更新制約】に必要なユースケース間の依存関係である．存在従属クラス図は DAG(Directed Acyclic Graph) であるため，更新するユースケースの連鎖が永遠に続くことはない．なぜならば，たとえ再帰的な関連であっても，インスタンスレベルで見ればリンク先のインスタンスは異なるためである．DAG とは，クラスが頂点，矢印が方向を示す辺とすると，ある頂点から出発して辺をたどり，頂点には戻ってこないグラフである [9]．ステレオタイプは，≪ precedes ≫（先行する）とした．図 5.9 の precedes の依存関係は，当該クラスを更新する前に，当該クラスを参照するという意味である．これは，ユーザは当該クラスのインスタンスの内容を参照して確認しなければ，当該クラスの更新など出来ない筈だからである．ユーザインタフェースをイメージすると理解できる．

【Step 5】　任意のクラスのインスタンスを削除する場合，そのクラスの下流側に繋がっているクラス全てについて，削除するユースケースを連鎖的に呼び出す (invokes).
独立クラス，従属クラスの別なく，削除の問題である．下流側を無視して削除はできない．具体例を図 5.10 に示す．これは，5.2.1 節で述べた「削除制約」のためである．Step 4 と同様の理由で，ユースケースの連鎖が永遠に続くことはない．また，Step 4 と同様の理由で，≪ precedes ≫の依存関係を引く．

[9] 理論的には DAG であるが，存在従属クラス図を書いた経験では，ツリーの様な形になることも多い．

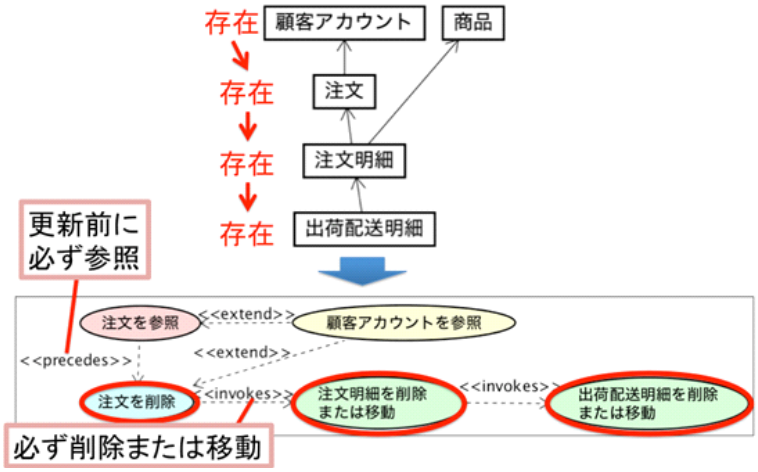

図 5.10: Delete に関するユースケース自動生成

【Step 6】　任意のクラスに対して継承されているクラスがある場合，そのクラスを CRUD するユースケースそれぞれに対して，継承されているクラスを CRUD するユースケースそれぞれを拡張する．

本 Step 6 と次の Step 7 は，クラス間の関係が，一般的な通常の存在従属では無い場合の対応である．本 Step 6 は，図 5.11 の上部にも示した様に，Is-a 関係（汎化関係）の関連が存在従属クラス図に登場するケースへの対応である．具体例を図 5.11 に示す．これは，任意のクラスに対して継承されているクラスがある場合，当該クラスを登録するとき，当該クラスの属性に加えて，被継承クラスの属性を登録する必要が生じるためである．そのとき，複数のクラスに継承されている場合，どの被継承クラスの属性を登録するか分からないため，拡張ユースケースとしている．参照，更新，削除についても同様の理由で，例に倣ってユースケースを生成する．

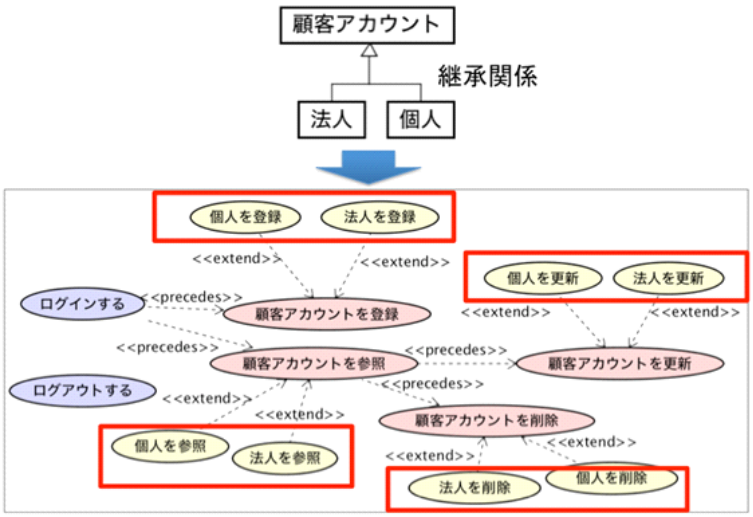

図 5.11: 継承関係に関するユースケース自動生成

【Step 7】　任意のクラスが参照しているクラスがある場合，任意のクラスを CRUD するユースケースそれぞれに対して，参照しているクラスを Read するユースケースを拡張する．

存在従属ではない，単なる関連（参照関係）が 2 つのクラス間に存在する場合である．具体例を図 5.12 に示す．当該クラスと参照関係にあるということは，当該クラスから参照先クラスをこのリンクを用いて，ポインタによるアクセスの様に，参照したいためである．そのため，当該クラスを登録・参照・更新・削除するときに，ユーザが任意で参照先クラスを参照できるように，拡張ユースケースとした．当該クラスと参照関係にあるということは，存在従属性とは異なり，当該クラスのインスタンスは，参照先クラスのインスタンスが存在しなくても存在できることであり，その場合，参照先には NULL が設定される．

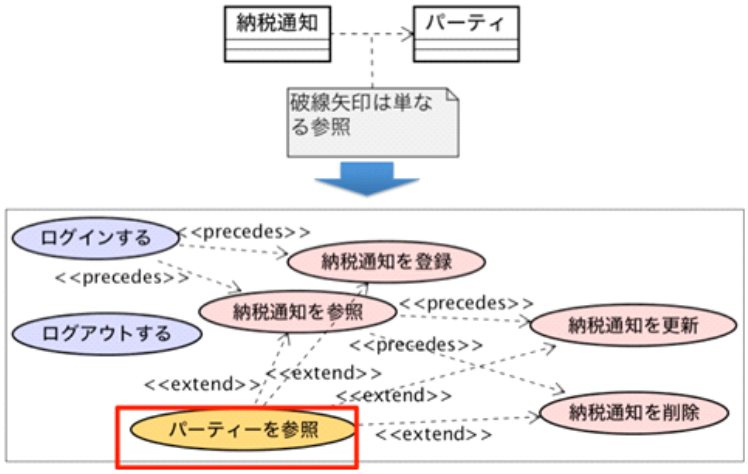

図 5.12: 単なる参照関係に関するユースケース自動生成

(3)　具体例による説明

アルゴリズムの Step 1~Step 5 について，簡単な例で示す．図 5.13 は，左側に書かれた存在従属クラス図のクラス C をターゲットとして，アルゴリズムの適用イメージである．

まず，最初に，Step 1 では,「CRUD」に対する 4 個のユースケース「C クラスを登録」「C クラスを参照」「C クラスを更新」「C クラスを削除」が生成される．

次に Step 2 では，この中の「C クラスを登録」から 2 つのユースケース「A クラスを参照」「B クラスを参照」に対して≪ invokes ≫で接続する．このとき，これら 2 つのユースケースが未登録なら登録する．

次に，Step 3 では,「C クラスを参照」「C クラスを更新」「C クラスを削除」の 3 つのユースケース（Step 1 で作成済）に対して，上流側である A クラスのユースケース「A クラスを参照」「B クラスを参照」へ≪ extend ≫で接続する．この 2 つのユースケースは，既に，Step 2 で存在は確認されている．

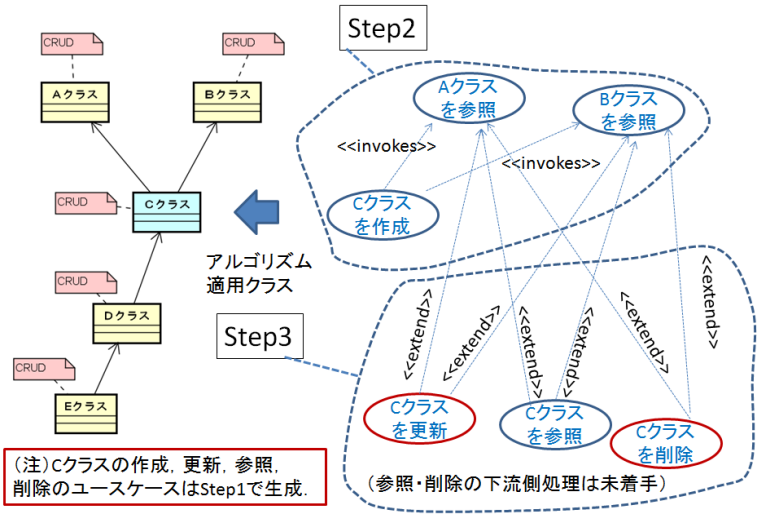

図 5.13: アルゴリズム適用例 (Step 2, Step 3)

次に図 5.14 に移る．Step 4 では,「U(Update)」に対する処理として，念のため，下流側の更新が無いかどうかを確認する．この確認作業は，下流側のすべてのクラスに対して,「U」の処理をアルゴリズムに再帰的に要求することで実現される．同様に，削除である Step 5 でも,「D(Delete)」に対する処理として，下流側全体の削除を行うユースケースを準備する.「D」の処理をアルゴリズムに再帰的に要求する．

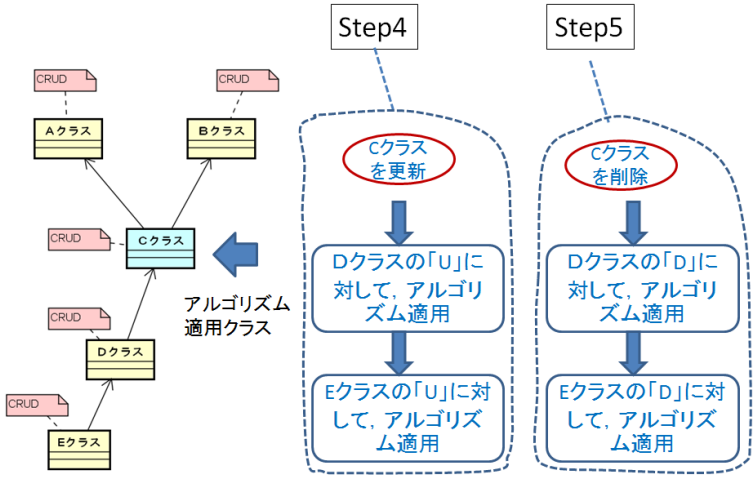

図 5.14: アルゴリズム適用例 (Step 4,Step 5))

5.4　評価実験

5.4.1　実験内容

提案したアルゴリズムを Java で実装し，存在従属クラス図からユースケース図を自動生成するツールのプロトタイプを開発した．具体的には，モデリングツール astah*

professional [57] のプラグインとして実装した．ユースケース図の描画は自動化されている．ただし，生成されたユースケース図は，astah*の画面の位置までは指定しない．XML ファイルでユースケース図を受け取った後，見やすい配置とするには，手作業が現状では必要である．図 5.15 に，実験システムのイメージを示す．

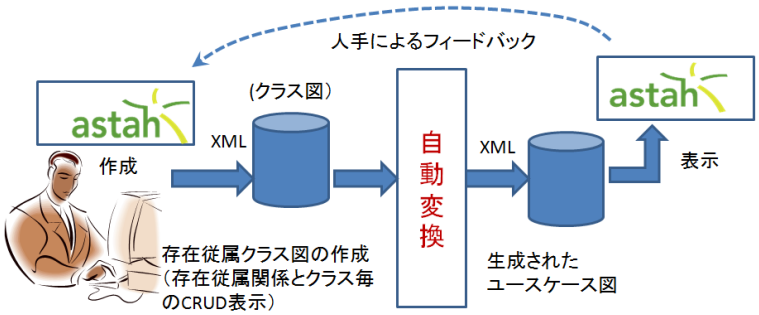

図 5.15: 評価システムの構成

本ツールのプロトタイプを用いて，実在の某地方自治体の住宅管理システムの調達仕様書から存在従属クラス図の作成を行った．また，存在従属クラス図の全てのクラスについて CRUD が要求されていると設定し，存在従属クラス図からユースケース図を自動生成した．前述の住宅管理システムの調達仕様書に対して提案手法を適用し，自動生成したユースケースと，調達仕様書の機能要件を比較して，両者が一致するかどうかを調べた．自動生成したユースケースの比較対象は，調達仕様書の機能要件から手作業で作成したユースケースとした．ただし，このユースケースは，機能要件から直接導くことができるものに限定し，推測により間接的に導かれるものは除外した．

5.4.2　実験結果

ユースケース図が正しく自動生成されたため，アルゴリズムが正しく動いていることが分かった．総クラス数は 86 個，独立クラスは 26 個，従属クラスは 60 個となった．ユースケースの個数 (同一のユースケース名を含む) は，813 個となった．存在従属クラスの一部として，滞納ドメインのクラス図を図 5.16 に示す．更に，図 5.16 から自動生成したユースケース図を図 5.17 に示す．

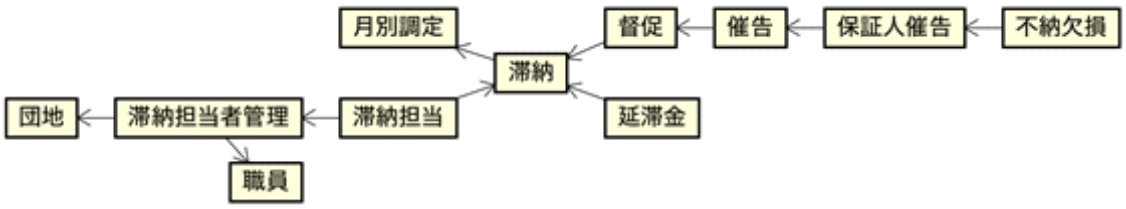

図 5.16: 滞納ドメインの存在従属クラス図

図 5.17: 滞納クラスから自動生成したユースケース図

ユースケースの個数 (同一のユースケース名を含む) を比較した結果を図 5.18 に示す．自動生成されたユースケース 813 個のうち，自治体の調達仕様書から導出できたのは約 2 割に過ぎなかったことが分かる．

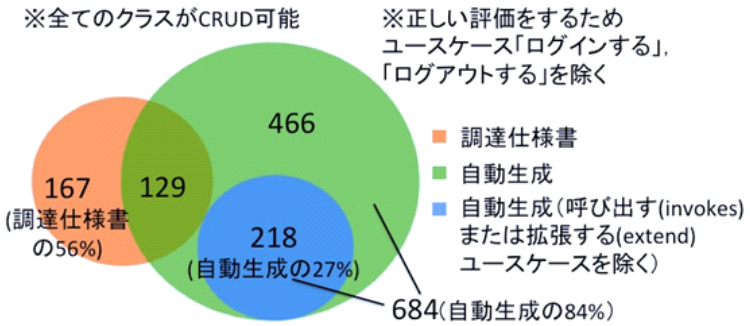

図 5.18: ユースケースの個数

これは，調達仕様書には，提案アルゴリズムで生成されるように，呼び出す (invokes) または拡張する (extend) ユースケースが 1 個のみしか無かったためである．

上記の状況は，考えてみると当然である．例えば，団地のある部屋の情報を読みだす際 (R) には，当然，複数の団地及び棟の一覧を参照し，その中から棟を選択する様な処理が必要となる．しかし，調達仕様書では,「部屋の情報を参照する」といった形で書かれて，その前提として，棟を選択するような機能は指示されないからである．提案手法は，そのようなオプションであっても可能性のあるリンクの探索をすべて含んで網羅的にユースケース図を自動生成する．しかも，ユースケース間のステレオタイプ付きの依存関係を自動的に付けている点でアドバンテージがある．

自動生成したユースケースの中から，呼び出す (invokes) または拡張する (extend) ユー

スケースを除いた場合を計算した．この場合でも，調達仕様書には，自動生成されたユースケース 347 個のうち，自治体の調達仕様書から手作業で導出できたのは約 4 割であった．この大きな原因は，調達仕様書では，CRUD を全部考えない又は一部しか考えないクラスがあるためである．CRUD を全部は必要とはしないクラスは 22 個，一部しか考えないクラスは 56 個であったのに対して，CRUD を全部考えるクラスは 8 個に過ぎなかった．存在従属クラス図の CRUD を，現実の調達仕様書に則して修正すれば，カバー率は格段に向上すると思われる．

逆に自動生成では生成できなかったユースケースは 167 件あった．図 5.18 に示す通り，それらは調達仕様書の 56%である．これらをカテゴリに分類した結果を表 5.2 に示す．この結果の分析は次節の考察にて示す．

表 5.2: 自動生成できなかったユースケースのカテゴリ

ユースケースのカテゴリ	サブカテゴリ数	ユースケース数
印刷メニュー	4	9
書類・帳票の作成	1	4
業務メニュー	1	1
シミュレーション系（家賃，住戸移動）	2	4
バッチ計算系	7	24
バッチ判定系	7	8
バッチ参照系	7	34
特定の項目で抽出して削除	1	9
バッチ削除	1	1
特定の項目のみを更新	4	34
特定の種類として登録，データのコピーなど	7	39
	合計	167

5.5　関連研究と考察

5.5.1　提案手法の特徴

本提案のユースケース図自動生成アルゴリズムでは，インプットされる情報として，存在従属クラス図を用いている点に最大の特徴がある．存在従属クラス図は，一般的な UML のクラス図と異なり，クラスのライフタイムの相互関係の情報（どちらが先に生成されるかとの業務処理のシーケンス情報）を,「存在従属」という，追加情報として多少なりとも持っている．結果として，生成されるユースケース図では，ユースケース間の依存関係のステレオタイプとして,「invokes(必ず参照しなければならない．存在従属の上流側インスタンスの確認が必要)」「extend（存在従属下流側を見に行く処理なので，必須ではなく，オプション機能として表現)」「precedes（上流側を先にみないことには，更新・

削除もできないので先行)」等の種別を設けることが可能となった．これにより，より厳密なユースケース図を生成しようとするものであり，本提案手法のオリジナリティがここにある．既存の存在従属ではないクラス図から自動生成する場合には，この様なきめの細かいステレオタイプを付加することはできない．

存在従属は，ER 図の提案者である Chen の提案である．存在従属は，何故か，UML には採用されず，広く知られているわけではない．存在従属クラス図が，従来のクラス図（UML）と大きく異なる点は,「『関連』でリンクされた 2 つのオブジェクト間で，一方のライフタイムが他方のライフタイムを包含する場合に，誘導可能性のある関連を定義し，ライフタイム上で包含している側のオブジェクトに厳密に多重度が 1 の役割を付加する」点のみである．この結果，存在従属クラス図は，オブジェクトの処理（CRUD）を実現する順序の上での制約を表す情報を持つものとなり，従来のクラス図とは大きく異なるものとなる．なお，与えられた仕様記述からトップダウンに概念レベルのクラス図を作成する従来手法の枠組みと，存在従属クラス図を用いて概念レベルの分析を行うプロセスには，基本的な差異はないと考える．

実際にクラス図を描くと，ほとんどが従属クラスとなる．独立クラスと従属クラスの識別に困難性は感じない．ただし，業務知識の不足から，存在従属クラス図自体が誤りを含むことは多々起こり得る．その場合には，本提案のアルゴリズムは，誤った処理手順を持つユースケース図を生成する．また，クラス図を，提案アルゴリズムが自動的に正規化してくれるわけではない．自動生成されたユースケース図を眺めて，誤った箇所が指摘できるか否かは，大変，興味ある部分であるが，本研究では評価できていない．今後の課題としたい．

なお，提案手法は，通常のユースケース駆動開発手法が採用している「要求仕様からユースケースを作成し，その後にクラス図を作成する」のではなく,「要求分析段階で永続化されるべきデータに対してクラス図を描くなら，ここから自動生成でユースケース図を作れる」ことを主張している．通常と順番が逆であり，この手法で，すべてのユースケースを作成できるはずもない．しかし，現実のユースケースの中には，かなりの割合で，自動生成できるユースケースが含まれている可能性があるのではないかとするのが著者の問題意識である．もし，そうであるなら，本章で提案したアルゴリズムは，既存の RUP(ラショナル統一プロセス) 等と併用して，ソフトウェア開発の支援ツールとして活用できる可能性があると考えている．

5.5.2　ユースケース自動生成のメリット

提案手法の優位性は，自動生成されるユースケースおよび，生成されたユースケース間の依存関係が網羅的であり，かつユースケース名の命名が規則的である点にある．すなわち，業務で捕捉，管理，蓄積すべきエンティティのインスタンスの存在期間の前後

関係 (時間的な制約関係) を描写した存在従属クラス図をインプットとして，各々のエンティティに設定された CRUD に従って漏れなく，愚直に生成する．

生成すべきユースケースは，存在従属クラス図上に登場するそれぞれのエンティティのインスタンスを CRUD するためのユースケース群であるため，ユースケースの数はエンティティ数の 4 倍（これを N とする）になる．そして，ユースケース間の依存関係は理論的には $(N(N-1)/2) \times 3$ の依存関係が生成される．3 倍しているのは，提案手法で生成される依存関係の種類は 3 種類あるためである．

例えば，存在従属クラス図上にたった 5 つのクラスしかなかったとしても，最大 20 のユースケースと 30 の依存関係が生成される可能性がある．これは既に手作業で正確に扱うのは困難である．また，ユースケース名の命名も提案手法であれば規則的にできる．提案手法では,「<クラス名>を登録する」,「<クラス名>を参照する」,「<クラス名>を更新する」,「<クラス名>を削除する」という名称に統一される．

この様なことは，手作業では困難である．たとえば，注文を登録するユースケース 1 つを採って見ても,「注文を新規登録する」,「注文を登録する」,「注文の新規登録」,「注文の登録」など，実に多様なユースケース名が命名されることが珍しくない．

上の点からも，ユースケース自動生成のメリットは大きいと考えている．

5.5.3　関連研究

OOSE(Object-Oriented Software Engineering)[36] や Rational Unified Process[58] の手法も，ICONIX[51] と同様にユースケース駆動開発手法である．

Deeptimahanti ら [59] は，機能要件に書いてある文それぞれを，主語と動詞句に分けて，主語をアクター，動詞句をユースケースに対応させて，ユースケースの自動生成を行っている．中鉢ら [60] は，顧客の業務を自然言語によるシナリオでモデル化し，ここからソフトウェアの機能要求を体系的にユースケース分析できる SBVA(Scenario-Based Visual Analysis) 法を提案している．

機能要件について，これらの手法はすべて，機能要件が網羅的である保証はない問題がある．また，ユースケースの粒度については，Deeptimahanti らの研究 [59] は，ユースケース記述の一文ごとにユースケースを区切ってしまう可能性があるため，粒度が細かすぎる問題がある．中鉢らの研究 [60] では，分析手順が体系的であるため，担当者の主観に任されない利点がある．提案手法は，担当者の主観に依存しないことに加えて，管理対象に対して忠実にユースケースが導けることに優位性がある．

5.5.4　考察

最後に，表 5.2 に示した，本提案システムがカバーできなかったユースケース群について分析する．これらのユースケースを，本提案システムが，カバーできなかった背景には，幾つかの理由が考えられる．一つの問題として，現実の業務システムにおける業務支援機能では，ユーザ登録等の管理機能，1 週間，1 カ月単位の集計表等の，特定のクラスに属するインスタンスへの CRUD 処理を束ねて統計値を算出したものがある．しかし，考えてみると，これらがビューであるなら，昔とは異なり，CPU も高速化して，データベースもインコアにできる時代に，集計の違いに合わせて，蓄一，モジュールを設計する必要性に疑問を感じる．

一方で，ユーザの使いやすさを考慮したユースケースが不足していた．例えば,「家族データのある住戸から別住戸への移動」などは，2 つの処理をつないだマクロ的な機能である．本来は，個別の機能を呼び出す様なマクロ定義や単一のビューモジュールで対処するべきものなのではないだろうか．特定のデータ項目のみを複数案件で一括処理する機能にも，マクロ的意味を感じる．

上記の分析から,「業務システムでは，ユーザから業務のシーケンスを聞きだして，システム化するべき」との従来のアプローチに対しても，本章は，ひとつの見方を提示している．つまり，もともと，データが決まれば具備するべき機能も，処理手順も「データから自動的に決まる」部分がかなりあるのではないかとの問題意識である．ユースケースの粒度についても，本章の手法では，CRUD 単位に分割しているため，客観的かつ明確である．そして，何より，ユーザ利便性を提供する画面の多くも，この単位機能をマクロ的につなぎ合わせればできるのでは（当然，全部ではないとはおもわれるが）との印象も持つ．少なくとも，データから決まって来る処理については，もれなく提案方法は提示できる．存在従属性を示したクラス図に基づくユースケース自動生成は，業務システムの機能要件の洗いだしに有効と考える．

最後に，検討を要する問題に,「ユースケースではなく，クラス図を先に書けば，本当に仕様の網羅性をカバーできるのか」との問いがある．本提案手法は，クラス図からユーケースのかなりの部分が自動的に生成できることに一つの意義がある．従来のユースケースを作り，その後にクラス図を作る方法では，自動化のステップは含まれていないからである．しかし，仕様の網羅性が限られる時，そこから生成されるクラス図も，不備なものとなり，結果的に，ユースケースの網羅性が保証できない恐れは可能性としては存在する．著者は，厳密な記述を要求されて，本質のみを取り出しているクラス図の方が，対象ビジネスの論理構造に迫れると考えているが，科学的に証明されたものではない．今後の課題としたい．

5.6　結言

本章では，先ず，機能要件から「存在従属性と CRUD 表記を併記したクラス図」を作成し，次に，その存在従属クラス図から，ユースケースを自動生成する手法を提案した．

現在，ソフトウェア開発手法の主流となっている，ユースケースからクラス図を作成する「ユースケース駆動」とは逆のアプローチである．本提案手法では，クラス図からユースケースへの変換は，自動生成が可能である．機能要件の洗い出しに際しての，迅速化が期待される．なお，生成されるユースケースは，クラスの名称に，動詞「登録する」「参照する」「更新する」「削除する」をそれぞれ，CRUD に応じて付与したものである．このため，ユースケースの粒度は統一されており曖昧性はない．具体的には，クラス図からユースケース（図）を生成するアルゴリズムを提案し，astah*のプラグインとして，Java で実装した．

実装した自動変換ツールを用いて，某自治体の住宅管理システムの調達仕様書に提案手法を適用した．そして，自動生成されたユースケースと自治体の住宅管理システムの調達仕様書との間で，機能要件（ユースケース）が一致するか否かを調査した．その結果，存在従属性クラス図（全クラスに CRUD 属性を付与）に基づくユースケース 813 個中，仕様書から手動で導けたのは，約 2 割に過ぎなかった．差異の原因は種々あると思われるが，一つの原因は，現実のクラスについては，CRUD が必ずしも常に 4 種類付与されてはいないことが挙げられる．また，文書化された調達仕様書は，特徴的な事のみ記述しており，全機能を網羅している訳ではないと考えられる．

自動生成ユースケース図は，ステレオタイプを含めて，ユースケース間の依存関係を自動生成する．これは，提案方式の大きなメリットである．一方，人手によるユースケース作成では，ユースケース間の依存関係や依存関係のタイプは，別途生成する必要がある．そうしないとユースケース図は描けない．今回の評価では，人手部分の依存関係とステレオタイプ生成は行っていない．自動生成のユースケースは，ある側面のユースケースに限られるが，依存関係とそのステレオタイプの自動生成も可能である．業務システムの大きな目的が，管理すべきデータの CRUD を行うことにあるとするなら，本提案の手法は，業務システムの機能要件の洗いだしと妥当性検証に有効と思われる．

一方，自動生成できなかったユースケースが約 5 割あった．それらをカテゴリに分類した結果，ユーザの使いやすさを考慮したユースケースが不足していたと分かった．今回得られたカテゴリ分類を今後に活かして，提案手法の改良を如何に進めるかが今後の課題である．ただし，実際の機能要件を見ていると,「家族のある住宅から別の住宅への移動」,「一週間の当該事務所の処理一覧」と言った，マクロやビューに類する機能も多い．逆に言えば,「業務フロー中には，データの CRUD，マクロ機能，ビュー機能といった種別が存在する」事に，改めて気付いたと言うのが，本研究の正直な印象である．現実の業務フローを分析する際には，闇雲に現場担当者の業務フローを写し取るのではなく，こ

の様なカテゴリー種別を意識して，本提案の自動生成に任せ得る部分と，そうでない部分を分けて設計するようなアプローチも検討すべきと思われる．今度の課題としたい．

なお，本提案のアルゴリズムを用いると，ソフトウェア開発過程で,（存在従属）クラス図を修正しても，毎回，ユースケース図を自動生成できる．これは，明らかに，アジャイル向きである．ただし，現状のプロトタイプシステムは，Astah*のプラグインとして動作しているが，生成されたユースケース図を人間に見やすく自動再配置することができていない．この問題を解決できれば，アジャイル開発手法にも適した手法となる可能性がある．

第6章　ドメインモデルからの機能規模測定

6.1　緒言

業務システム[1] 開発の現場において，開発対象の機能規模を簡単かつ正確に測定することへの期待が高まっている．なぜならば，これによって開発に要する工数をあらかじめ見積ることができ，ユースケース候補を費用対効果を考慮しながら取捨選択できるようになるからである．そうなれば，利用者と開発者の間で納得感のある開発契約を締結することができるようになる．さらに開発後にも機能的規模を基準にプロジェクトの生産性や欠陥密度などを反省することもでき，以降の開発をより工学的ものに改良していく判断材料にもなる．さらに，近年では，スコープを細分化して，小さなリリースを繰り返すスタイルが広まってきたため，見積りの頻度も高くなる傾向にある [61].

ソフトウェアの規模をどのように測るかについては予てから種々の議論が展開されている領域であり，いくつもの測定手法が提案されている．ソースコードの行数で規模を測定するのは確実な方法であるが，実装が終わってからでないと測定できないため，見積りの用途には使えない.

一方，あらかじめモデルを作成してソフトウェアの規模を測る方法としては，ファンクション・ポイント（IFPUG：International Function Point Users Group）法 [62]，COSMIC（COmmon Software Measurement International Consortium）法 [63] などが知られている．中でもCOSMIC法はユースケース・モデルと親和性が高く，ISOで承認されJIS標準にもなっている測定手法であるため，著者は機能規模を測定する必要がある時にはCOSMIC法を用いている.

COSMIC法は認知された測定手法であるが，その精度を保つためには，イベントフローレベルのユースケースの論理的な実現の分析を必要とする．しかしながら，実用的な観点からはもっと早期に，すなわち，ユースケース候補の中から開発対象をユースケースとして決定する時点で機能規模に関する情報が得られることが望まれる.

そこで，本節では，COSMIC法をベースに，業務アプリケーション専用としてカスタマイズした測定手法を提案する．カスタマイズには，その業務ドメインで扱うエンティ

[1]取引等に関する事実を効率的に捕捉，蓄積，管理するためのアプリケーションと緩やかに定義する．その特性上，データベースを核としたアプリケーションである.

ティの存在従属性に着目した．なぜならば，存在従属性はインスタンスのライフサイクルに基づく関係であるため，それらを扱う機能の時間的前後関係も半ば規定するからである．そのことを自然な形で取り入れた提案手法は，要求定義の早い段階で適用でき，測定も簡単かつ正確であり，その作業成果物も機能的規模の測定の用途だけでなく，機能要件の網羅性を確認にも優れたものである．

以下，6.2 節では，COSMIC 法の概要と課題を説明する．6.3 節では提案手法について説明する．6.4 節では，検証実験として COSMIC 法と提案手法の結果を比較する．6.5 節は本章の結言である．

6.2　COSMIC 法

6.2.1　COSMIC 法の概要

COSMIC 法はソフトウェアの機能規模を測定する手法である．1997 年に Full Function Point（FFP）法（ver1.0）という名前でリアルタイム，技術，システムソフトウェアの機能規模を測定する手法として提案された．その後 2002 年に ISO で承認され，2006 年には JIS X0143 として JIS 標準となった手法である [63]．

6.2.2　COSMIC 法の測定要素

COSMIC 法では，図 6.1 に示すように測定対象ソフトウェアについて，システム境界をまたがったデータ移動数と永続化記憶域に読み書きするデータ移動数の合計値を求めることで利用者の立場から見た機能の規模である機能規模を測定する [64]．

システム境界の外側は対等システム（Peer System/Peer Component）と呼ばれる．また，永続化記憶域はファイルやデータベースなどが対応する．ただし，データ移動数といってもその回数をカウントするのではなく，その種類数を数えることに注意しなければならない．読み書きするインスタンスの件数が増えても，ソフトウェアの行数が増えることはないためである．

測定するデータ移動の種類は表 6.1 に示す 4 種類である．移動するのは，データ・グループと呼ばれるデータのまとまりで，論理データモデルのエンティティに相当する．データ移動の対象となるデータ・グループの種類数を機能プロセスと呼ばれる単位毎に数えていくことによって，最終的にソフトウェア全体の機能規模を求めることができる．

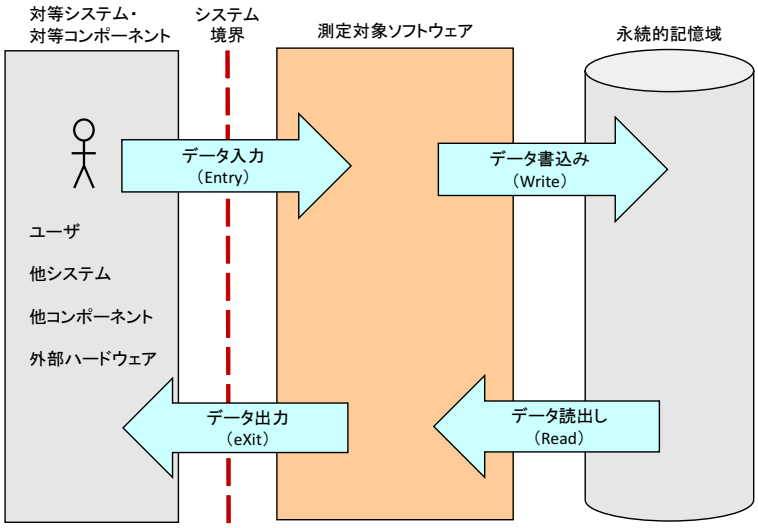

図 6.1: COSMIC 法の要素間のデータ移動モデル

表 6.1: COSMIC 法におけるデータ移動の種類

データ移動の種類	説明
エントリ （Entry）	システム境界外からユーザインタフェース(UI)や通信を通じて入力されることに対応するデータ移動
エクジット （eXit ）	UI や通信によりシステム境界外に出力されることに対応するデータ移動
リード （Read）	永続化記憶域の注目オブジェクト(エンティティ)から読み出されるデータ移動
ライト （Write）	永続化記憶域の注目オブジェクトに対して書き込まれるデータ移動

6.2.3　COSMIC 法の測定手順

COSMIC 法の測定手順は以下の通りである [64].

【Step 1】システムの境界と利用者機能要求を定義する

COSMIC 法では，測定対象ソフトウェアとデータの授受を行うシステム境界の外を対等システムと呼ぶ．対等システムは利用者であったり，他システムやハードウエアであってもよいため，丁度 UML のアクターに対応する概念である．そして，利用者の立場から見たソフトウェアの機能のまとまりを利用者機能要件（Functional User Requirement）と呼ぶ．利用者機能要件は 1 つ以上の機能プロセスで構成され，UML のユースケースにほぼ相当する．このため，COSMIC 法はユースケースモデルと親和性が高い方法であるといえるだろう．

【Step 2】利用者機能要求を構成する機能プロセスを明らかにする

機能プロセスとは，1 つのエントリをトリガとして実行される機能のまとまりである．たとえば，システム境界外からの顧客 ID のエントリをトリガとして，顧客エンティティを読み出し，その諸属性をエクジットする機能のまとまりは機能プロセスの例である．これは，ちょうどユースケース記述としてのイベントフローにおける，アクターの入力から，それに対するシステムの応答という対話の一往復に相当すると考えられる．ここで，顧客 ID の入力を 1 エントリ，顧客エンティティの読み出しを 1 リード，顧客の諸属性の表示を 1 エクジットとしてカウントするため，この機能プロセスは 3 種類のデータ・グループのデータ移動が観測される．したがって，データ移動に基づく機能規模値は 3CFP ということになる．なお，CFP（COSMIC Function Point）は測定される機能規模の単位である．

【Step 3】利用者機能要求毎の機能規模とソフトウェアの機能規模を求める

利用者機能要求毎の機能規模は，利用者機能要求を構成するすべての機能プロセスの機能規模（CFP 値）を合計することで求められる．さらに，ソフトウェアの機能規模は，ソフトウェアが提供するすべての利用者機能要求の機能規模を合計することで求められる．

6.2.4　COSMIC 法適用の課題

著者は，COSMIC 法を適用するための課題は 2 つあると考えている．

（ア）要求定義プロセス上の測定時期に関する課題

１つ目の課題は，機能規模を測定可能な要求定義プロセス上の時点に関するものである．図 6.2 は，著者が検討している要求定義プロセスの部分である．業務システム開発のための要求定義プロセスであり，要求工学実践ガイド [65] を参照しながらまとめつつある．分析の早い時期にエンティティの存在従属クラス図を作成して，ドメインについての理解を深めつつ，後続の作業で積極的に活用する意図がある．

著者は，機能規模の測定結果を図 6.2 中の（X）の作業で使いたいと望んでいる．なぜならば，(X）の作業で，費用感も考慮しながら今回の開発対象をユースケースとして決定したいからである．そのためには遅くとも（A）の作業時点で機能規模の測定が必要である．しかしながら，COSMIC 法で正確な測定を行うためには，図 6.2 中の（B）の作業時点まで待たねばならない．前述の通り COSMIC 法では，アクターの入力からシステムの応答までを 1 つの機能プロセスとし，その中で行われるデータ移動を計測するためである．

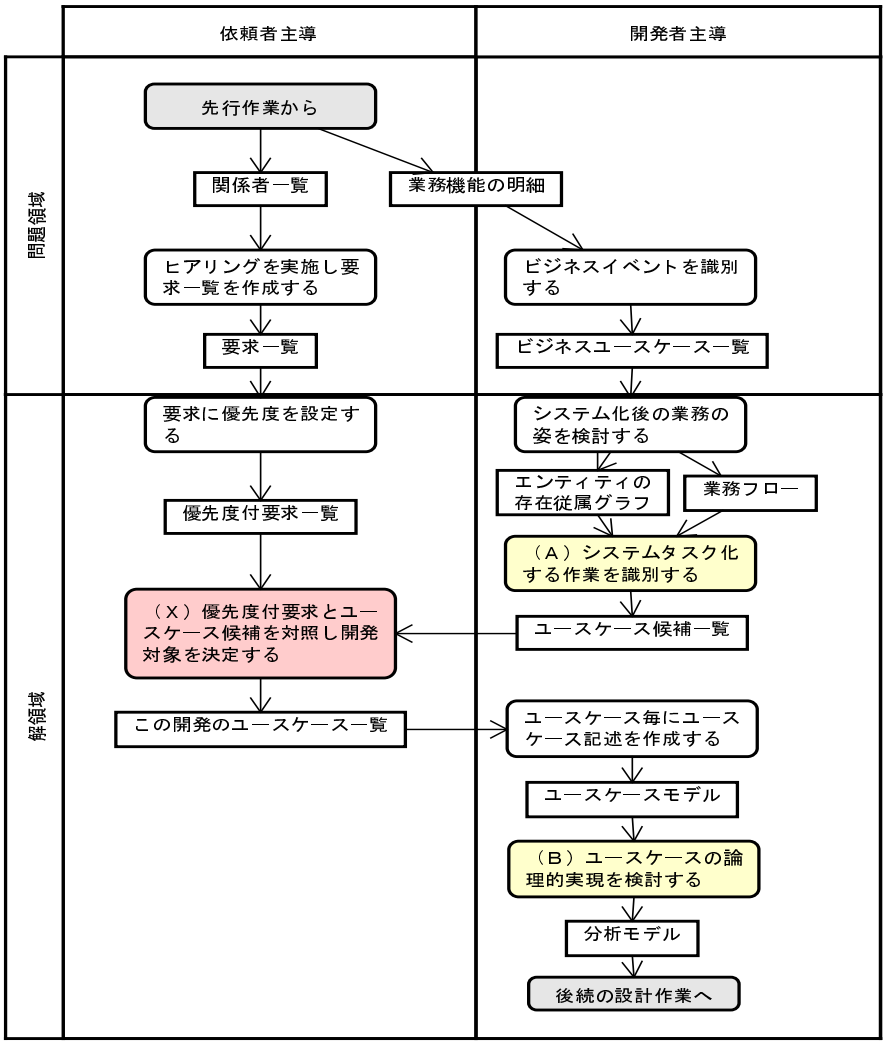

図 6.2: 著者の要求定義プロセス

（イ）要求記述の特性に関する課題

2 つ目の課題は，要求記述の特性に関するものである．表 6.2 は，要求記述の例として一般的なものの一つである．一見，ビジネスイベントによって起動される業務プロセスに即して記述されているが，COSMIC 法が要求する機能プロセスの観点からは，至って粒度が粗く，網羅性も低いものである．

たとえば，受注プロセスに関する記述だけをとってみても，顧客を確認する場面において，顧客が登録されていない場合の対処プロセスについては言及されていない．商品についても同様である．このことから，要求記述は，何を管理するかについてはある程度言及されているものの，どのように管理を実現するかについては捨象されていると考えらえる．そのため，これらの課題を克服するべく，著者は図 6.2 中の（A）の作業時点でも正確な機能規模が測定できるように，COSMIC 法のカスタマイズを検討することにした．

表 6.2: 受注から出荷までの要求記述の例

システム化を検討している業務プロセス
受注プロセスは，顧客から注文が舞い込むと開始される．担当者は，顧客と商品を確認して，注文の内容を受注台帳に書込む．
出荷指示プロセスは，毎日15時になると開始される．担当者は受注台帳から未出荷の受注データを抽出しては，出荷指示済にステータスを更新し，出荷指示書を起票して出荷プロセスに渡す．このとき，出荷コストを抑えるため同一顧客への出荷は複数の受注明細をまとめて行うようにする．
出荷プロセスは，出荷指示書を受取ると開始される．担当者は，出荷指示書に記載された商品をピッキングし，納品書を作成・同封して梱包し，宛先ラベルを作成・貼付して宅配業者に引渡すとともに，出荷指示書のステータスを出荷済に更新して，出荷実績登録プロセスに渡す．ただし，システム化後も商品のピッキングは担当者が手作業で行う．
出荷実績登録プロセスは，出荷済の出荷指示書を受取ると開始される．担当者は，受注台帳の当該受注データのステータスを出荷済に更新するとともに，在庫台帳の当該商品の残高を更新する．

6.3　存在従属クラス図に基づく測定手法

6.3.1　提案手法の概要

これまで見てきた通り，COSMIC 法が要求する入力モデルは，ユースケース記述としてのイベントフローレベル以上の詳細さを有する必要がある．すなわち，測定のためにはアクターとシステムの対話の詳細がモデル化されている必要がある．しかし，これでは著者が望むタイミングで機能規模の見積りを得ることはできない．そこで，提案手法では，ユースケース・モデルではなく，業務についての要求記述から存在従属クラス図を先に作成し，それを測定のための入力とする．説明を具体的に進めるために，引き続き表 6.2 に示した業務を例題として用いる．留意すべきは，要求記述は，あくまでも作成者が業務の一部分だけをプロファイルして描写した結果であるため，ストレートフォワードに機能規模測定のための機能プロセスに展開することはできないことである．

6.3.2　提案手法適用の手順

図 6.3 に提案手法適用の手順のフローを示す．以下では，フロー上の各処理ステップについて説明する．

【Step 1】エンティティの存在従属クラス図を作成する

エンティティとは，業務で管理すべき重要な概念である．エンティティの存在従属クラス図は，主にエンティティ間の存在従属性に着目して作成する有向グラフである．存在従属性の概念は P．チェンが用いている [30][25]．あるオブジェクトが，別のオブジェクトの先立つ存在を前提として存在し得るとき，前者のオブジェクトは後者のオブジェクトに存在従属するという．本節では，文献 [26] の呼称に倣って，前者のオブジェクトを

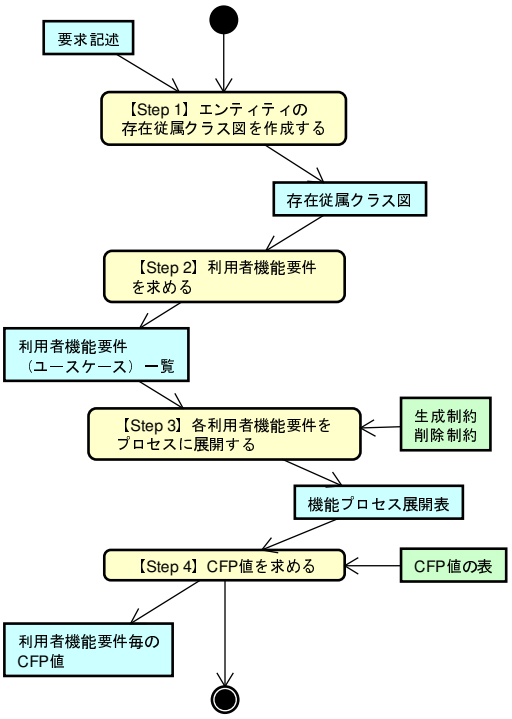

図 6.3: 提案手法適用の手順の流れ

従属クラス（dependent class）のオブジェクト，後者のオブジェクトを親クラス（parent class）のオブジェクトと呼ぶ．この存在従属という概念を用いることによって，たとえば，オークションサイトの入札は，出品に存在従属する．月々のローンの返済は，過去の購買という事象に存在従属する，などと表現できる．存在従属クラス図は UML クラス図の記法を流用しているため，静的な構造図に見えるが，実際はエンティティのインスタンスのライフサイクルに関する依存関係を表現するので著者は動的モデルとしても位置づけている．日本語要求記述文からのエンティティの識別については文献 [4] を，存在従属という概念，および存在従属クラス図作成の詳細については，文献 [52] を，それぞれ参照されたいが，ここではその概略を紹介する．

文献 [4] では，存在文が頻出する日本語要求記述を英文に近い行為文単文（動詞が一つの文）に変換してからクラス図に表記すべき要素を識別する．表 6.3 は，その際に用いる記述中の品詞とクラス図の要素の対応づけの指針を示している．そして，クラスとする段階で帳票や台帳の名称は，たとえば,「受注台帳」は「受注」のように概念としての名称に置き換えたり,「納品書」のように「出荷」のビューにしかすぎないと判断されるものは除外したりする．

さて，クラス図に表記すべき要素が求まると，次にそれらの間の関係を検討していく．表 6.4 は，識別されたクラス図の要素を存在従属クラス図に組み立てる際のガイドラインを示している．文献 [52] によれば，先に単独で存在可能なクラス群を網羅してから，それらに従属，あるいはそれらを参照するクラスまたは属性を UML クラス図本来のモデル

表 6.3: 日本語要求記述中の品詞とクラス図の要素の対応

日本語要求記述中の品詞	クラス図の要素	例
可算名詞	リソース・クラスまたは属性	顧客，商品，名称など
質量名詞	属性値	数量，金額，色，サイズなど
動作動詞またはそれに由来する名詞	イベント・クラスまたは操作	受注，発注，契約など
状態動詞またはそれに由来する名詞	関連または関連クラス	在庫，提供，履行など

要素に加え，表 6.5 に示すモデル要素を用いてグラフに表記していく．表 6.5 には，UML クラス図と重複する表記が登場するが，その表記の意味は，存在従属クラス図として作成する場合には，表 6.5 に記載したモデル要素の意味を優先させるものとする．たとえば，誘導可能性のある関連の表記は，存在従属クラス図においては存在従属性を意味しており，コンポジションの表記は存在従属性でなおかつライフサイクルも一致するという属性的従属性を意味している．

表 6.4: 存在従属クラス図作成時のガイドライン

項		ガイドライン
属性		あるデータ項目をあるクラスの属性とするには，その値がそのクラスのインスタンスに存在従属する場合に限る．ただし，一旦あるクラスの属性とされたデータ項目であっても，構造を持っていたり，テータ項目に2回以上の繰り返しが認められる場合は，そのデータ項目を元のクラスから分離する．そして，もとのクラスと分離したデータ項目との間に存在従属性のライフサイクルが一致する形である属性的従属性を定義する．
独立クラス		そのインスタンスが，独立して存在可能なクラスのIDには「必ず」単一属性のIDを与える．
従属クラス		そのインスタンスが，独立して存在できない（存在従属な）クラスのIDには，「決して」単一属性のIDは与えず，「必ず」その存在根拠（前提）となるクラスのIDを含むIDを与える．結果，存在従属なクラスの識別子は複合主キーとなる．
イベント・クラス		そのインスタンスが，（リソースではなく）イベント（時点が帰属するインスタンス）の場合は，上記のルールに加えて，そのクラスのIDに「必ず」時点をあらわす時刻または版（バージョン）のシリアル値を含める．
その他		存在従属でないクラス間の関係は，汎化関係と単なる参照関係のみとする．
	i.	汎化関係の場合：サブクラスのIDは「必ず」スーパークラスと同じIDを共有する．たとえば，「プレミアム会員」のスーパークラスが「会員」クラスで，そのIDが「会員ID」であった場合は，「プレミアム会員」クラスのIDも「会員ID」となる．
	ii.	単なる参照関係の場合：参照元のインスタンスと参照先のインスタンスのそれぞれのライフサイクルは互いに独立している（制約を受けない）．このような場合は，表記では参照元のクラスから参照先のクラスへの依存関係を定義し，参照元は参照先のIDを外部キーとして保持するものとする．

図 6.4 左は前述のプロセスを経て，表 6.2 の例題について作成したエンティティの存在従属クラス図である．図 6.4 中の実線矢印（UML のモデル要素では，誘導可能性のある関連）は存在従属性を表す．矢印の元が従属クラス，矢印の先が親クラスを表す．また，破線矢印（UML のモデル要素では，依存関係）は単なる参照関係を表す．矢印の根元が参照するクラスで，矢印の先が参照されるクラスである．なお，図 6.4 左では，表 6.4 のガイドラインから導かれる ID（識別子）および，外部キーを明示的に記載している．従属クラスの ID は，必ず親クラスの ID をその一部に含む複合主キーとなっている．また，

表 6.5: 存在従属クラス図に表記する主なモデル要素

モデル要素	モデル要素の意味	表記
主キー属性	インタンスを識別ための属性	ステレオタイプ<<pk>>を付与
外部キー属性	インスタンスを外部参照するため属性	ステレオタイプ<<fk>>を付与
主キー属性兼外部キー属性	インタンスを識別ための属性がインスタンスを外部参照するため属性を兼ねている	ステレオタイプ<<pkfk>>を付与
存在従属性	あるインスタンスとその存在前提となるインスタンスの関係をクラスレベルに一般化したもの	親クラス ⟵ 従属クラス
強い存在従属性（属性的従属性）	あるインスタンスとその属性値の関係をクラスレベルに一般化したもの	元のクラス ◆⟵ 属性的従属クラス
（単なる）参照関係	あるインタンスが他のインスタンスを参照する関係，参照先の未定や変更があってもよい	参照先クラス ⟵ 参照元クラス
汎化関係	同じ主キー属性を共有するインスタンスのクラス間における特化したクラスとより一般的なクラスの間の関係	スーパークラス ◁⟵ サブクラス

時点が帰属するイベント・クラスの場合は，そのクラスの ID にイベントの生成時点を表す時刻の情報を含めていることに注目していただきたい．

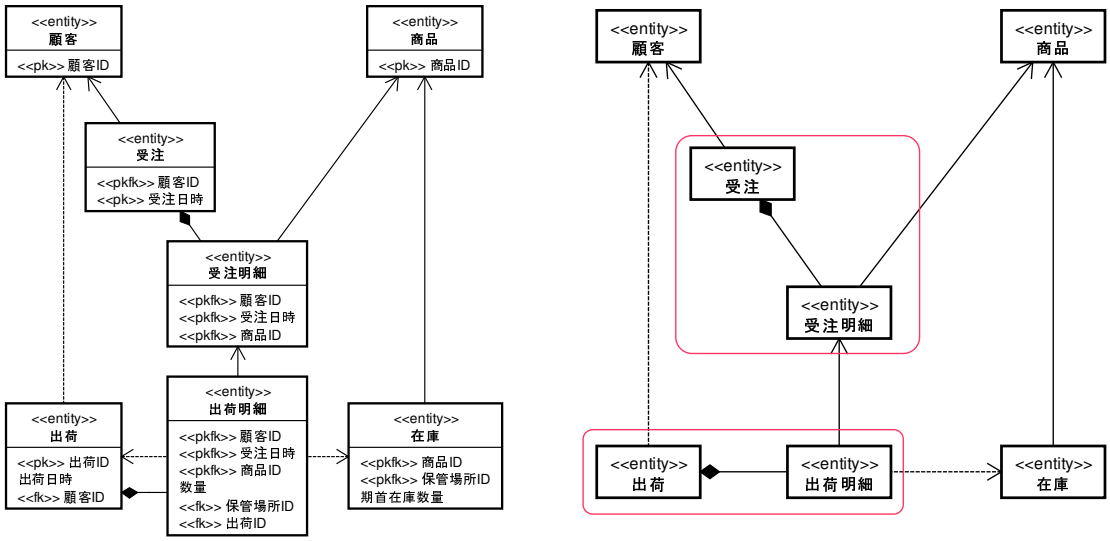

図 6.4: エンティティの存在従属クラス図（左側の図は識別子の関係を示すために記載）

属性的従属性も含めた存在従属性と単なる参照関係の違いは，前者が，依存先（親）クラスのインスタンスを参照元（存在従属）クラスのインスタンス（存在従属クラスのインスタンス）が生成される時点で，その識別子の一部として設定するため NULL が許容されず，また，設定後はライフサイクル途中で依存先を変更できないのに対して，後者は，参照先（参照される）クラスのインスタンスを参照元（参照する）クラスのインスタンスの非キー属性として保持するため，参照先のインスタンスが存在しない時点にお

いてはNULLが許容され，参照先を設定するタイミングは任意の時点で構わない．また，参照元のインスタンスのライフサイクル中において，参照先のインスタンスを自由に挿げ替えることができる点である．

図6.4左は，受注のインスタンスが存在するためには，

1. 発注者である顧客のインスタンスがあらかじめ存在していなければならないこと
2. 受注明細インスタンスは，その見出しとなる受注のインスタンスに属性的に従属し，注文対象である商品のインスタンスに存在従属すること
3. 出荷明細のインスタンスは，受注明細のインスタンスに存在従属し，同一顧客への出荷明細をまとめて出荷を構成すること

を表現している．

ここで，出荷明細と出荷の関係には，若干の注意が必要である．このケースでは明細が先に存在し，それらを見出しを付与して束ねる形であるため，著者は「明細－見出しパターン」と呼んでいる．「見出し－明細パターン」の逆であるが，概念としては出荷という１つの管理対象であることには違いがないため，通常の「見出し－明細パターン」と同様な属性的従属性と捉えることも可能である．図6.4左において出荷明細と出荷の間に2種類の依存関係が引いてあるのはそのためであるが，前述の理由により属性的従属性を優先してもよいだろう（図6.4右枠囲い部分）．

さらに，出荷明細のインスタンスは商品に存在従属する在庫のインスタンスを参照すること，また出荷は出荷先として顧客のインスタンスを参照すること，といった業務上の制約やルールを表現している．なお，以降の測定では，提案手法は，COSMIC法をベースにしているため，COSMIC法でその移動をカウントすべきデータ・グループは，提案手法では，エンティティの全属性またはエンティティの一部の属性を摘み集めたものに相当し，COSMIC法でいう注目オブジェクトは，永続化記憶域に格納される正規化されたデータであると定義されている [64] ため，提案手法ではエンティティそのものに相当すると見做す．そして，COSMIC法では，エンティティの属性数は測定に影響しないため結局は図6.4右の存在従属クラス図を用いることができる．

【Step 2】存在従属クラス図から利用者機能要件を求める

エンティティの存在従属クラス図が得られたならば，それをもとにCOSMIC法でいう利用者機能要件を導出する．利用者機能要件とは，利用者の視点でソフトウェアの機能のまとまりを捉えたものであり，UMLのユースケースとそのまま対応する．

図6.4には，明細エンティティを除くと5種類のエンティティが登場するため，次の5つの「大まかな利用者機能要件」が導かれる．これを提案手法では「ドメイン管理要件」

と呼ぶことにする．ドメイン管理要件は原則，存在従属クラス図に登場するエンティティの数だけ存在すると考えられる．エンティティが見出しと明細に分れている場合は，それらは概念としては 1 つであると見做す．従って，例題におけるドメイン管理要件は，1) 利用者を管理する，2) 商品を管理する，3) 在庫を管理する，4) 受注を管理する 5) 出荷を管理する，の 5 つである．

そして，1 つのドメイン管理要件は暗黙裡に 4 つの利用者機能要件を含意する．すなわち，1) 管理対象のインスタンスを新規登録する，2) 管理対象のインスタンスを検索・同定する，3) 管理対象のインスタンスを更新する，そして，4) 管理対象のインスタンスを抹消する，の 4 つである．これらは，CRUD（Create，Refer，Update，Delete）処理と対応し，エンティティのインスタンスを管理していくために欠かせない基本的な機能である．例題の場合は，5 つのドメイン管理要件が識別されため，自動的に 20 の利用者機能要件が導かれる．表 6.6 はこれらを一覧にした表である．

表 6.6: 利用者機能要件毎の CFP 値

ドメイン管理要件	FUR#	利用者機能要件（≒ユースケース）
顧客を管理する	1.1	顧客を新規登録する
	1.2	顧客を検索・同定する
	1.3	顧客の属性を更新する
	1.4	顧客を抹消する
商品を管理する	2.1	商品を新規登録する
	2.2	商品を検索・同定する
	2.3	商品の属性を更新する
	2.4	商品を抹消する
受注を管理する	3.1	受注を新規登録する
	3.2	受注を検索・同定する
	3.3	受注の属性を更新する
	3.4	受注を抹消する
出荷を管理する	4.1	出荷を新規登録する
	4.2	出荷を検索・同定する
	4.3	出荷の属性を更新する
	4.4	出荷を抹消する
在庫を管理する	5.1	在庫を新規登録する
	5.2	在庫を検索・同定する
	5.3	在庫の属性を更新する
	5.4	在庫を抹消する

表 6.6 からは，図 6.4 の存在従属クラス図を扱うのに 140 の CFP 値に相当する規模が必要になることも分る．

【Step 3】利用者機能要件を機能プロセスに展開する

COSMIC 法でいう機能プロセスは，システム境界外からのエントリに基づいて実行されるソフトウェアの機能で，利用者機能要件を構成するものであり，必ず複数種類のデー

タ移動が続けて実行されるものであるとされる [64]．ここでいうデータ移動とは，6.2.3 節で言及した COSMIC 法の測定対象となる表 6.1 の 4 種類である．

機能プロセスも存在従属クラス図からある程度は導くことが可能である．著者は，存在従属クラス図からユースケースおよびユースケース記述を逆生成する提案を行っている [53][56]．この提案によれば，存在従属性は，これで接続された 2 つのエンティティのインスタンス間に次のような制約を与える．

【生成制約】：独立エンティティのインスタンスは，制約（前提条件）なしに生成し，新規登録することができるが，従属エンティティのインスタンスは，その親となる従属先のインスタンスなしでは，生成・新規登録することができない．したがって，従属先のインスタンスの存在チェックを行う処理（データ移動）が必要である．

【削除制約】：あるインスタンスを削除する場合は，そのインスタンスに存在従属するか，あるいはそのインスタンスを参照しているインスタンスの存在チェックを行う処理が必要がある．存在する場合は，削除を中止する．ただし，業務要件によっては，存在従属または，参照しているインスタンスも含めて連鎖的に削除しなければならない場合もある．

上記制約から，例題の場合，図 6.4 が示す通り，受注は，顧客と商品に存在従属するため，受注を新規登録するには，表 6.7 に示す機能プロセスが必要であることが導かれる．受注を新規登録するためには，あらかじめ登録済みの顧客のインスタンスと商品のインスタンス群を新規の受注のインスタンスを生成して結び付けなければならないためである．

表 6.7: 利用者機能要件を機能プロセスに展開した表

FUR#	利用者機能要件（≒ユースケース）	FP#	機能プロセス
3.1	受注を新規登録する	3.1.1	顧客を同定する
		3.1.2	商品と数量を同定する
		3.1.3	受注を登録する

存在従属クラス図は，エンティティのインスタンスのライフサイクルに関する制約を表現しているため，存在従属クラス図が得られれば，ドメイン管理要件，利用者機能要件，および，利用者機能要件を実現するための機能プロセスも自動的に導かれることは注目に値する．しかしながら，提案手法では，機能プロセスにまで詳細化を行う必要はない．その代わりに，あらかじめ利用者機能要件タイプ毎に CFP 値を計算した後述の表 6.8 の値を適用することができる．

【Step 4】存在従属クラス図から利用者要件毎の CFP 値を求める

提案手法では，存在従属クラス図と利用者機能要件の一覧が得られれば，利用者機能要件毎の CFP 値は，表 6.8 を用いて求めることができるようにした．表 6.8 は，利用者

機能要件のタイプ別に存在従属クラス図に登場するエンティティの制約を考慮して著者があらかじめ CFP 値を計算したものである．このように，あらかじめ CFP 値が計算できるのは，存在従属クラス図には 6.3.2 節で述べた制約が表現されているからである．

ここで表 6.8 について若干の説明を行う．表 6.8 は，たとえば，図書館で利用者と図書を同定して貸出を行うのも，通販サイトで会員と商品を同定して販売を行うのも機能規模に影響するデータ移動という観点からは全く同じであるとの発想に基づいている．表 6.8 では，エンティティを単独で存在可能なタイプ，単独で存在できないタイプ，他のエンティティから依存や参照されている場合に分け，それらに対して，新規登録，検索・同定，更新，および削除を行うという抽象的な機能プロセスを想定し，それらに COSMIC 法を忠実に適用してそれぞれの CFP 値を計算している．

文献 [64] には，受注を登録する場合の計測例が掲載されているが，そこでの典型的なデータ移動は次の通りである．1) 顧客から舞い込んだ注文，すなわち「顧客」と「商品」群に関するデータ・グループの 2 つのエントリ.「受注」オブジェクトを記述するこれらのエントリの内，最初のものは機能プロセスを開始するためのデータ移動でもある．2) 指定された「顧客」と「商品」が実在するかを確認するための「顧客」と「商品」に関するデータの 2 つのリード．3) 登録されたデータを記憶域に移動するための「受注」に関するデータの更新（1 つのリードと 1 つのライト）．そして，4) 受注の合計や各「商品」に対応した倉庫への出荷指示などを含んでいる「受注確認」メッセージを含んだデータの 1 個以上のエクジット．合計 7 種類以上のデータ・グループの移動が計測されるとされ，これは，表 6.8 の FUR タイプ#6 の CFP 値と一致している．

表 6.8 作成時に考慮したことは，手作業において使用しやすい表にすること，および重複計測をできるだけ排除することである．そのために，表 6.4 に示した存在従属クラス図作成時のガイドラインの情報を活用している．具体的には，利用者機能要件のタイプでは，存在従属クラス図上のあるエンティティからみて，そのエンティティが直接依存するエンティティの種類数と，そのエンティティを直接参照するエンティティの種類数だけを問題とし，芋づる式に参照を辿ってエンティティ数をカウントしないようにしている．これは，表の使いやすさと重複計測の回避のためである．そして，上位のエンティティのドメイン管理要件を実現するためにカウントされるであろうデータ・グループの移動は重複してカウントしないようにしている．たとえば，再び表 6.7 の利用者機能要件である「受注を新規登録する」に注目する．

もしも，ここで表 6.7 に記載された個々の機能プロセスに対して誤って表 6.8 を適用した場合は，$4 + 4 + 7 = 15$ という過大な CFP 値を得ることになってしまうが，表 6.8 は「機能プロセス」ではなく「利用者機能要件」に対して適用するように作成されていることに注意されたい．

この例では表 6.8 の FUR タイプ#6（n=2）に該当するが，そこでは n 種類の親クラスのインスタンスを検索・同定するために，あらためてそのためのデータをエントリすべ

くカウントするようにはなっていない．もしもそうした場合は，少なくとも CFP 値は n だけ余分に増加してしまう（エクジットについても同様）．しかしながら，実際に登録するインスタンスは n 種類の親クラスのインスタンスは既知であるとしなければならない．よって，これらのエントリをカウントしないようにしている．表 6.8 の使い方は以下の通りである．

1. Step 3 で明らかにした利用者機能要件に着目する．
2. 利用者機能要件が扱うエンティティの特性を Step 1 で作成した存在従属クラス図から確認する．このとき，対象となるエンティティが，存在従属クラス図上で，何種類の直接の依存先と何種類の直接の被依存エンティティを持つかを確認しておく．
3. 表 6.8 を参照しながら，着目している利用者機能要件のタイプに応じた CFP を求める．

たとえば，例題の表 6.6 について実施した場合,「顧客を新規登録する」利用者機能要件は，表 6.8 の FP タイプ#6 に該当し，受注の n（存在従属している親エンティティの数）は 0 であるため，その CFP 値は 5 である．また,「顧客を抹消する」は，表 6.8 の FP タイプ#5 に該当し，顧客に直接依存するエンティティの数は 2（受注と出荷）であるため，その CFP 値は 10 である．この作業を，表 6.6 のすべての利用者機能要件について適用すると表 6.9 を得る．表 6.9 からは，図 6.4 の存在従属クラス図を扱うためのアプリケーションの機能規模は 139CFP であることも分る．

6.4　比較のためのケーススタディ

本節では，ある程度の大きさと複雑さを有する業務アプリケーションに対して提案手法を適用して機能規模を測定した結果と，COSMIC 法を本来の手順で適用して機能規模を測定した結果を比較する．COSMIC 法のためには，要求記述がよりイベントフローに近い情報をもっている方が適しているため，測定対象には，ビジネスホテル宿泊予約サイトの事例を用いた．この事例ではインターネットから日本国内の提携ホテルの宿泊プランの検索と予約を行うことができる．図 6.5 に宿泊予約サイトのやや詳細な要求記述を示す．

6.4.1　提案手法による測定

本節では，図 6.5 の要求記述に対して，6.3.2 節で述べた提案手法の手順を適用する．具体的には，1) 存在従属クラス図の作成，2) ドメイン管理要件の識別，3) 利用者機能要件

表 6.8: 存在従属性から利用者機能要件タイプ毎にあらかじめ計算で求めた CFP 値の表

FURタイプ#	利用者機能要件(FUR)タイプ	Entry	Read	Write	eXit	CFP値
1	あるエンティティのインスタンスを検索･同定する	同定するためのIDまたはキーワードを入力する	そのエンティティの該当するインスタンスを検索する	なし	同定に成功したインスタンスの諸属性を表示する 同定に失敗したメッセージを表示する	4
2	親を持たない独立クラスのあるエンティティのインスタンスを新規登録する	新規登録するインスタンスの諸属性値を入力する	重複登録を避けるためエンティティのインスタンスを検索する	新規インスタンスを書き込む	登録に成功したインスタンスの諸属性を表示する 登録に失敗したメッセージを表示する	5
3	親を持たない独立クラスのあるエンティティのインスタンスの属性値を更新する	更新対象のインスタンスを同定するために機能プロセスタイプ#１を実行する．そのためのCFP値は4				8
		属性値を更新するインスタンスの諸属性値を入力する	なし	属性値変更後のインスタンスを書き込む	更新に成功したインスタンスの諸属性を表示する 更新に失敗したメッセージを表示する	
4	あるエンティティのインスタンスを削除する．ただし，そのエンティティに存在従属したり，そのエンティティを参照するエンティティはグラフ上に存在しない	削除対象のインスタンスを同定するために機能プロセスタイプ#１を実行する．そのためのCFP値は4				8
		削除の実行を入力する	なし	同定されたインスタンスを削除する	削除に成功したインスタンスの諸属性を表示する 削除に失敗したメッセージを表示する	
5	あるエンティティのインスタンスを削除する．ただし，そのエンティティに存在従属したり，そのエンティティを参照するn種類のエンティティがグラフ上に存在する	削除対象のインスタンスを同定するために機能プロセスタイプ#1を実行する．そのためのCFP値は4				n+8
		削除対象のインスタンスに存在従属，または削除対象のインスタンスを参照しているインスタンスを同定するために，削除対象のエンティティに存在従属，または削除対象のエンティティを参照しているエンティティの種類数nだけReadする．そのためのCFP値はn				
		削除の実行を入力する	なし	同定されたインスタンスを削除する	削除に成功したインスタンスの諸属性を表示する 削除に失敗したメッセージを表示する	
6	n種類の親を持つ，ある存在従属エンティティのインスタンスを新規登録する	親エンティティのインスタンスを同定するために，種類数nだけエンティティをReadする．そのためのCFP値はn				n+5
		自分自身を新規登録するために機能プロセスタイプ#2を実行する．そのためのCFP値は5				
7	n種類の親を持つ，ある存在従属エンティティのインスタンスの属性値を更新する	親エンティティのインスタンスを同定するために，種類数nだけエンティティをReadする．そのためのCFP値はn				n+8
		自分自身の属性値を更新するために機能プロセスタイプ#3を実行する．そのためのCFP値は8				

の識別，4) 表引きによる利用者管理要件毎の CFP 値の取得の順で測定を進める．図 6.6 に要求記述から作成したエンティティの存在従属クラス図を示す．複雑さを回避するため図 6.6 では，表 6.4 で示した存在従属クラス図作成時のガイドラインから導かれる識別子や外部キーは表記していない．そして，表 6.10 に存在従属クラス図から導出したドメイン管理要件，利用者機能要件および，表 6.8 を参照することによって求めた機能プロセス毎の CFP 値を示す．

表 6.10 の CFP 値は，利用者機能要件から直ちに図 6.6 の存在従属クラス図と表 6.8 を参照して求めたものである．提案手法では，このサイトのソフトウェアの機能規模は 53 と測定された．なお，表 6.10 では，図 6.5 の記述からだけでは直接的に求めることができない利用者機能要件についてはグレイアウトしてある．ここで，機能規模の測定とは直接的には関係ないが，要求記述からエンティティの存在従属クラス図をまず最初に作成することは，ドメイン管理要件を明確にし，それらの CRUD の必要性から利用者機能要件がほぼ求まるため，要求定義プロセスにおける機能要求の網羅性確認のために非常に有効な方法であると思われる．

表 6.9: 利用者機能要件毎の CFP 値

ドメイン管理要件	管理対象の親クラスの数 n	FUR#	利用者機能要件（≒ユースケース）	CFP値
顧客を管理する	0	1.1	顧客を新規登録する	5
		1.2	顧客を検索・同定する	4
		1.3	顧客の属性を更新する	8
		1.4	顧客を抹消する	10
商品を管理する	0	2.1	商品を新規登録する	5
		2.2	商品を検索・同定する	4
		2.3	商品の属性を更新する	8
		2.4	商品を抹消する	10
受注を管理する	2	3.1	受注を新規登録する	7
		3.2	受注を検索・同定する	4
		3.3	受注の属性を更新する	10
		3.4	受注を抹消する	9
出荷を管理する	1	4.1	出荷を新規登録する	6
		4.2	出荷を検索・同定する	4
		4.3	出荷の属性を更新する	9
		4.4	出荷を抹消する	8
在庫を管理する	1	5.1	在庫を新規登録する	6
		5.2	在庫を検索・同定する	4
		5.3	在庫の属性を更新する	9
		5.4	在庫を抹消する	9
			合計	139

6.4.2　COSMIC 法による測定

今度は，図 6.5 の要求記述に対して，6.2.3 節で説明した COSMIC 法の手順を適用する．COSMIC 法では，最初に利用者機能要件を明確にする必要がある．利用者機能要件は，UML のユースケースに相当するとされる [66] ため，最初にユースケース図を作成した（図 6.7）．

次に，ユースケース毎にイベントフローを作成し，ユースケースを論理的に実現するための機能プロセスを洗い出す．これは，結構労力の必要な作業である．もちろん，開発が決定したユースケースに対してこの作業を行うのは必要であるが，作り込みを行うかどうかが定まっていないユースケース候補に対してこの作業を行うのは避けたいところである．しかしながら，COSMIC 法で機能規模の測定を行うのであればそうはいかない．

表 6.11 は，そのようにして識別した，おのおのの機能プロセスと機能プロセスが移動するデータ・グループの種類を数えたものである．利用者機能要件欄には，図 6.7 で明らかにしたユースケース名を記入した．機能プロセス欄には，ユースケース毎に作成したイベントフローで明らかにした，アクターの入力からそれに対するシステムの応答までの対話の一往復を記入してある．そして，エントリ欄からエクジット欄の間のデータ・グループは図 6.6 の存在従属クラス図に記載されているエンティティをそのまま使用した．エクジット欄で X(eXit) が複数マークされている機能プロセスは，アクターへの通常の応答以外にもエラーメッセージの出力とメール送信があることを示している．そして，CFP 値の欄には，その行に含まれる E，R，W，X の個数を単純にカウントした値が記入してある．

このシステムは，インターネットから日本国内のビジネスホテルの宿泊プランの検索と宿泊プランの予約を行うことができる．宿泊プランの検索は会員でなくても可能であるが，予約は会員でないとできない．利用者が宿泊を希望する地区（都道府県，エリア），利用期間（チェックイン日付，および泊数）を指定すると，指定した地区に所在する提携ビジネスホテルの一覧が表示される．利用者が一覧に表示されたホテルを選択すると，指定されたホテルの指定された宿泊期間に予約可能な宿泊プランの一覧が表示される．宿泊プランとは，部屋グレード，部屋ベッドタイプ，喫煙可否，食事オプションの組み合わせに名前を付けたものである．利用者が一覧から希望する宿泊プランを選択すると，会員の認証が求められる（認証済みでない場合）．会員はあらかじめ登録した会員IDとパスワードを入力する（この際，会員登録が済んでいない場合は，新規登録を行うこともできる．その場合，利用者は氏名，住所，メールアドレス，パスワードを登録する）．認証が成功すると，選択された宿泊プランの詳細，チェックイン日，泊数，チェックイン日からチェックアウト日にかけての日毎の宿泊料金，そして宿泊料金の合計金額が表示される．会員が表示内容を確認し，予約を実行すると，宿泊プランが引当てられ（ただし，具体的な部屋の割り当てまでは行われない），予約番号が表示されるとともに，予約番号と予約内容が記載された電子メールが会員の登録済みアドレス宛に届く．

図 6.5: サンプルプロジェクトの要求記述

6.4.3　計測結果に関する考察

(1)　ケーススタディの計測結果について

さて，表 6.10 と表 6.11 では，異なるプロセスで CFP 値の合計を求めているにも関わらず，提案手法が 53 で，COSMIC 法は 52 と極めて近い値になっている．また，表 6.12 は，表 6.10 と表 6.11 の値を，ドメイン管理要件毎に集計して比較したものであるが，こちらも極めて近い値になっている．

これらの値が世界標準の COSMIC 法の値であることの価値は大きい．業界では CFP を基準に，これを介して，人時値への換算事例や実装環境毎のソースコード行数（LOC）への換算実績データが蓄積されつつあるからである．ただし，COSMIC 法だけを用いた測定では，機能プロセスに近視眼的にならないように注意しなければならない．なぜならば，表 6.10 中でグレイアウトしてある利用者機能要件は，要求記述からは直接読み取りにくいものの必要な要件のはずである．提案手法では，先に存在従属クラス図に変換し，利用者機能要件はそこから導出するため，このような利用者管理要件も取りこぼさないが，COSMIC 法では取りこぼしてしまう可能性がある．これは，高速道路の建設に例えれば，本線の見積りに気を奪われ，パーキングやインターチェンジの存在を見落とすようなものである．

(2)　重複計測について

利用者機能要件には，取りこぼしのリスクとは反対に，それらの機能規模を重複計測するリスクがある．提案手法では，6.3.2 節で説明したような対策を行ってはいるものの重複計測を回避することはできない．なぜならば，存在従属クラス図に登場する個々のエ

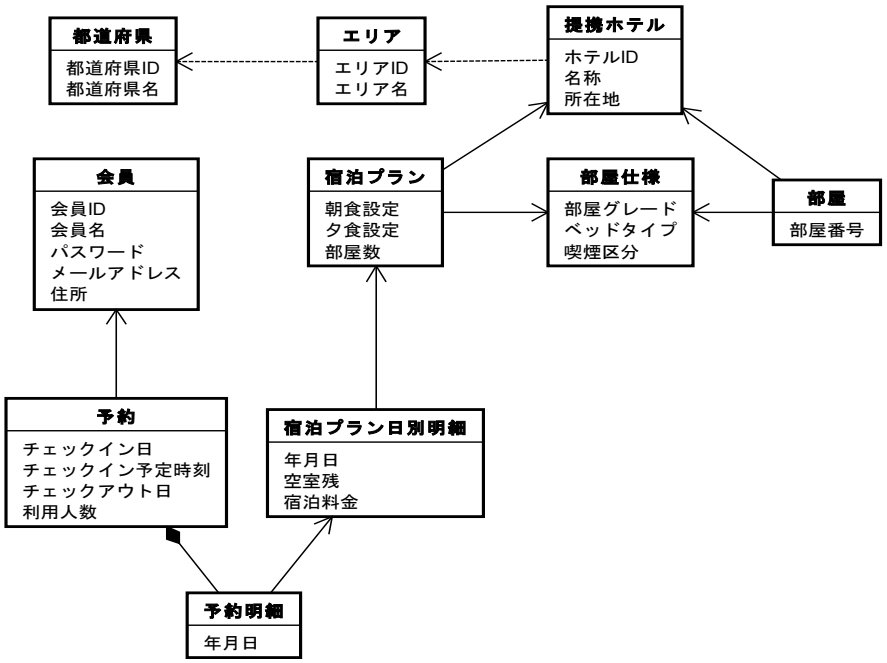

図 6.6: エンティティの存在従属クラス図

ンティティについて CFP を求めていく際に，どうしても依存性の直上・直下のエンティティが含まれるデータ・グループの移動を重複してカウントするためである．

回避策としては，次のようなアルゴリズムが考えられる．すなわち，

1. 存在従属クラス図に登場するすべてのエンティティについて，親のないエンティティから開始して，その依存関係を再帰的下降的に辿りつつ，それぞれを CRUD するための利用者機能要件をすべて生成する．その際，利用者要件には，親のないエンティティを 0 としたレベル番号を付与する．親のないエンティティが複数存在する場合には，それぞれについて同様に利用者要件を生成する．再帰関連が現れた場合は 1 回だけの再帰で処理は打ち切るようにする．

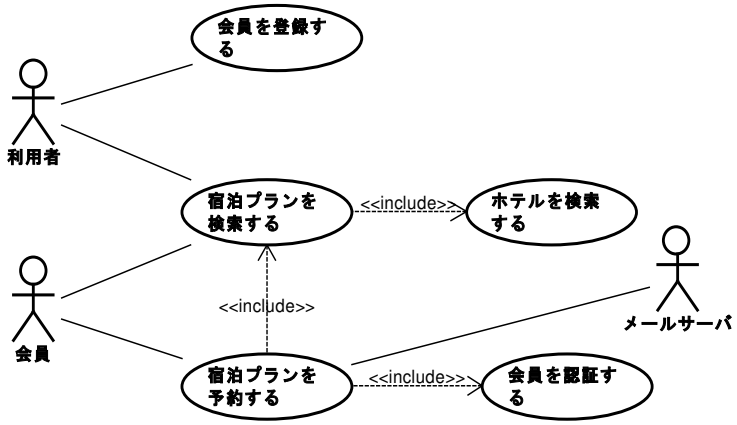

図 6.7: 宿泊予約サイトのユースケース図

表 6.10: 提案手法で求めた CFP 値

ドメイン管理要件	管理対象の親クラスの数n	利用者機能要件	提案手法で求めたCFP値
会員を管理する	0	会員を新規登録する	5
		会員を検索･同定する	4
		会員を更新する	8
		会員を抹消する	9
予約を管理する	2	予約を新規登録する	7
		予約を検索･同定する	4
		予約を更新する	8
		予約を抹消する	8
宿泊プラン日別明細を管理する	1	宿泊プラン日別明細を新規登録する	6
		宿泊プラン日別明細を検索･同定する	4
		宿泊プラン日別明細を更新する	9
		宿泊プラン日別明細を抹消する	9
宿泊プランを管理する	2	宿泊プランを新規登録する	7
		宿泊プランを検索･同定する	4
		宿泊プランを更新する	8
		宿泊プランを抹消する	9
部屋仕様を管理する	0	部屋仕様を新規登録する	5
		部屋仕様を検索･同定する	4
		部屋仕様を更新する	8
		部屋仕様を抹消する	9
部屋を管理する	2	部屋を新規登録する	7
		部屋を検索･同定する	4
		部屋を更新する	8
		部屋を抹消する	8
提携ホテルを管理する	1	提携ホテルを新規登録する	6
		提携ホテルを検索･同定する	4
		提携ホテルを更新する	9
		提携ホテルを抹消する	10
エリアを管理する	1	エリアを新規登録する	6
		エリアを検索･同定する	4
		エリアを更新する	9
		エリアを抹消する	9
都道府県を管理する	0	都道府県を新規登録する	5
		都道府県を検索･同定する	4
		都道府県を更新する	8
		都道府県を抹消する	9
		要求記述に直接関係するCFP値合計	53
		CFP値合計	245

2. 生成が終了したならば，利用者要件名で整列し，重複する利用者要件を除去する．

3. 今度は，存在従属クラス図の末端側に位置するエンティティを CRUD する利用者機能要件（付与されたレベル番号が大きい利用者機能要件）からレベル番号の降順に，参照先のエンティティの移動を除外したデータ移動を計測する．

(3)　エントリのバリエーションについて

提案手法では，エンティティごとに基本となる 4 つの利用者機能要件（新規登録，検索，更新，削除）を想定し，それを基に CFP 値の単価を設定している．しかしながら，新規登録，検索，更新について，1 つのエンティティに対して様々なバリエーションが存在する業務アプリケーションも多いはずである．例えば，新規登録においては，画面上での 1 件ずつエントリするほかに，表計算ソフト等で複数件数のデータを用意し，それを一括エントリするような機能が実装されることが多い．

最後に，このようなエントリのバリエーションがCOSMIC 法で計測する機能規模に影響するかについて検討する．結論から先に言えば，COSMIC 法では，一切影響されない．なぜならば，COSMIC 法による機能規模測定は，ソフトウェアの「規模」に影響するかもしれない機能性のあらゆる側面を測ろうとしているわけではないからである [64]．したがって，COSMIC 法では，ただ愚直に移動するデータ・グループ種類数をカウントするのみであり，エントリの方法や実現手段方法の違い，あるいは 1 データ移動あたりのデータ属性の数の影響はこの測定法では捕らえられない．

表 6.11: COSMIC 法で求めた CFP 値

利用者機能要件	機能プロセス	トリガ・エントリ	会員	都道府県	エリア	提携ホテル	宿泊プラン	部屋仕様	宿泊プランの日別明細	予約	予約明細	エクジット	CFP値
会員を登録する	アクターがメニューから会員登録を選ぶと，会員登録フォームが表示される	E										X	2
	アクターが会員属性を入力すると，当該会員が新規登録され，登録完了メッセージが表示される	E	R W									X X	5
会員を認証する	アクターが会員IDとパスワードを入力すると，認証の成否が表示される	E	R									X X	4
ホテルを検索する	アクターが都道府県とエリアを指定すると，当該エリアに存在する提携ホテルの一覧が表示される	E		R	R	R						X X	6
宿泊プランを検索する	アクターが，宿泊を希望する地区（都道府県，エリア），利用期間（チェックイン日付，および泊数）を指定すると，システムは「ホテルを検索する」を実行する	E	ホテルを検索するでカウント済みの6										7
	アクターがホテルを選択すると，指定されたホテルの指定された宿泊期間に予約可能な宿泊プランの一覧が表示される	E				R	R	R	R			X X	7
宿泊プランを予約する	アクターが宿泊プランを選択すると，システムは「会員を認証する」を実行する	E	会員を認証するでカウント済みの4										5
	認証が成功すると，選択された宿泊プランの詳細，チェックイン日，泊数，チェックイン日からチェックアウト日にかけての日毎の宿泊料金，そして宿泊料金の合計金額が表示される	E				R	R	R	R			X X	7
	アクターが予約を実行すると，宿泊プランが引当てられ（ただし，具体的な部屋の割り当てまでは行われない），予約番号が表示されるとともに，予約番号と予約内容が記載された電子メールが会員の登録済みアドレス宛に届く．	E							R W	W	W	X X X	8
											CFP値合計		51

表 6.12: 提案手法と COSMIC 法で求めた CFP 値の比較

ドメイン管理要件	提案手法で求めたCFP値	COSMIC法で求めたCFP値
会員を管理する	9	11
予約を管理する	44	40
CFP値合計	53	51

6.5　結言

本節では，業務アプリケーションの機能規模測定のためにエンティティの存在従属性に着目したグラフを導入することによってCOSMIC 法をカスタマイズする手法を提案し

た．COSMIC 法を開発対象やプロジェクトの事情に合うようにカスタマイズする提案はすでにいくつか存在している（[67][68][69] など）が，エンティティの存在従属性に由来する制約を，機能プロセスへの展開ルールとして位置づけ，扱うエンティティの存在従属クラス図上の位置に着目することによって機能規模の測定に結び付けたところに本提案手法のオリジナリティがある．

渡辺はそのブログ [70] の中で，業務アプリケーションというものが「テーブル構造から導出可能な多彩な影」でしかない，と発言しているが，まさにその通りであろう．エンティティの構造が決まれば，アプリケーションのユースケースはそれに沿って大部分は自動的に決まってしまうものであること [53] を提案手法の検討中にも改めて確認した．

本節の提案手法が要求定義プロセスの早期の段階で簡単に規模を見積れる手法として幅広く利用され，システム開発の依頼者と開発の技術者の間で互いに納得感のある開発請負契約を締結したり，あるいは機能規模を基準にプロジェクトの生産性や欠陥密度など検討する材料に使われるようになることを願っている．提案手法には，特に表 6.5 などはまだまだ改良の余地が残されていると思われる．それらは現場で提案手法を実際に運用する中で，改良していきたい．また，運用の中で CFP 値と実際のコストの関係も明らかにしていきたい．

第7章　存在従属分析用DSL

7.1　緒言

企業情報システム開発の分野では，システム間の相互連携のためにRESTfulなWebサービス技術[71]や分散台帳（ブロックチェーン）技術[72]の適用範囲が広がるにつれて，関係者間で管理すべきエンティティをリソースとして可視化し，合意し，交換していくことの重要性が増してきている．

そのため，近年では図7.1に示すようなアーキテクチャの採用が進んでいる．従来は，サーバ上のアプリケーション（APP）からSQLを発行して直接，基幹DBにアクセスしていたが，図7.1は，Webサービス API層を設け，サーバAPPからはこの層を介して間接的にアクセスする方式を示す．この方式により，基幹DB層をWebサービスAPI層によってカプセル化が可能となる[73].

このような状況に伴って，基幹DBの設計はもちろんのこと，リソースの構造を反映したAPI層の設計が重要課題となってきている．すなわち，業務で扱う重要な概念（エンティティ）とそれらの間の関係をモデル化した概念モデルを適切に設計，記述，維持しながら，それを元にデータベースを構築し，それらをサーバ・リソースとして，クライアントから容易にアクセスするためのAPIを提供していくことが開発者にとって重要な責務となっている．よって，これからの開発に用いるツールは，概念モデル構築，データベース設計，API設計といった一連の作業の整合性と合理化に資するものでなければならない．

概念モデルは従来，ER図やUMLクラス図などを用いて表記されてきた．著者は，概念モデル作成のガイドラインとしてエンティティの存在従属性に着目することを予てから提案してきた[52]が，ER図ではモデル要素が僅かに不足し，UMLクラス図にはモデル要素の種類こそ多いものの，そのままでは使いにくいといった問題があった．

そこで，本章では，業務で扱うエンティティの存在従属性に着目した分析に用いるのに特化したドメイン固有言語（DSL）を策定し，関係データベース（RDB）をエンティティの永続化手段，Webサービス API群をそれらへのアクセス手段とした開発への適用を提案する．

以下，7.2節では提案手法の関連研究を概観する．7.3節では，提案手法の詳細を述べる．7.4節では，提案手法の実現性と記述性についての評価を行う．7.5節は，本章の結

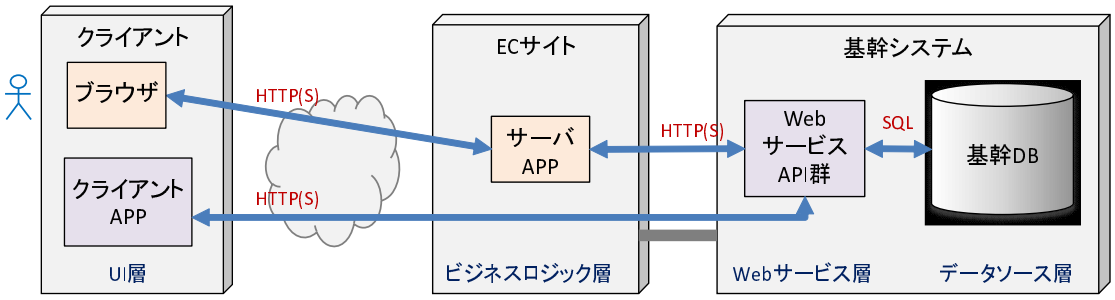

図 7.1: 近年，採用が多いアーキテクチャ（Web サービス層にて基幹 DB をカプセル化）

言である．

7.2　関連研究

7.2.1　エンティティの存在従属分析

エンティティの存在従属分析は，著者が予てから提案している，適切な概念モデルの構築を目的としたエンティティのインスタンス間に見られる存在期間の依存関係および，それによって生じる制約に着目する手法である [52].

存在従属分析の概要は，まず，業務で捕捉，蓄積，管理すべき概念をエンティティクラス（以下，単にクラスまたはエンティティと記す場合がある）として切り出し，そのインスタンスと存在期間（管理期間）が等しい値をそのクラスの属性とする．次に，そのインスタンスは，前提なしで存在できるのか，あるいは，他のインスタンスの先立つ存在を前提として存在し得るのかを，業務要件に照らして検討していく．そして，インスタンス間に次の 4 つのタイプの依存関係を見いだす．すなわち，1) あるインスタンスとその属性値の関係（属性従属性），2) あるインスタンスとそのインスタンスの存在前提となるインスタンスの関係（存在従属性），3) あるインスタンスは別のインスタンスを参照するが，参照先のインスタンスが参照元のインスタンスの存在前提ではない関係（参照従属性），および，4) あるインスタンスは，別のインスタンスに対して共通の属性群と固有の属性群を持つ関係（汎化関係），これら 4 種類のインスタンス間の関係を，概念間の関係として一般化し，モデル化することで，概念モデルを構築する手法である．

なお，概念モデルは，対象領域に関わるさまざまなエンティティとそれらの関係を説明する実装独立なモデルであり，エンティティは，業務ドメインで捕捉，蓄積，管理すべき重要な概念であるため永続化が前提である．現状では，業務システムは RDB を用いてエンティティを永続化するケースが最も多く，SQL は十分に高水準な言語となっているため，概念モデルは実装独立とはいいつつも，SQL の DDL 文に直結していて差し支えない，と著者は考えている．

存在従属分析を業務要件に即して忠実に行えば，正規化のプロセスなしで第 4 正規形以上の正規形レベルをもった概念モデルを得ることができるとされる [52]．そして，存在従属クラス図は存在従属分析の成果を表記するための有向グラフである．図 7.2 は物販業の受注管理ドメインについて存在従属分析を行った結果得られた，存在従属クラス図の例である．図 7.3 は，図 7.2 と等価で，図 7.2 の存在従属クラスが含意するとされる主キーと外部キーを文献 [52] の記述に従って明記することで得られた ER 図である．ここで法人顧客の法人番号 ，および個人顧客の個人番号 は，マイナンバーとも称される自然キー [1] であるため，顧客の主キーとして用いるのは適切でないことに注意されたい．また，図 7.4 は，注文エンティティの主キーを見直した ER 図である．

後の 7.3 で述べる提案手法では，図 7.4 に表記された主キーの付与方式を採用する．注文エンティティの主キーを顧客 ID と受注日時からなる複合主キー から，単一属性の注文 ID に変更する．顧客 ID を非キー属性の外部キーに格下げする代わりに，注文から見た顧客のオプショナリティを排除することと，受注日時に非 NULL 制約を付すことで，図 7.3 のモデルと同等のデータ整合性の論理的強度を確保している．なお，存在従属クラス図は UML クラス図の記法を流用しているため，静的な構造図に見えるが，実際はエンティティのインスタンスの存在期間に基づく依存関係を表現するので著者は動的モデルとしても位置づけている．日本語要求記述文からのエンティティの識別についての詳細は文献 [4] を，存在従属という概念，および存在従属クラス図作成の詳細については，文献 [52] を，それぞれ参照されたい．

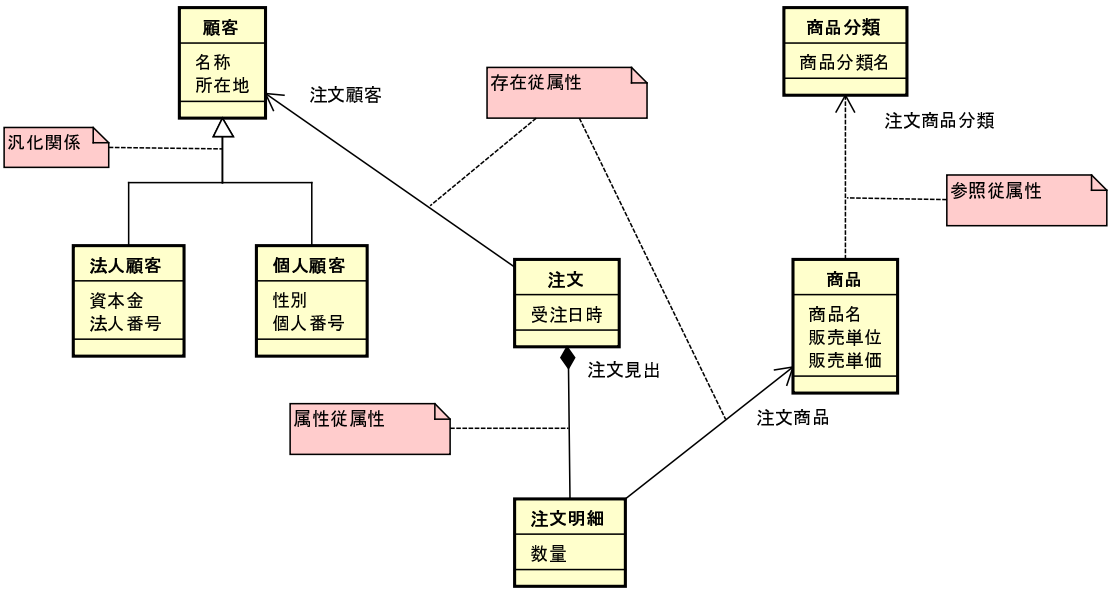

図 7.2: 受注ドメインの存在従属クラス図の例

[1] 自然キー（natural key）は，業務で用いられているキーであり，通常，識別以外にも分類や整列の機能を有する．その特性上，業務の都合で付与体系が変更される可能性がある．一方，乱数や連番などを組み合わせて人為的に付与されたキーは，人工キー（artificial key）と呼ばれ，識別以外の機能を持たないため，業務の都合で変更されることはない．

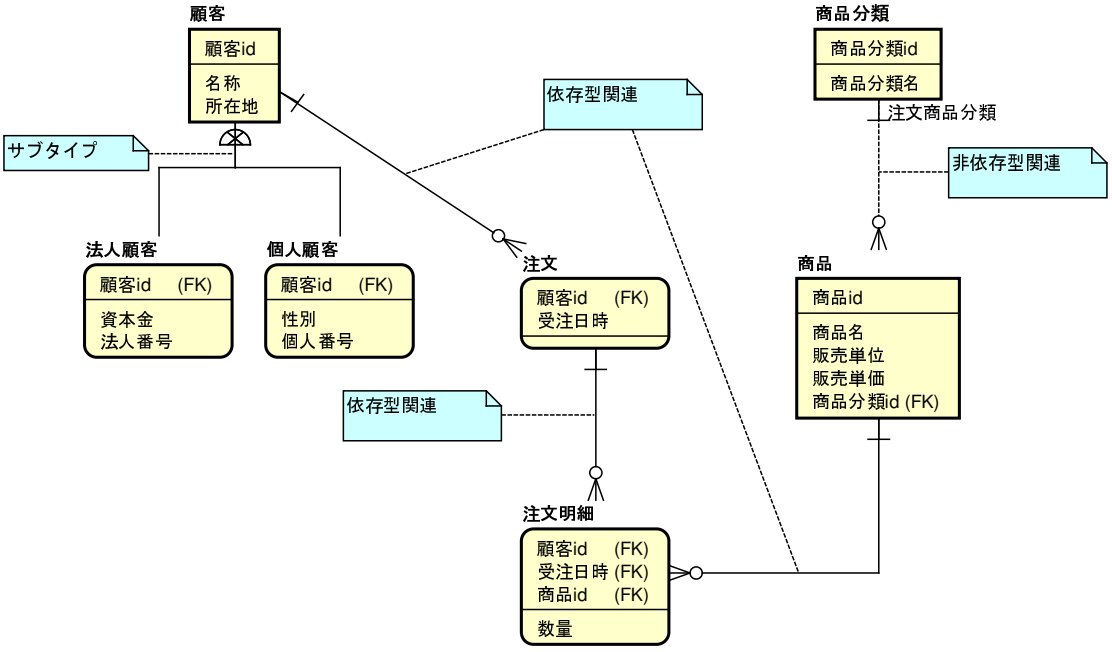

図 7.3: 図 7.2 を ER 図化したもの（文献 [52] のガイドラインにて主キーを定義）

7.2.2　ドメインモデルとしての DSL について

DSL（domain-specific language）とは，特定のタスク向けに設計されたコンピュータ言語を意味する [74]．DSL は一種類の課題をうまく実行することに集中したものであるため，扱う問題に寄与しない言語要素は思い切って削ぎ落とすことが出来る点が DSL 採用の最大の利点であると考えられる．そのことによって，DSL を人間にとって理解しやすい簡潔さに保つことができる．

従来，業務システムの核となるエンティティの構造を可視化するために，ER 図や UML のクラス図が用いられてきた．しかしながら，ER 図では，リレーションシップの種類が 3 種類（依存型，非依存型，サブタイプ）しかない上，あるエンティティから見た関連先のエンティティの役割を表記することができないため，現場で用いる場合には，注釈等で制約を補う必要がある．一方，クラス図では，関係の種類は 6 種類（依存関係，汎化関係，実現化関係，関連，集約，コンポジション）[35] と多いものの，業務システムの概念設計で使用するのは，関連，集約，コンポジション，汎化関係の 4 種類である．クラス図では，関連線は自由に引くことができ，関連の意味は関連名や関連端名によって柔軟に示せるものの，主キーや外部キーを明示的に表記するためには，限定子やステレオタイプを使って示す必要があるため，それらを愚直に行った場合，クラス図は複雑化し非常に見づらいものになってしまう．

存在従属分析では，使用するエンティティ間の関係は，属性従属関係，存在従属関係，

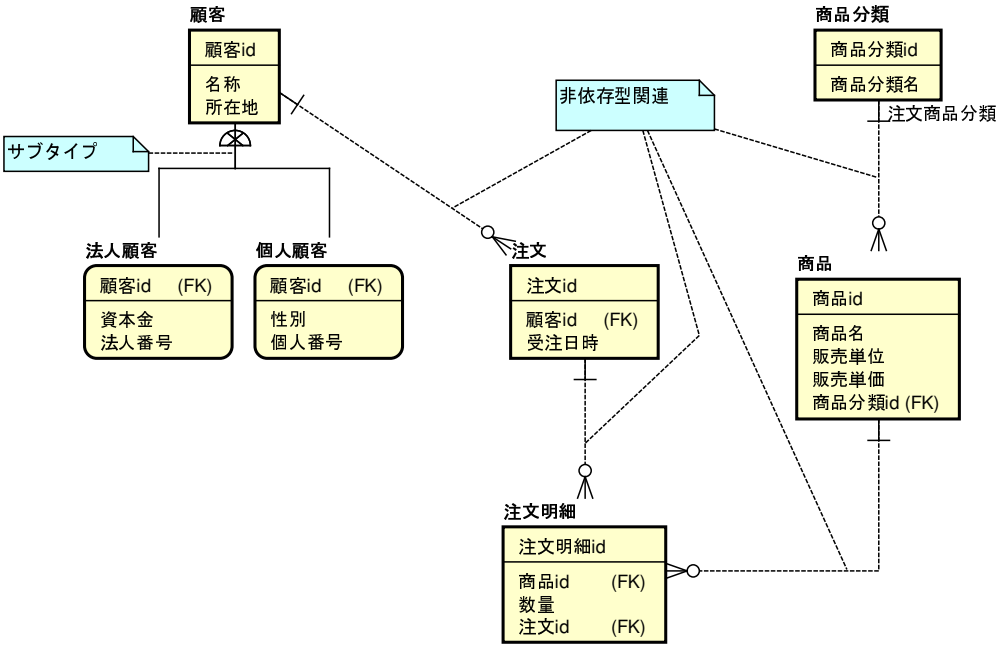

図 7.4: 図 7.3 の主キーを見直したもの（提案手法における主キーの付与方式）

参照関係，および汎化関係の 4 種類である．これは，ER 図のリレーションシップよりは 1 種類多く，UML クラス図のそれとは，種類数こそ 4 種類と等しいものの，それぞれの意味や含意する制約が微妙に異なるため，汎化関係を除いては，ストレートに対応づけることができない．たとえば，属性従属関連はコンポジション，存在従属関連は集約，参照従属関連は関連に近いと考えられるが，文献 [52] に示されるように，そのまま 1 対 1 対応というわけではない．また，それらの表記法をサポートするモデリングツール群を開発ツールとしてみた場合は，モデルから DB 生成，DB にアクセスするための API 生成，および API のドキュメント生成へとシームレスにつなげられると便利であるが，現時点でそのような範囲を一貫してサポートするツールは存在しない．

7.3　提案手法

7.3.1　提案手法の目標

そこで，著者は，存在従属分析の成果を書き表すための DSL を定義し，DLS4EDA（Domain Specific Language for Existence Dependency Analysis）と名付けた（以降，DSL4EDA と記す）．DSL4EDA は，次の事柄を達成すべくその仕様策定を行った．

a) 特殊なツールを用いることなく，テキストエディタだけでドメインの概念モデルが簡潔かつ機敏に記述できること．

b) UML クラス図の学習者が DSL4EDA に移行しやすいように，エンティティ間の関係を表す表記は，UML クラス図の図式表記に可能な限り似せること．

c) DSL の利点を活かして，DSL4EDA の要素は存在従属分析の結果を表記するために，必要最小限の構文とし，人間が読みやすく，習得のための学習コストを抑えること．

d) 存在従属分析結果のモデルを記述する際に，ER 図で足りない要素や，クラス図で表記しきれない要素を表記できるようにすること．

e) 表記されるエンティティやそれらの間の関係は，後続の機械処理によって，データベース定義や，定義されたデータベースにアクセスするための API 生成に直結できるものであること．特にデータベースへのアクセスを提供する API の実装は現在人気のある ORM 製品（Ruby の ActiveRecord[75] や PHP の Eloquent[76] など）を用いての実装から利用されるケースも考慮して，複合主キーは使わないこと．

f) モデル表記を人間にとって可読な簡素さに保つために，自明なことは，モデルに記述しないで済ませられるようにすること．

表 7.1 は DSL4EDA において表記を省略する（あるいは省略できる）モデル要素の一覧である．

表 7.1: DSL4EDA では表記を省略する要素

表記を省略する要素	省略する理由
主キー	DSL4EDAでは，主キーは一切表記できない．その代わり人工キーであるidが自動的に定義される．これにより設計者が誤って自然キーを主キーとして採用してしまうリスクを回避する
外部キー	関係のタイプによって自動的に決定されるため
多重度（カーディナリティおよびオプショナリティ）	関係のタイプによって自動的に決定されるため
属性の型が文字列型である場合の型指定	最も頻出する型であるため

7.3.2 DSL4EDA の仕様

DSL4EDA の文法を，図 7.5 に示す．図 7.5 では拡張バッカス・ナウア記法（BNF 記法）[77] を採用している．図 7.5 では，ダブルクォート（"）で囲で囲まれた文字列は，終端記号 であり，小なり記号（<）と大なり記号（>）で囲まれた文字列は，非終端記号 を表している．そして，縦棒記号（|）は，その前後にある終端記号や非終端記号のいずれかが選択されることを示す．また，プラス記号（+）は，直前の文字の 1 回以上の繰り返し，

終端記号でないシャープ記号（#）以降，行末までは，文法の BNF 表記そのものの注釈である．

DSL4EDA では，たとえば，ドメイン名は，2 重の角括弧（[[および]]）で囲まれた識別名であり，エンティティ名は，角括弧（[および]）で囲まれた識別名である．そして，識別名は，特殊記号以外の文字の 1 回以上の繰り返しであることを表現している．例として，図 7.2 に示した存在従属クラス図を DSL4EDA の文法に則って表記した場合は，図 7.6 のようになる．以下の節では，DSL4EDA の主だった文法と言語要素，および DDL への変換時の実装指定について，順に説明する．

```
<特殊記号> ::= "," | "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">" | "="
| "-" | "|" | ":" | "@N" | "@V" | "#" | "*" | "+"
<注釈> ::= "#" <文字>+ "¥n"
<識別名> ::= <特殊記号以外の文字>+
<ドメイン名> ::= "[[" <識別名> "]]"
<エンティティ名> ::= "[" <識別名> "]"
<属性名> ::= <識別名>
<役割名> ::= "(" <識別名> ")"
<属性型> ::= [ “string” ] | “number” | “dateTime”
<属性> ::= <属性名> [ ":" <属性型> ]["*"] # '*'は履歴管理したい属性に付与する
<属性並び> ::= "{" <属性> [","  <属性> ]+  "}"
# <属性並び反復> ::= "{" <属性並び> "}+"
<エンティティ型> ::= [ "@N" ] | "@V"
<エンティティ> ::= <エンティティ名> [ <属性並び> ]
<被役割> ::= <役割名>
<有向関連> ::= <被役割>"<>=" | <被役割>"<==" | <被役割>"1--" | <被役割>"<--"
| "=<>"<被役割> | "==>" <被役割> | "--1" <被役割> | "-->" <被役割>
<エンティティ関連> ::= <エンティティ> <有向関連> <エンティティ>
<特化関係> ::= <エンティティ> "<|-" <エンティティ>
<汎化関係> ::= <エンティティ> "-|>" <エンティティ>
```

図 7.5: DSL4EDA の構文規則

```
[[受注ドメイン]]
[顧客]{名称，所在地*}
[法人顧客]{資本金:number，法人番号}
[個人顧客]{性別，個人番号}
[注文]{受注日時:datetime}
[注文明細]{数量:number}
[商品]{商品名，販売単位，販売単価:number}
[商品分類]{商品分類名}
[法人顧客]-|>[顧客]
[個人顧客]-|>[顧客]
[顧客](注文顧客)<==[注文](注文見出)<>=[注文明細]==>(注文商品)[商品]-->(商品分類)[商品分類]
```

図 7.6: 図 7.2 の存在従属クラス図を DSL4EDA で表記したもの

(1) 識別名（Identifier）

識別名は，DSL4EDA 自体で用いる特殊記号を含まない任意の文字列である．識別名は，エンティティ名，属性名，被役割名の一部または全部として用いられる．なお，特殊記号は半角文字を使用する．

(2) ドメイン（Domain）

DSL4EDA では，モデル化の対象領域に識別名を与えて，ドメインを定義できる．ドメイン名は，エンティティクラスの定義を格納するパッケージの名称，エンティティに対応した RDB のテーブル群を格納するデータベースの名称として使用することを想定している．ドメインは名前空間を形成し，次に述べるエンティティ名はドメイン内でユニークである必要がある．DSL4EDA では，あるドメイン定義から，次のドメイン定義が現れるか，あるいは，DSL ファイルの終端までを，同一のドメインと解釈する．

(3) エンティティ（Entity）と属性（Attribute）

（3a）エンティティ（Entity） エンティティは，対象業務で捕捉，蓄積，管理すべき重要なデータ項目の塊である．何をエンティティとするかは，業務要件に強く依存する問題であるが，エンティティと属性の間の存在制約に着目することにより，かなりの精度でその塊を切り出すことは可能である [52]．DSL4EDA では，識別名を大括弧で囲むことにより，その識別名がエンティティの名称であることを表現する．たとえば，[顧客]，[注文]，[商品] の表記は，3 つのエンティティの存在と，それぞれのエンティティの識別名を表している．

【エンティティの実装指定】 1 つのエンティティは関係データベースの 1 つのテーブルにて実装する．エンティティ名が英語の場合，テーブル名はエンティティ名を複数形 にしたものに変換する．また，エンティティ名が日本語の場合は，エンティティ名の後ろに単純に英小文字半角の「s」を追加したものをテーブル名とする．

【エンティティが持つ属性群】 エンティティは 1 つ以上の属性を持つ．属性は，エンティティの性質を規定するデータ項目である．ある値を，あるエンティティのインスタンスの属性値とするためには，両者の存在期間が等しくなければならない．エンティティが，どのような属性を持っているかを表記するために，エンティティ名の後ろに属性並びを表記することができる．たとえば，「[顧客]{氏名，住所}」という表記は，顧客はエンティティであり，属性並びにて，氏名，および住所と命名された属性を持つことを示している．

【エンティティの主キー】 エンティティを識別するための主キー属性は ID 方式 [78] を採用する．RDB においては，主キー属性は識別性だけでなく，データの整合性を担保する役割を担う．たとえば，複合主キーは，複数の実在するエンティティの組合せに由来する制約を実装するのに簡潔かつ合理的な方法である．しかしながら，提案手法では，あえて ID 方式を採用する．ID 方式であるため，どのテーブルをとってもその主キーは，複合主キーは出現せず，すべて単一属性の ID である．

ID 方式を採用する理由は，1）ID は連番等の意味のない値であるため，未来永劫，業務の都合で変更される心配がないため．2）すべてのエンティティの主キーは ID である，と決めてしまうことによって表記を省略できるため．3）ID 方式は Rails の ActiveRecord[75] や Laravel の Eloquent モデル [76] など，現在よく使われている ORM 製品から RDB を扱う際に親和性が高いため．4）複合主キーを持ったテーブルを参照するための外部キーは，やはり，複合属性になるため，属性数が多くなると，扱いや管理が煩雑になるため．といった理由による．

ID 方式であっても，非キー属性である外部キーの組み合わせに対してユニーク制約を設定することより，複合主キー兼外部キーを用いる場合と同等の整合性を実現することができる．後述するように，DSL4EDA の実装指定では，外部キーまたは，その組合せに対して，非 NULL 制約やユニーク制約，連鎖削除等の設定を追加することによって，複合主キーが担う識別性以外のデータ整合性に関する機能を肩代わりさせる．

（3b）属性 (Attribute) 属性は，その値がエンティティのあるインスタンスを特徴付け，その値の存在期間（管理期間）がエンティティのあるインスタンスのそれと一致するデータ項目である．属性として，属性名を表す識別名と属性の型を定義する．属性名の直後に，コロン「:」に続けて，属性の型を指定する．属性の型は,「文字列（string)」,「数値（number)」,「日時（dateTime)」が指定できる．ビジネスドメインでは，文字列型が使用されるケースが圧倒的に多いため．属性の型が文字列型である場合に限り，その表記を省略してよい．

【属性並び】 属性並びは，中括弧内に属性をコンマ区切りで並べたものである．モデルは，上から下へ読まれていくため，人間にとっての可読性のために，属性並びはエンティティが最初に登場する位置で表記することを推奨する．

【属性値の履歴管理】 属性の直後にアスタリスク（*）を付与することで，その属性の値については履歴管理すべきことを指定できる．

履歴管理の実装指定は，図 7.7 のように，RDB と MongoDB[79] に代表されるドキュメント指向データベース（以下，DDB と記す）を併用して実装するのが現時点における最適解であると考えている．なぜならば，RDB はあらかじめ定義されたスキーマに従って，

格納されるデータの整合性を CUD 時に厳密にチェックするため，実行速度は多少犠牲なるものの，データの一貫性は RDBMS の機能によって担保される．一方，DDB は，データの一貫性の維持は犠牲にしても，高速なアクセスが必要で，スキーマを柔軟に成長させていくような用途に向いている．両者の特性を組み合わせることで，データの一貫性を犠牲にすることなく，柔軟かつ高速なデータ管理が実現できると考えられるからである．

図 7.7 は顧客エンティティの顧客住所が履歴管理指定された場合の実装イメージである．DSL4EDA の処理結果が実行される時点で，RDB 側に顧客テーブルが作成されるとともに，DDB 側には，顧客住所履歴を追記していくためのコレクションが作成される．顧客のインスタンスが新規登録されるタイミングで，RDB 側には，当該顧客の新規レコードが登録されるとともに，DDB 側には，当該顧客の住所履歴を記録していくためのドキュメントが挿入され，そのドキュメント ID が，RDB 側の住所履歴属性に値として登録される．

そして，顧客の住所が更新されるタイミングでは，RDB 側の顧客住所の値を最新の値で更新するとともに，DDB 側の当該顧客の住所履歴に関するドキュメントには新しい住所とその更新日時のペアが追記される．こうすることで，RDB 側の顧客テーブルには，従来通りアクセスできる上，顧客の過去の住所が必要になった場合には，アクセス時の引数に時点を追加指定することにより，指定された時点における住所の値を取得することができる．

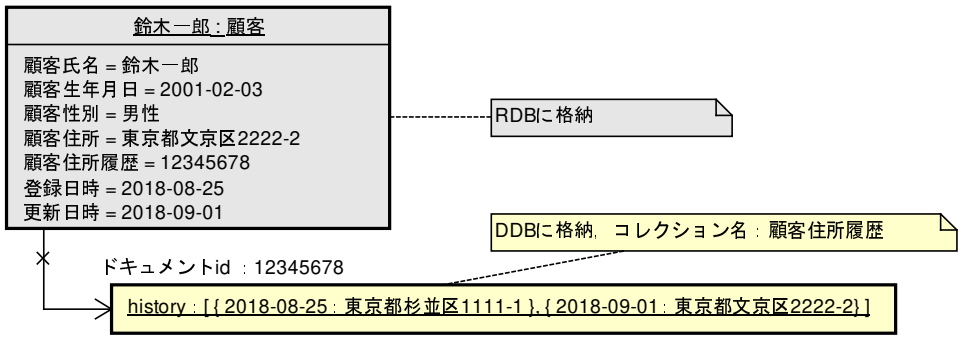

図 7.7: RDB と DDB を併用した属性値の履歴管理のイメージ

(4) 有向関連（Directed Association）

DSL4EDA で扱う 2 つのエンティティ間の関連は，属性従属関連，存在従属関連，参照従属関連の 3 種類であり，それらはすべて，依存元から依存先に向けての向きを持った有向関連である．依存元からみた依存先への多重度は,「1」または「0..1」に限定されるが,関連のタイプによってそれぞれ独特の制約を含意する．なお，図 7.8 に示されるように,3 つの関連種のうち，参照従属関連には参照先の多重度に応じて，さらに 2 つのバリエー

ションがあり，それぞれ右向きと左向きの表記を許すため，結果的に 8 つのバリエーションが生まれる．

```
<有向関連> ::= <被役割>"<>=" | <被役割>"<==" | <被役割>"1--" | <被役割>"<--"
             | "=<>"<被役割> | "==>"<被役割> | "--1"<被役割> | "-->"<被役割>
```

図 7.8: 有向関連の表記のバリエーション

依存先には，依存元から見た役割を,「被役割（responsibility）」として表記する必要がある．被役割は，識別名を小括弧で囲ったもので，UML クラス図における関連端名（ロール名）に相当する要素である．たとえば,「注文」エンティティから見た「顧客」エンティティの被役割を「(注文顧客)」として定義する．被役割はあとあと外部キーの名前にしたり，仮想表としてリソース化し，API 経由で完結したリソースとして検索，取得できるようにする際のリソース名としての使用に有用である．ちなみに，関連端名（ロール名）のモデル要素は，UML クラス図には存在するが，ER 図では，現在主流である IE 表記，IDEF1X 表記，それらいずれの表記においてもサポートされていない．

（4a）属性従属関連 (Attributive Dependency Association)

【表記】 属性従属関連は,「<>=」 または 「=<>」の記号で表記する．等号が置かれる側に依存元，その反対側に依存先のエンティティを記述する．なお，依存先のエンティティには被役割を付記するが，属性従属関連の場合に限り，その名称が依存先のエンティティ名と同じものを指定することができる．そして，その場合に限り，被役割の表記は省略してよいが，この場合は，被役割名として，依存元エンティティ名に依存先エンティティ名を結合した名称が採用される．

【適用】 この関係は，この記号で接続された 2 つのエンティティのインスタンスが有するライフサイクルが等しい場合に限り，適用することができる．

【適用例と実装指定】 属性従属関連の適用例としては,「注文」エンティティと「注文明細」エンティティの関係がある．これらは，本来，概念としては 1 つであるが，通常，1 回の注文において，注文明細は 1 件以上反復されるため,「注文明細」は「注文」から分離され，本体のエンティティ「注文」に向けて属性従属関係が定義されることになる．この例の DSL4EDA による表記は,「[注文]（注文見出）<>=[注文明細]」のように書ける．

属性従属関連を RDB にて実装する場合は，依存側のエンティティに，非 NULL 制約を付与された外部キーが定義され，その外部キーには連鎖削除が指定されるようにする．「[注文]（注文見出）<>=[注文明細]」の例から，生成すべき DDL 文は図 7.9 のようになる．注文明細テーブル（注文明細 s）から注文テーブルに向けての非 NULL 制約付きの外

部キーが生成され，さらに，その外部キーには連鎖削除が設定される．なお，後で API から「注文」そのものを 1 つの完結したリソースとして参照できるようにすべく，個々の注文明細と注文とを結合した仮想表（view）の DDL を，前述の被役割名（「注文見出」）を用いて生成する．

```
create table 注文s (
    id              integer primary key,
);
create table 注文明細s (
    id              integer primary key auto_increment,
    注文見出id      integer not null
    foreign key(注文見出id) references  注文s(id) on delete cascade
);
create view 注文見出s as
select 注文s.id , 注文明細s.id
From   注文s  inner  join  注文明細s on 注文s.id = 注文明細s.id;
```

図 7.9: 属性従属関連の実装例

（4b）存在従属関連（Existence Dependency Association）

【表記】 存在従属関連は,「<==」または,「==>」 の記号を用いて表記する．等号が置かれる側に依存元，反対側に依存先のエンティティを記述する．

【適用】 この関連は，依存元エンティティのインスタンスが存在できるためには，依存先のエンティティのインスタンスの先立つ存在が前提であり，かつ，依存元エンティティのインスタンスの存在期間を通じて，依存先のエンティティのインスタンスは消去も挿げ替えも行わない場合に限り適用することができる．結果として，依存元からみた依存先の多重度は厳密に 1 であり，依存先エンティティのインスタンス（これは依存元のインスタンスにとっての存在前提であるため）を挿げ替えたり消去したりすることは許されない上，依存元のインスタンスが 1 つでも存在している間は，依存先のエンティティの当該（依存されている）インスタンスは削除できないという制約が実現できる．

【適用例と実装指定】 存在従属関連の適用例としては,「納品」と「請求」の関係がある.「納品」のインスタンスが存在しなければ「請求」のインスタンスを作れないとする業務要件は頻繁に見受けられる．この場合，[納品](請求対象)<==[請求] のように表記する．存在従属関連を RDB にて実装する場合は，依存側のエンティティに，非 NULL 制約を付与された外部キーが定義される．この例の場合，図 7.10 のような DDL 文を出力する．

存在従属関連に関しては，業務ドメインでよく現れる別の例を見よう．たとえば,「在庫」は「倉庫」と「商品」に存在従属するエンティティである．提案 DSL の表記例は,

```
create table 納品s (
    id    integer primary key auto_increment,
);
create table 請求s (
    id    integer primary key auto_increment,
    請求対象id      integer not null,
    foreign key(請求対象id) references 納品s(id)
);
create  view  請求対象s   as
    select 納品s.id, 請求s.id
    from 納品s inner jopin 請求s on 納品s.id = 請求s.id;
```

図 7.10: 存在従属関連の実装例（その 1）

「[倉庫]（在庫倉庫）<==[在庫]==>(在庫商品)[商品]」となる．実在する倉庫のインスタンスと商品のインスタンスの組み合わせによって，在庫の諸属性が一意に特定できる．この場合，図 7.11 のような DDL 文を出力する．

```
create table 倉庫s (
    id    integer primary key auto_increment,
);
create table 商品s (
    id    integer primary key auto_increment,
);
create table 在庫s (
      id          integer primary key auto_increment,
      在庫倉庫id  integer    not null,
      在庫商品id  integer    not null,
      unique(在庫倉庫id, 在庫商品id),
      foreign key(在庫倉庫id) references  倉庫s(id),
      foreign key(在庫商品id) references  商品s(id),

);
create view 在庫倉庫在庫商品s as …
```

図 7.11: 存在従属関連の実装例（その 2）

（4c）参照従属関連（Referencing Dependency Association）

【表記】 参照従属関連は,「<--」または「-->」の記号を用いて表記する．マイナス記号が置かれる側に依存元（参照する側），反対側に依存先（参照される側）のエンティティを記述する．

【適用】 この関連は，依存元エンティティのインスタンスと依存先エンティティのインスタンスのライフサイクルは独立しており，依存元エンティティのインスタンスの存在期間を通じて，依存先のエンティティのインスタンスを消去したり挿げ替えたりしたい場合に限り適用することができる．

【適用例と実装指定】 参照従属関連の例として,「野球選手」と「球団」の関係を採り上げよう．この関係は，ある野球選手インスタンスの存在期間を通じて，所属球団は不在であったり，挿げ替えられたりされる．DSL4EDA では,「[野球選手]-->（所属球団）[球団]」のように表記できる．参照従属関連を RDB で実装する場合は，依存側のエンティティに，参照先のエンティティのインスタンスを特定するための外部キーが定義されるのが適切である（図 7.12）．

```
create table 球団s (
    id    integer primary key auto_increment,
);
create table 野球選手s (
    id    integer primary key auto_increment,
    所属球団id       integer,
    foreign key(所属球団id) references 球団s(id)
);
```

図 7.12: 参照従属関連の実装例（参照先の多重度が 0..1 の場合）

なお,「野球選手は常時どこか 1 つの球団に必ず所属していなければならない」といったルールがある場合には，ある選手から見て，所属球団の不在は許されない．そのような場合は「<--」の代わりに「1--」を,「-->」の代わりに「--1」を使用することができる．両者の違いは，その RDB 実装において依存元の依存先に向けての外部キーに非 NULL 制約が設定されるか，されないかの違いである（図 7.13）．

```
create table 球団s (
    id    integer primary key auto_increment,
);
create table 野球選手s (
    id    integer primary key auto_increment,
    所属球団id       integer not null,
    foreign key(所属球団id) references 球団s(id)
);
```

図 7.13: 参照従属関連の実装例（参照先の多重度が 1 の場合）

【備考】 上記の例は，所属期間，年俸など，所属にまつわる諸属性を管理したい場合などは，参照従属関連としてではなく，所属そのものをエンティティとして切り出し，野球選手と球団にそれぞれ存在従属関連を定義して,「[野球選手](所属野球選手)<==[所属]{開始日，終了日，年俸}==>(所属球団)[球団]」のようにモデル化する必要がある．

(5) 汎化関係（Generalization）および特化関係（Specialization）

【表記】 汎化関係および特化関係は「<|--」 および 「--|>」の記号を用いて表記する．マイナス記号が置かれる側に特化されたエンティティ（サブクラス）を，反対側に汎化されたエンティティ（スーパークラス）を書く．

【適用】 この関係は,「サブクラスはスーパークラスの一種である」，という文章が成り立つ場合に限って適用可能である．

【用例と実装指定】 たとえば,「[個人顧客]--|>[顧客]」，または「[顧客]<|--[個人顧客]」という表記は,「顧客」がスーパークラスで「個人顧客」がサブクラスであることを表現する．なお，DSL4EDA では，１つのスーパークラスに対して，複数のサブクラスが存在する場合は，図 7.14 のように複数行に分けて記述する必要がある．

```
[法人顧客] -|> [顧客]  または  [顧客] <|- [法人顧客]
[個人顧客] -|> [顧客]  または  [顧客] <|- [個人顧客]
```

図 7.14: DSL4EDA における複数のサブクラスが共通のスーパークラスを持つ場合の記法

汎化および特化の関係を RDB にて実装する場合には，スーパークラスの主キーをサブクラス側の主キー兼外部キーとして共有する形にする．

この例では，図 7.15 のようにする．

以上，DSL4EDA の主だった文法と言語要素，および DDL への変換時の実装指定について，順に説明してきたが，表 7.2 にエンティティ間の依存関係についてまとめておく．

7.3.3 PlantUML との比較

ここで，DSL4EDA の類似表記についても確認しておきたい．UML の世界では，PlantUML というオープンソースのツールが 2009 年頃から存在する [80]．PlantUML もプレーンテキストでモデルを表記するため，表記されたモデルの見た目は DSL4EDA のそれとよく似ている．

```
create table 顧客s (
    id    integer primary key auto_increment,
    # 顧客の属性群
);
create table 法人顧客s (
    id    integer   primary key,
    # 法人顧客特有の属性群
    foreign key(id) references  顧客s(id)
);
create table 個人顧客s (
    id    integer   primary key,
    # 個人顧客特有の属性群
    foreign key(id) references  顧客s(id)
);
```

図 7.15: 汎化関係の実装例

しかしながら，前者がプレーンテキスト言語からグラフィカルベースの UML ダイアグラムを作成できるツールであるのに対して，後者は，データベース定義およびそれにアクセスための API の仕様を生成できるツールであり，立脚しているモデル要素の意味論も前者は UML[35] のそれであるのに対して，後者は，インスタンスと属性値および，それらの存在期間の時間的前後関係に由来する制約であるため，両者は，本質的に異なる言語である．

表 7.2: DSL4EDA で表記するエンティティ間の依存関係

関係の種類	意味	依存先（参照先）エンティティのインスタンスの挿げ替え	提案DSLにおける表記	依存先への外部キーへの制約
属性従属関連	依存元のエンティティは依存先のエンティティの属性であるが，構造をもった繰り返し属性項目であるため，別のエンティティとして分離されている	不可	[依存先]<>=[依存元] または [依存元]=<>[依存先]	参照整合性制約，非ナル制約，ユニーク制約，連鎖削除を設定する
存在従属関連	依存元エンティティのインスタンスは依存先エンティティのインスタンスの先立つ存在を前提条件として存在可能である	不可	[依存先]<==[依存元] または [依存元]==>[依存先]	参照整合性制約，非ナル制約，依存先が2つある場合は，その組合せについてユニーク制約を設定する
参照従属関連	参照元のエンティティのインスタンスと参照先のエンティティのインスタンスの存在期間は独立しており，参照先インスタンスの不在を許す	可	[依存先]<--[依存元] または [依存元]-->[依存先]	参照整合性制約，非ナル制約を設定する
	参照元のエンティティのインスタンスと参照先のエンティティのインスタンスの存在期間は独立しており，参照先インスタンスの不在を許さない	可	[依存先]1--[依存元] または [依存元]--1[依存先]	参照整合性制約を設定する
汎化・特化関係	「[サブクラス]は[スーパークラス]である」，という文が成り立つ．スーパークラスの主キーはサブクラスでも共用され，それがスーパークラスのインスタンスへの外部キーを兼ねている	不可	[スーパークラス]<|-[サブクラス] または [サブクラス]-|>[スーパークラス]	参照整合性制約，非ナル制約，ユニーク制約，連鎖削除を設定する

7.3.4 DSL4EDA コンパイラの試実装

本提案の目的は，単に概念モデルをコンパクトに表記できることではない．DSL にて記述された概念モデルを元に，データベースを構築した上で，クライアントから，それらに容易にアクセスできるようなインターフェースの提供までを一貫して行うことで，開発を支援することである．

そのためには，DSL4EDL で表記されたモデルを読み込んで，基幹 DB（RDB）のためのスキーマ定義，履歴管理 DB（DDB）のためのコレクション定義，さらに，生成される DB 群（以下，リソースと記す）にアクセスするための API 仕様を出力するようなコンパイラが提供されてしかるべきであろう．

そこで，著者は，DSL4EDA コンパイラを試実装した．試実装は，DSL4EDA の実現可能性の確認も兼ねている．いかに使いやすい DSL の文法を定義できたとしても，それが実装できなければ，DSL の価値がなくなるためである．図 7.16 にコンパイラの入力と出力を示す．なお，コンパイラが生成するコード群は，第 7.1 節の図 7.1 で示したアーキテクチャの下で使用することを想定しており，Web サービスの設計・実装者，サーバ APP の設計・実装者，およびクライアント APP の設計・実装者のそれぞれに価値が提供できることを意図している．

Web サービスの実装者は，コンパイラが出力した API 仕様を参照しながら，公開する Web サービスのメソッド部を実装していくことができる．そして，その時点では，RDB 用のスキーマ定義（SQL の DDL 文），および履歴管理用 DDB のコレクション定義が提供されている．

一方，サーバ APP の設計・実装者，およびクライアント APP の設計・実装者は，コンパイラが出力した API 仕様文書を参照しながら，アプリケーションのビジネスロジック部分を実装していけばよいことになる．ビジネスロジック部分の実装時に必要なデータの操作は，Web サービスに委譲されるため，ビジネスロジックの実装に専念することが可能となる．

DSL4EDA コンパイラの試実装は Ruby 言語で行い，パーサ部の実装要素技術として，Treetop[81] を用いた．本節では DSL4EDA の文法を拡張 BNF で記載しているが，試実装では，これを PEG 形式 [82] に直した後，Treetop に読込ませることで，パーサを得ることにした．そのため，トークンの切り出しやトークンタイプの解釈といった処理を自前で実装する必要がないため，少ないステップ数で実装できた．以下に，コンパイラの処理内容を説明する．

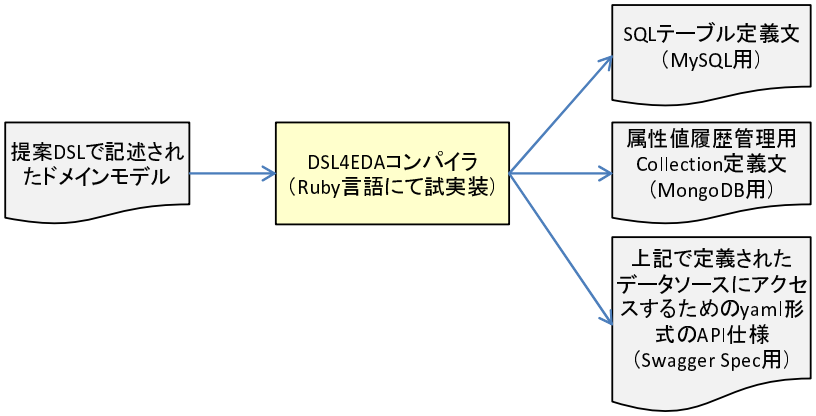

図 7.16: DSL4EDA コンパイラの入力と出力

(1)　DSL から SQL テーブル定義文の出力

ここで，コンパイラが行うべき処理は，DSL に記述されたそれぞれのエンティティの定義を，エンティティと 1 対 1 で対応する，RDB テーブルのスキーマ定義（CREATE TABLE 文），および，有向関連が表記された DSL の各行に対応する仮想表のスキーマ定義（CREATE VIEW 文）を生成して出力することである．

CREATE TABLE 文の生成では，テーブルごとに，属性並びで指定されたカラム定義に加えて,「id」という名称の主キー属性を追加し，属性の型は，7.3.2 節の (3) 項の記述に従って定義する．また，当該エンティティに依存先のエンティティがある場合には，7.3.2 節の (4) 項および (5) 項にて紹介した，依存関係の種類に応じた実装指定に基づき，外部キー制約及びカラム制約を追加する．さらに，アスタリスク（*）が付与されて履歴管理が指定された属性については，後で述べる DDB のコレクションにおける履歴管理のドキュメント ID を保持するための属性を数値型の属性として追加しておく．

CREATE VIEW 文の生成では，DSL に定義された依存関係を読み込む都度，依存元テーブルの外部キーと依存先テーブル主キーの一致を内部結合の条件とした SELECT 文を，仮想表の中味として生成する．その際，射影する属性は，依存元のテーブルの属性群と依存先のテーブルの属性群の和集合をものである．

なお，テーブル，仮想表を問わず，その名称は，7.3.2 節 (1) 項の記述に従って，DSL に表記されたエンティティ名または被役割名を複数形化（英語の場合は，ruby のライブラリを用いて，日本語の場合は，半角英小文字の's' をエンティティの識別名に追加）したものを使用する．

(2)　DSL から属性値履歴管理用コレクション定義の出力

ここで，コンパイラが行うべき処理は，DSL 上で履歴管理が指定された属性毎に DDB 用のコレクションを作成することである．試実装では DDB 製品として，MongoDB を採

用することにした．コレクション名は，エンティティ名と履歴管理する属性名を連結した名称である．7.3.2 節 (3) 項で述べた通り，コンパイラは，履歴管理が必要な属性を検出した時点で，所定の DDB（デフォルトの名称は,「<ドメイン名>historiesdb」）に対して，db.createCollection 文を発行するスクリプトを出力して，コレクションの生成に備える．

(3) DSL からデータソースにアクセスするための API 仕様の出力

ここで，コンパイラが行うべき処理は，生成したテーブル毎に，そのテーブルに対してクライアントとなるアプリケーションからネットワーク越しにアクセスできるように CRUD 操作を行うための WebAPI 仕様（API 仕様）を生成することと，生成した仮想表についても同様に，個々の仮想表に対してクライアントから Read 操作を行えるように API 仕様を生成することである．そして，テーブルと仮想表をまとめてリソースと呼ぶ．

API 仕様の要素は，HTTP 操作，URL で指定される操作対象のリソース名，リクエスト時のパラメータ，およびレスポンスの内容で構成される．DSL4EDA コンパイラでは，URL の名称として，リソースがテーブルの場合は，エンティティ名を，仮想表の場合は被役割名を用いている．たとえば,「get /顧客」にて，顧客の一覧が取得でき,「get /注文顧客」とすると,（API 呼び出し時点で）注文を持っている顧客の一覧を取得できる，といった API 仕様を生成することができる．DSL4EDA コンパイラは，テーブルおよび仮想表に対して，それぞれにアクセスするための API 仕様を生成するが，表 7.3 にそれらの基本形を示す．

当然，API 呼び出し時に，時点を示すパラメータが追加指定された場合には，DDB 側もアクセスして，該当する時点の属性値を返却する API を用意しなければならない．具体的には，文献 [71] の第 4 章，リソース指向アーキテクチャの統一インターフェースに関するガイドラインに則って URL パタンを定義し，その呼び出し法のドキュメントをマークアップ言語にて生成する．試実装では，ドキュメント作成ツールとして Swagger[83] を採用したため，WebAPI の仕様は YAML 形式で書き出される．試実装では，著者は，過去に存在従属クラス図から RESTful Web サービスを生成する提案を行っていた [84] が，その実装を DSL4EDA のモデルを入力とするように改良したものを使用した．図 7.17 は DSL が出力した API 仕様を Swagger Editor で開いた様子である．

(4) DSL4EDA コンパイラのアルゴリズム

本節では，試実装コンパイラのアルゴリズムについて説明する．図 7.18 に DLS コンパイラ内部の処理フローを示す．

処理フローでは，まず最初に存在従属クラス図の DSL 表現を読み込み，内部的に隣接行列を生成する．以降の処理では存在従属クラス図の DSL 表現と生成した隣接行列の双

表 7.3: DSL4EDA が生成する API 基本形

操作対象	サービスの機能	キー値の採番	HTTP操作	Pathパターン	リクエストのボディ	レスポンスのボディ	レスポンス・ヘッダのステータス	備考
独立エンティティティ	インスタンスの新規登録	サーバ側で連番を自動採番	POST	エンティティ名	新規登録されるインスタンスの属性と属性値の表現	新規登録されたインスタンスの属性と属性値の表現	201 Created 415 Unsupported Media Type 404 Not Found	この操作は安全でもべき等でもない
	特定のインスタンスの照会	なし	GET	エンティティ名/{id}	空	該当したインスタンスの属性と属性値の表現	200 OK 204 No Content 404 Not Found	照会の操作はすべて安全かつべき等である 存在しない{id}が指定された場合はエラーとなる
	特定のインスタンスの編集	なし	PUT	エンティティ名/{id}	編集するインスタンスの属性と属性値の表現ただし，キー属性とその値はなくてもよいし，あっても無視される	編集されたインスタンスの属性と属性値の表現ただし，キー属性とその値はなくてもよいし，あっても無視される	200 OK 204 No Content 415 Unsupported Media Type 404 Not Found	編集の操作はすべて安全ではないが，べき等である 存在しない{id}が指定された場合はエラーとなる
	特定のインスタンスの削除	なし	DELETE	エンティティ名/{id}	空	削除されたインスタンスの属性と属性値の表現	204 No Content 500 Internal Server Error 404 Not Found	削除の操作はすべて安全ではないが，べき等である 存在しない{id}が指定された場合，または，そのインスタンスを参照しているオブジェクトが存在する場合はエラーとなる
独立エンティティティ・従属エンティティ共通	指定範囲のインスタンスの照会	なし	GET	エンティティ名/{from}/{to}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	最初のインスタンスの位置を0番目とし，fromで指定された位置のインスタンスからtoで指定された位置のインスタンスまでが含まれる
	すべてのインスタンスの照会	なし	GET	エンティティ名/	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である
	インスタンスの件数照会	なし	GET	エンティティ名/count	空	件数	200 OK 404 Not Found	件数を照会する操作はすべて安全かつべき等である
	すべてのインスタンスの削除	なし	DELETE	エンティティ名/	空	削除されたインスタンスの件数	200 OK 404 Not Found	削除の操作はすべて安全ではないが，べき等である
従属エンティティ	インスタンスの新規登録	クライアント側で既知のためクライアント側が指定	POST	エンティティ名 /{id1}+{id2}+{id3}+…+{idn}	新規登録されるインスタンスの属性値の表現	新規登録されたインスタンスの属性と属性値の表現	201 Created 415 Unsupported Media Type 404 Not Found	この操作は安全ではないがべき等である
	特定のインスタンスの照会	なし	GET	エンティティ名 /{id1}+{id2}+{id3}+…+{idn}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	照会の操作はすべて安全かつべき等である
	特定の親に従属するインスタンスの照会	なし	GET	エンティティ名 /{id1}+{id2}+{id3}+…+{id(n-1)}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	複合識別子の属性がn個の場合，識別子は最低1個からn-1個までを+記号で連結して指定できる．途中の識別子の省略はできない
	特定のインスタンスの編集	なし	PUT	エンティティ名 /{id1}+{id2}+{id3}+…+{idn}	編集するインスタンスの属性と属性値の表現ただし，キー属性とその値はなくてもよいし，あっても無視される	編集されたインスタンスの属性と属性値の表現ただし，キー属性とその値はなくてもよいし，あっても無視される	200 OK 204 No Content 415 Unsupported Media Type 404 Not Found	編集の操作はすべて安全ではないが，べき等である 存在しない{id}が指定された場合はエラーとなる
	特定のインスタンスの削除	なし	DELETE	エンティティ名 /{id1}+{id2}+{id3}+…+{idn}	空	削除されたインスタンスの属性と属性値の表現	200 OK 204 No Content 404 Not Found	編集の操作はすべて安全ではないが，べき等である 存在しない{id}が指定された場合はエラーとなる
仮想表(ビュー)	特定のインスタンスの照会	なし	GET	仮想表名/{id1}+{id2}+{id3}+…+{idn}	なし	該当したインスタンスの属性と属性値の表現	200 OK 204 No Content 404 Not Found	仮想表の名称は隣接行列のエンティティ名をfrom, toの順に結合したもの
	特定の親に従属するインスタンスの照会	なし	GET	仮想表名/{id1}+{id2}+{id3}+…+{id(n-m)}	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	複合識別子の属性がn個の場合，識別子は最低1個からn-1個までを+記号で連結して指定できる．途中の識別子の省略はできない
	指定範囲のインスタンスの照会	なし	GET	仮想表名/{from}/{to}	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	最初のインスタンスの位置を0番目とし，fromで指定された位置のインスタンスからtoで指定された位置のインスタンスまでが含まれる
	すべてのインスタンスの照会	なし	GET	仮想表名/	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である
	インスタンスの件数照会	なし	GET	仮想表名/count	なし	件数	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である

方を参照しながら処理を進める．

(a) 隣接行列について　存在従属クラス図は，依存元のクラスから依存先のクラスへの誘導可能性を持った有向グラフであるため，生成すべき隣接行列は対角成分に対して非対称な正方行列となる．また，関係の両端のエンティティを始点（from）と終点（to）の役割で区別する必要がある．隣接行列の行と列にそれぞれエンティティを割り当て，各セルには，行のエンティティから列のエンティティに向かう関係の種類，すなわち，属性

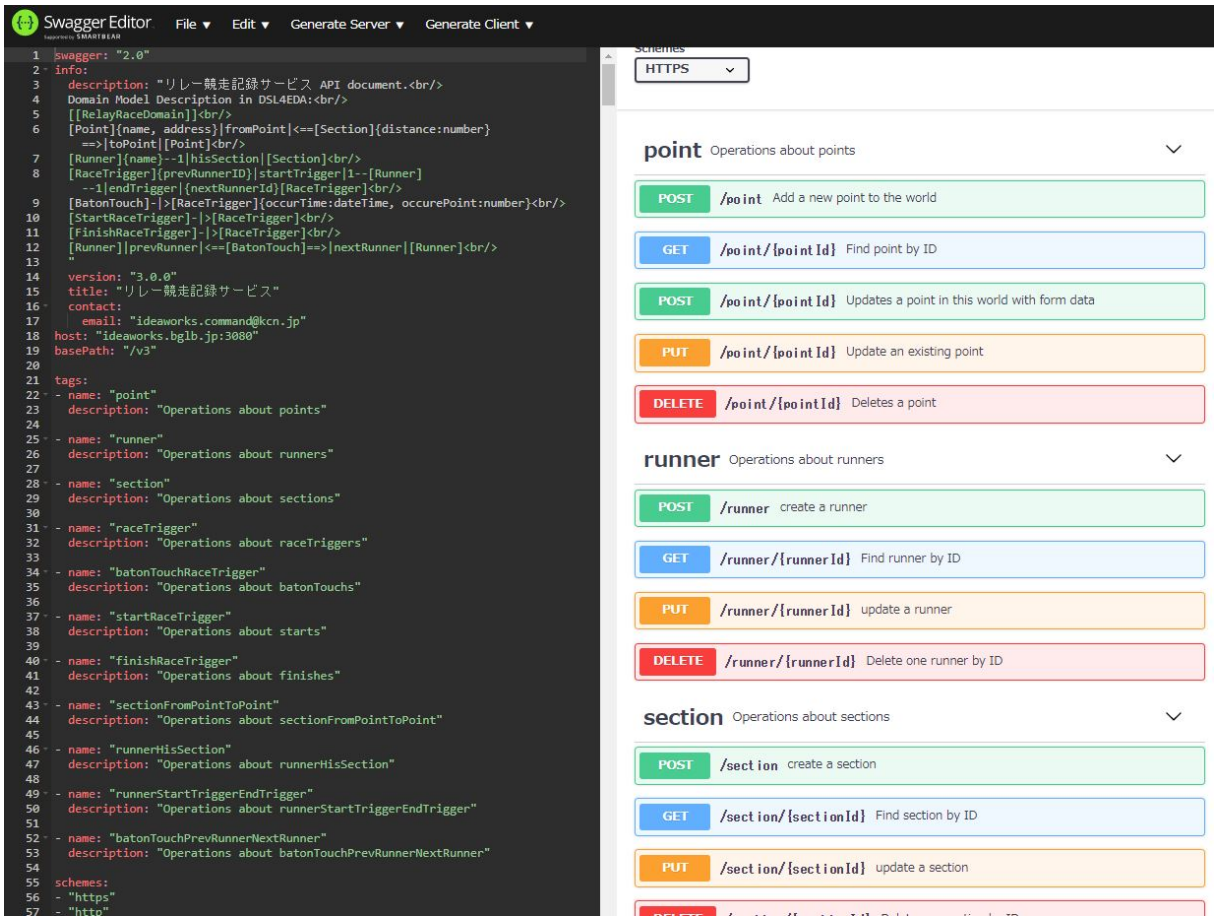

図 7.17: DSL が出力した API 仕様を Swagger Editor で開いた様子

従属関連，存在従属関連，参照従属関連，または汎化関係のいずれか，を記入する．なお，無関係の場合は空欄としておく．表 7.4 に図 7.2 の存在従属クラス図の DSL 表現（図 7.6）から生成される隣接行列を示す．

表 7.4: コンパイラ内部で生成される隣接行列の例

from \ to	顧客	法人顧客	個人顧客	注文	注文明細	商品	商品カテゴリ
顧客							
法人顧客	汎化関係						
個人顧客	汎化関係						
注文	存在従属関連	存在従属関連	存在従属関連				
注文明細				属性従属関連		存在従属関連	
商品							参照従属関連
商品カテゴリ							

隣接行列を走査することより，主キーの指定順が明らかになる．例えば，表 7.4 の 4 行目第 1 列のセルを見れば注文から顧客に向けて存在従属していることがわかる．インスタンスの存在期間は顧客が先でないといけないため，主キーの指定順も顧客の ID が先で，

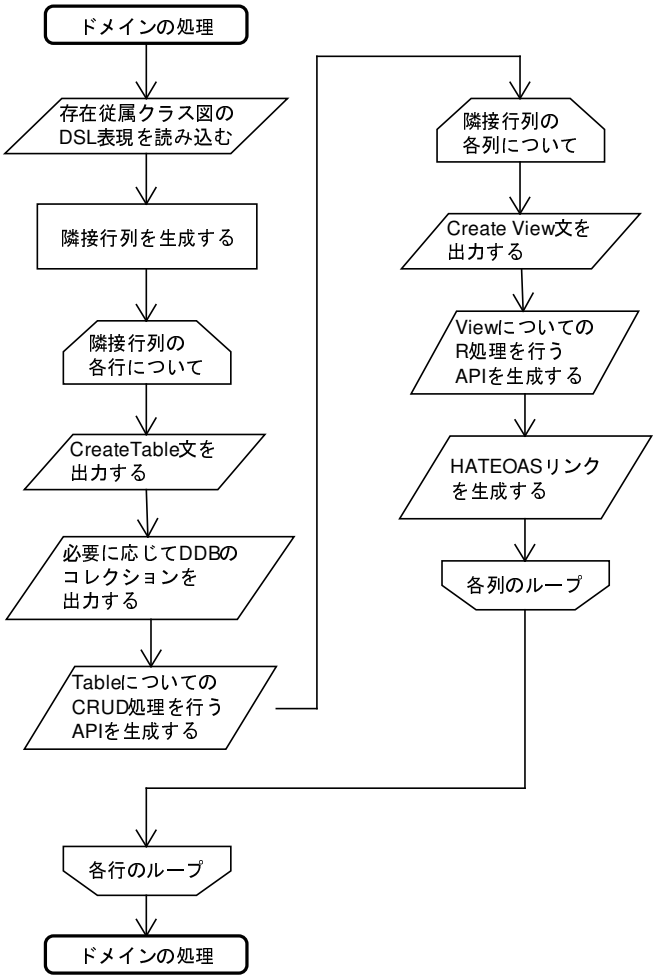

図 7.18: DSL4EDA コンパイラ内部の処理フロー

注文の ID は後であることが自然である．また，同じセルの情報により，注文から顧客への関連にアクセスするために被役割名が注文顧客である仮想表および，それにアクセスするための WebAPI も生成すべきことが理解される．

隣接行列は 2 次元の配列であるため，コンパイラアルゴリズムの制御構造は 2 重ループになっており，その外側がおのおののクラスに関する処理のループであり，その内側がクラス間の関連に関する処理のループである [2]．

(b) 外側ループの処理について 外側のループでは，DSL に定義されたエンティティについて，7.3.4 節で示した，(1) 項および (3) 項の処理を行う．エンティティ毎にそのエンティティクラスのインスタンスを RDB に格納するための実表の定義（Create Table 文），および，実表を CRUD 操作するための WebAPI 群を出力する．なお，履歴管理が必要な属性を持っているエンティティの場合は，(2) 項で述べた通り，ドキュメント指向 DB の

[2] もちろん，ドメインの定義ごとに繰り返し処理を行うため，正確にいえば，3 重ループ構造になるが，ここでの説明はドメインの内の処理アルゴリズムを対象としている．

コレクション定義も併せて出力する．

(c) 内側ループの処理について 内側のループでは，DSL に定義されたエンティティ間の関係について，7.3.4 節で示した，(1) 項および (3) 項の処理を行う．DSL コンパイラは隣接行列の各セルを走査し，セルの内容に応じて仮想表の定義（Create View 文），仮想表サービスの定義，および，HATEOAS [3] リンクの生成を行う．

(d) HATEOAS リンクの生成について DSL4EDA コンパイラが生成する HATEOAS リンクは直上の従属先クラスおよび直下の従属元クラスを対象に生成する．なぜならば DSL4EDA への入力は，存在従属クラス図の DSL 表現のみであるため，ツールが生成する API から見て知り得る状況は，

1) 当該リクエストが参照している URI で表現されるクラス名
2) 当該リクエストが参照している URI で表現されるクラスのすべてのインスタンス群
3) 当該リクエストが成功したのか，失敗したのかの情報
4) 現在のリクエストが注目している URI で表現されるリソースの従属先（参照先）のリソースのクラス名
5) 現在のリクエストが注目している URI で表現されるリソースの従属先（参照先）のリソースのインスタンス
6) 直下の従属リソースの URI 名
7) 直下の従属リソースのインスタンス群（自分と同じ識別子の値を識別子の一部として持つため）

の最大 7 種の情報である．5) と 7) については理論上はすべてのインスタンスを参照することも可能であるが，その実装は現実的ではない．なお，最大というのは，リクエストが独立リソースに対するものであった場合と，従属または関連リソースに対するものあった場合とでは得られる情報の種類数が変化するためである．

図 7.19 は，このような観点からエンティティ間の従属性からみた直上および直下のインスタンスの状態と，トリガの関係を状態遷移図に表わしたものである．DLS4EDA ではこの図をガイドラインとして，実験的にリンクの生成を行っている．

[3]HATEOAS（Hypermedia as the Engine of Application State）とは，REST を拡張する概念で，すべてのエンドポイントは，処理の流れの中でリンクとしてサーバから提供されるべきであり，API がクライアントに返すデータ（レスポンス）の中に，次に行う行動，取得するデータ等の URI をリンクとして含めることで，そのデータを見れば，クライアントが次にどのエンドポイントにアクセスすればよいかがわかるような設計 [85] である．

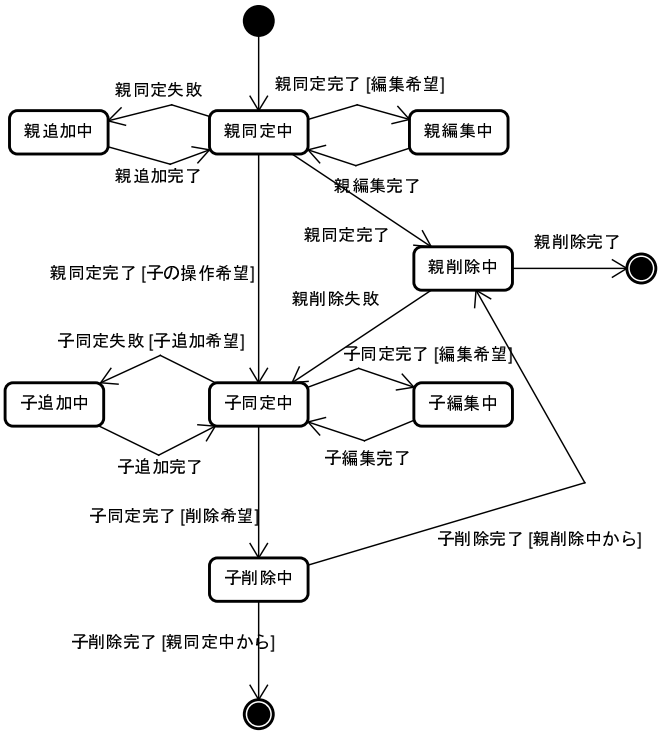

図 7.19: HATEOAS リンク生成のためのコンパイラの状態遷移

7.4 評価実験

7.4.1 実験の概要

DSL4EDA の記述性を確認するために，表 7.5 に示す 7 つのモデリング題材[4] に対して，存在従属分析を行い，その分析結果を DSL4EDA の文法で問題なく記述できることを確認した．

本節では，その中から，DSL4EDA で扱うほとんどの関係種が登場する，表 7.5 中の題材番号 5 番の題材を採り上げて説明する．図 7.20 は，題材番号 5 番の概念モデルに反映すべき要件のリストである．そして，図 7.21 は，図 7.20 の要件に対して存在従属分析を行った結果を，DSL4EDA で表記したモデルである．DSL4EDA で意図した通り，簡潔に題材の概念モデルを記述することができた．次節では，図 7.20 から図 7.21 を導出した過程を説明する．

[4]表 7.5 中の題材は，文献 [86]，[87]，[88] のようなデータベースやモデリングの教科書に記載された題材，および，著者が大学の講義や勉強会で使用している題材である．

表 7.5: 提案 DSL の記述性を確認するために用いた題材の一覧

題材番号	題材の概要
1	学生が科目を履修し，教員が科目を担当し．科目が参考書を指定するドメイン
2	日々のフライト運航便へパイロットと客室乗務員を割り当てるドメイン
3	バントはメンバーで構成され，バンドは，パートを要請し，メンバーはパートを担当するドメイン
4	提供科目に制約がある学級の，実施教室に制約がある科目に対して，登壇日に制約がある講師を割り当てるドメイン
5	町内リレーの走行区間に走者を割り当て，出来事を記録するドメイン
6	酒屋の在庫管理問題のドメイン
7	顧客に単品花で構成される花束を指定日に組み立てて出荷するドメイン

■ 走者には，名称がある：具体例：Aさん，Bさん，Cさん，など
■ 地点には，名称と所在地がある：具体例：
W地点，東京都港区1丁目，X地点，東京都世田谷区2丁目，など
■ ２つの地点の間を区間と定義した場合，区間は2つの地点なしでは，存在し得ない
■ 区間には，出発地点から到着地点への方向性がある（逆走できない），
よって区間からみた２つの地点を，それぞれ出発地点，到着地点として区別する必要がある
■ 区間が決まれば，距離が定まる，ただし，直線距離ではなく地点を結ぶ道の距離とする
■ 区間は，地点を介して隣接する，すなわち，スタート地点からゴール地点に向けて，
前の区間と次の区間が高々１つ存在する
■ 走者には必ず1つの区間が割り当てられる（そして走者はその区間を走行する）
■ 走者は，区間の始点と終点において必ずリレーのトリガ（競技トリガ）を実施する，
すなわち，つぎのいずれかである：
・走者はスタートして，受け持ち区間を走りきり，
次の走者にバトンタッチするか（この場合，前の走者は存在しない）
・走者は前の走者からバトンタッチを受けて，受け持ち区間を走りきり，
次の走者にバトンタッチするか（この場合，前の走者と次の走者がいずれも存在する）
・走者は前の走者からバトンタッチを受けて，受け持ち区間を走りきり，
ゴールするか（この場合，次の走者は存在しない）
■ イベントには，それが起こった地点（発生地点）と発生時刻が帰属する．
そして，それらはリレーの事象として，逐一記録したい

図 7.20: 評価実験のためのモデリング題材

7.4.2　業務記述からの導出過程

（ア）エンティティとその属性を識別した

例題の問題記述から，業務で捕捉，蓄積，管理すべき概念をエンティティとして切り出し，そのインスタンスと存在期間（管理期間）が等しい値をそのクラスの属性として識別する．この例題の場合,「走者」,「地点」,「区間」,「スタート」,「バトンタッチ」,「ゴール」，そして「競技トリガ」がエンティティである．そして,「走者」の「氏名」,「地点」の「名称」と「所在地」,「区間」の「距離」，そして,「スタート」,「バトンタッチ」,「ゴール」の「発生時刻」，および「競技トリガ」の「発生時刻」,「発生地点」が，それぞれのエンティティのインスタンスと存在期間が等しいため，それぞれの属性として識別する．なお，属性値として識別された値の中で，反復的に記述される値がある場合には，その

```
[[リレー競技ドメイン]]
[地点]{名称, 所在地}(出発地点) <== [区間]{距離} ==> (到着地点)[地点]
[走者]{氏名} --1 (担当区間)[区間]
[競技トリガ]( 走行開始トリガ )1-- [走者] --1 ( 走行終了トリガ )[競技トリガ]
[バトンタッチ] -|> [競技トリガ]{発生時刻:dateTime} ==>(発生地点)[地点]
[スタート]     -|> [競技トリガ]
[ゴール]       -|> [競技トリガ]
[走者](前走者)<==[バトンタッチ]==>(後走者)[走者]
```

図 7.21: 提案 DSL による分析結果の表記

値を，別のエンティティのインスタンスとして分離し，後続のステップで属性従属関連を定義することになる．なお，この例題の場合は，反復的な繰り返し属性は現れなかった．

（イ）エンティティ間の依存関係を識別した

次に，そのインスタンスは，前提なしで存在できるのか，あるいは，他のインスタンスの先立つ存在を前提として存在し得るのかなどを，業務要件に照らして検討し，2 つのインスタンス間の依存関係を識別していく．識別すべき依存関係は次の 1) から 4) のいずれかであり，識別作業は順不同で行う．

1) 属性従属関連を識別すること

２つのエンティティのインスタンス間に，例外なくライフサイクルの一致が認められる場合は，それを属性従属関連として識別することができる．エンティティの属性は先の（ア）にて識別済みであるが，反復的な属性を識別した結果，別エンティティのインスタンスとして分離していた場合は，属性従属関連を定義し，分離したエンティティから元のエンティティに向けて属性従属関連を定義する．

この例題の場合は,「名称」と「所在地」を「地点」クラスの,「氏名」を「走者」クラスの,「距離」を「区間」クラスのそれぞれ属性とした．

2) 存在従属関連を識別すること

２つのエンティティのインスタンス間に，依存元エンティティのインスタンスが存在できるためには，依存先のエンティティのインスタンスの先立つ存在が前提であり，かつ，依存元エンティティのインスタンスの存在期間を通じて，依存先のエンティティのインスタンスは消去も挿げ替えも行わないような関係が認められる場合は，それを存在従属関連として識別することができる．

この例題の場合,「区間」は 2 つの「地点」に存在従属する，と認識できたため,「区間」から「地点」に向けて．存在従属関連を定義した．その際，リレーの区間は，出発地点から到着地点に向けての方向性があるため，それぞれの地点について,「出発地点」,「到着地点」と呼ぶ被役割を定義した．また,「スタート」,「バトンタッチ」,「ゴール」につい

ても,「地点」に存在従属すると認識される．さらに,「バトンタッチ」は，前走者と後走者の存在が前提として発生する出来事であるため,「バトンタッチ」を,「前走者」と「後走者」という被役割を定義した「走者」の間に定義し,「バトンタッチ」からそれぞれの走者に向けて存在従属関連を定義した.

3) 参照従属関連を識別すること

２つのエンティティのインスタンス間に，依存元エンティティのインスタンスと依存先エンティティのインスタンスのライフサイクルは独立しており，依存元エンティティのインスタンスの存在期間を通じて，依存先のエンティティのインスタンスを消去したり挿げ替えたりしたい関係が認められる場合には，それを参照従属関連として識別することができる.

この例題の場合，走者には必ず 1 つの区間が割り当てられるため,「走者」から「区間」に向けて，多重度が必ず 1 の参照従属関連を定義する．また，すべての走者は，自分の受け持ち走行区間の始点と終点において，必ず 1 回ずつのリレーのトリガ（競技トリガ）,これには,「スタート」,「バトンタッチ」，あるいは「ゴール」というバリエーションがある，を経験するため,「走者」から「走行開始トリガ」および「走行終了トリガ」という被役割を定義した「競技トリガ」に向けて，多重度が必ず 1 の参照従属関連を定義した.

4) 汎化関係を識別すること

２つのエンティティクラス間に,「サブクラスはスーパークラスの一種である」，という文章が成り立つ場合は，それを汎化関係として識別することができる.

この例題の場合,『「スタート」は「競技トリガ」の一種である』,『「バトンタッチ」は「競技トリガ」の一種である』，そして,『「ゴール」は「競技トリガ」の一種である』，という文章がいずれも成り立つため,「スタート」から「競技トリガ」へ向けて,「バトンタッチ」から「競技トリガ」へ向けて，そして「ゴール」から「競技トリガ」へ向けて，それぞれ汎化関係を定義するとともに，それまでサブクラス側の属性および関連であった発生時刻，および発生地点をスーパークラス側に移動した.

以上のような分析を行うことによって，要件についての記述を，DSL4EDA に変換できる．評価実験では，表 7.5 のその他の題材についても，同様な分析を行い DSL4EDA を用いて概念モデルを簡潔に記述できることを確認した.

7.5 結言

本章では，存在従属分析によって得られた概念モデルの記述に特化した DSL を提案し，その文法とモデル要素の意味，およびモデルを RDB の DDL スキーマに変換する際の実装指定を説明した．エンティティの主キーを ID と定めて，エンティティ間の関係から自動的に外部キーが設定されるようにする，などで，表記を大幅に簡素化しながらも，後続の開発作業で必要となる，リソース定義へと結びつけていくことに成功している．

そして，データベースの教科書等に記載されている題材を評価実験のためのモデリングの題材として，その記述性について確認した．結果，すべての題材について，存在従属分析結果を DSL4EDA で表記でき，同時に試実装のコンパイラで意図した通りの出力結果が得られることを確認した．これにより，DSL4EDA の文法は，パーサにとっても無理のない構文であることを確認できた．記述性と可読性と機械処理可能性をすべて満たすのは一般に難しいことであるが，DSL4EDA はそれらを評価できる水準で満たしていると思われる．

概念モデルを DSL4EDA の様式で記述しておくことにより，その後の開発作業を機械化しやすい．DSL4EDA はテキストエディタとコンパイラさえあれば，存在従属分析手法に基づいたドメインモデルを手軽に作成・編集し，データベース定義とアクセスメソッドを瞬時に手に入れて，設計者間や開発環境間で共有できることが最大の売りである．

DSL4EDA コンパイラは試作段階ではあるが，この実現には，最新のソフトウェア工学の成果に依るところが大きい．Ruby などのプログラミング言語や treetop や Swagger といった優れたツールが存在しなかったならば，このような DSL を試実装することさえできなかったと思われる．このような優れた技術開発に日々取り組んでおられるコミュニティの方々に深く感謝したい．

今回の評価実験の題材は教材の域を出ない小さなものであったため，今後は，DSL4EDA をさまざまな分野の業務における概念設計に適用しながら，その文法を改良していくとともに，コンパイラの機能や品質を向上させてゆきたいと考えている．

第8章　結論

本論文では，我が国では，情報システムの開発を対象に，概念モデルが広くSE（ソフトウェアエンジニア）の間に普及しているとは言い難い原因が，概念モデルの入力となる要求記述，即ち，業務課題やシステムに関する要望の論述が日本語である点に起因するとの独自の問題意識の下で，日本語要求記述を出発点とし，より良い概念モデルが構築できるように，また構築した概念モデルが有効に活用できるように，そのための方法について提案可能なモデリング手法，モデル化のためのガイドライン，そして，モデル化対象に適した設計成果物の表記を確立することを目的とした．

本研究の特徴を，従来の開発プロセスと対比することで説明する．図8.1は，従来の開発プロセスのイメージである．勿論，開発プロセスは，企業や組織，また開発対象により異なるため，一概には言えないとしても，図8.1は，OOSE[36]に端を発し，RUP[58]に組み込まれ，ICONIX[51]へと発展した開発プロセスの特徴を示したものである．図8.1の開発プロセスは，ユースケースモデルあるいは，ユースケース記述を中心に，システム開発を行うためユースケース駆動型の開発プロセスと呼ばれる．

ユースケース駆動型の開発プロセスの特徴として，システムで提供すべきサービス（ユースケース）を単位として，分析・設計・実装を行うため，ユースケース単位でリリースして行ける反面，作成されるエンティティのクラス図が，ユースケース毎に局所最適化されたものになる恐れがある．また，機能規模の見積が，ユースケース記述の作成後に行われるため，フィードバックが遅れる，さらに，識別されたユースケースが網羅的であるという保証がない，といった問題点がある．

一方，図8.2は，本論文で紹介した種々の手法を取り入れた開発プロセスである．要求記述から，エンティティの存在従属クラス図を先に作成し，それを中心にシステム開発を行うため,「存在従属クラス図駆動開発」と命名したい．存在従属クラス図駆動開発は,文献[22]で提唱されている，ドメイン駆動設計の一種であるとの見方ができる．しかしながら，文献[22]では，何に着目してクラスや属性や関連を識別するか，という着眼点の代わりに，ドメインモデリングのパターンとして，40ものパターンを示し，500ページを超える分量になっている．佐藤は，データベース設計論[89]の中で,「モデルを作る際，個体を認知するための判断基準がなければ,「例示する」しかない.」と述べているが,例示の多さに比例して，ページ数が増加している．

存在従属クラス図駆動開発プロセスの特徴として，要求記述から先にドメインモデル

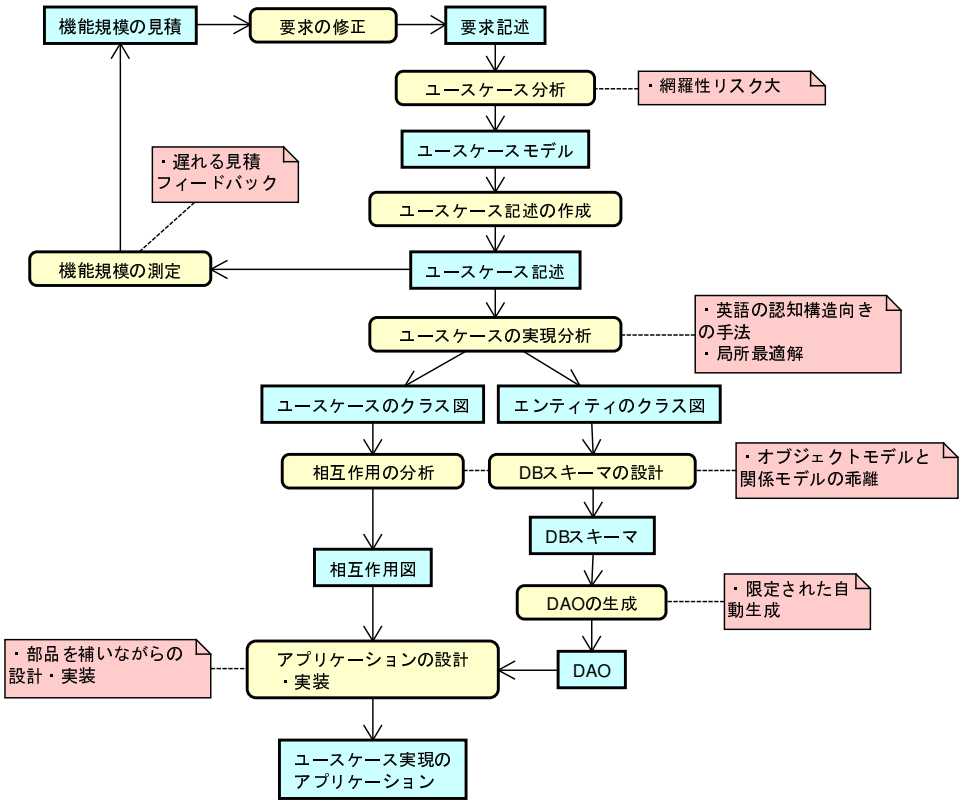

図 8.1: 従来の開発プロセスのイメージ

を構築するため，構築されたドメインモデルは，個々のユースケースに局所最適化されたものとはなりえないこと．要求記述に現れるエンティティの存在従属性に着目して，クラス，属性，関連の識別を行うため，存在文が多用される日本語の要求記述の分析に適していること．ユースケースは，ドメインモデルから生成するため，網羅性が高いこと．機能規模の見積もりも，ドメインモデルから測定するため，システム開発の依頼者へのフィードバックが早期に行えること．ドメインモデルから，実装に必要な多くの設計要素を自動生成するため，分析以降の開発作業が楽になること．また，要求記述が変更され，ドメインモデルが修正されても，自動生成のやり直しが行えること．などが挙げられる．

以上から，本研究の特徴として，(a) 概念モデルの構築の際に，行為文よりも状況描写文が頻出する日本語の特性を考慮に入れていること，(b) ドメインモデルとしての存在従属クラス図を，ドメインオブジェクトの存在に由来する時間的前後関係の制約を含意した動的モデルと位置づけていること，(c) 多次元尺度構成法を応用した定量的な概念モデルの妥当性の確認を含めていること，(d) 概念モデルの構築だけでなく，その有効活用も含めた包括的な提案していること，を挙げることができる．以下では，これらを章ごとに振り返る．

第 2 章では，業務についての記述（日本語要求記述）をもとに，概念モデルとしてのイベント応答モデルを導く方策について論じている．具体的には，行為者から飛来する，

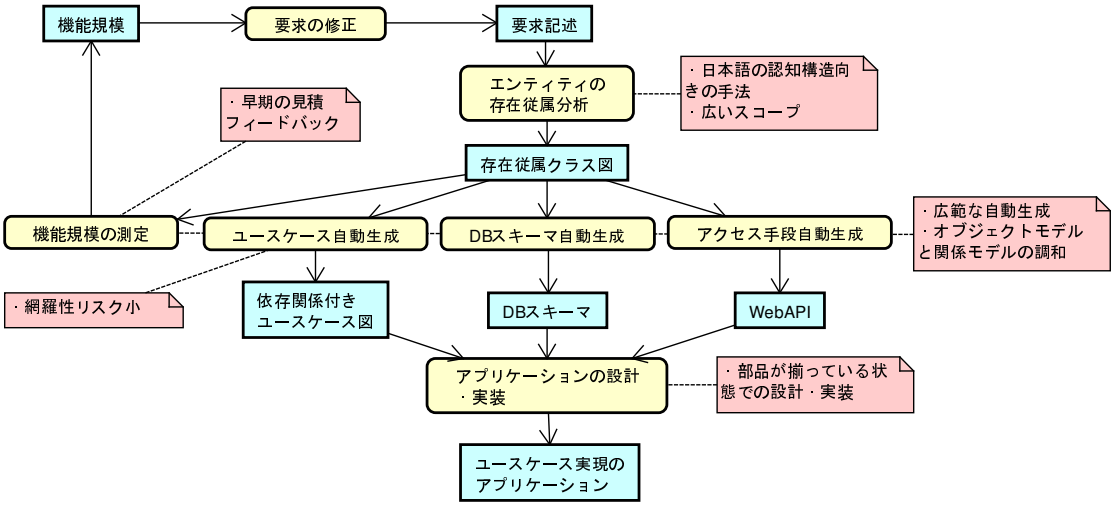

図 8.2: 本研究の成果を取り入れた開発プロセスのイメージ

業務プロセス起動の契機に対して，業務で行うべき処理や応答を，認知言語学の成果の一つである，ビリヤードボールモデルを援用して考案した様式に則って形式化する手法について論じている．これにより，日本語要求記述の中でも注目すべきは，動態プロセス（そのプロセスの結果，世界の状態が変化するプロセス）であることを明らかにし，業務プロセスを，ビジネスイベントを単位に，概念的に独立した単位として切り分け，分割されたプロセスについて，参与者（ドメインオブジェクト）と，それぞれの状態変化を抽出して，形式的に記述できることを示した．

第 3 章では，システム化の適用領域を，ドメインオブジェクト間の存在従属性に着目して，モデル化する研究の成果について論じた．本章では，これを存在従属分析と呼び，その成果物を存在従属クラス図と称する，UML のクラス図表記をカスタマイズした表記にてモデル化する提案した．存在従属分析のガイドラインは，母語の文法に依存せずに適用できる上，この手法により，正規化理論を陽に使うことなしに，第 4 正規形以上の正規化レベルを有するドメインモデルが得られることを確認した．これにより，存在従属クラス図は，十分に正規化された実体関連図（ER 図）と等価である，と言えるようになった．

第 4 章では，ドメインモデルを責務（ドメインオブジェクト群，および，それらの属性群と操作群）の 2 次元空間布置と見做した上で，既存手法によって得られたドメインモデルと存在従属分析によって得られたドメインモデルを比較している．既存手法の代表として，ソフトシステムズ方法論に着目し，概念活動モデルから知識の責務と振る舞いの責務を抽出するとともに，責務間の関連度合いを数量化する基準を提案している．既存手法によって得られたドメインモデルと存在従属分析によって得られたそれは，矛盾しない構造を有することを示した．

第 5 章では，存在従属クラス図を入力として，業務システムのユースケース群を自動生

成するとともに，同時にクラス間の依存関係を利用して，生成されたユースケース間の依存関係についても自動生成する研究について論じた．結果，使用性（ユーザビリティ）を向上させるユースケース以外は高い網羅性にて抽出できることを示した．

第 6 章では，存在従属クラス図から，それを扱う業務システムの機能規模を測定する研究について論じた．本章で提案している測定手法は，業界で広く用いられている COSMIC 法よりも，簡便でかつ，開発プロセスの早い段階で適用可能でありながらも，その測定結果は COSMIC 法を用いた測定結果と比較しても差がほとんどないことを示した．

第 7 章では，存在従属クラス図を入力として，データベースのスキーマを生成し，それらを業務リソースとして，リソースに対する包括的なアクセス手段を提供するサービスの仕様の生成について論じた．存在従属クラス図をテキストベースで記述可能な DSL の構文規則を提案し，7 つの題材について，その記述性を確認した．DSL で表記されたモデルを試実装のコンパイラで処理することにより，その後の開発作業で必要となる設計成果物，すなわち，データソースのスキーマ定義，定義されたデータソースにネットワーク越しにアクセスする手段を提供する Web サービスインターフェース群，およびそのドキュメントを自動生成できることを示した．このことにより，アプリケーション・インターフェース（API）の原型定義に規則性と網羅性がもたらされたことを報告した．

以上の研究成果より，日本語要求記述をもとに，日本語の言語特性を考慮しながら，ドメインモデルとしての存在従属クラス図を構築することが可能となった．さらに，構築した存在従属クラス図をもとに，DSL4EDA で表現することで，後続の設計・実装局面で必要となる設計成果物の自動生成手法を確立した．これらの技術は，ドメインモデルに重きを置いた機敏な情報システム開発ための有用なパーツとなる．存在従属クラス図をもとに，サーバサイドのデータリソースレイヤの部品の生成は可能となったが，一方，ユーザビリティを考慮したクライアントレイヤの画面群やその遷移のための部品の自動生成も可能であると考えられる．しかしながら，存在従属クラス図と UI の画面遷移の関係は，本研究では明らかとなっていない．今後は，その関係を明らかにすることで，UI を含めた自動生成が可能となるよう邁進していきたい．

我が国の業務システムの開発現場では，要求仕様のモデル（ドメインモデル）が不十分あるいは不適切であったために，失敗しているプロジェクトが多く存在していることから，本研究の成果が，日本語要求記述をもとに概念モデルを構築するプロセスを改善するとともに，改善された概念モデルを入力として，以降の開発で必要となる設計成果物の多くを自動生成することで，個別の業務システムから組織を跨るシステム間連携まで，その円滑な要求定義と実現に貢献することを願っている．

今後も，業務システム開発の生産性の改善のために，さまざま提案を行ったり，役に立つ便利なツールを開発したりしていきたいと考えている．

謝辞

本論文の執筆に際して，同志社大学 理工学部 金田重郎 教授には懇切多大なご指導とご鞭撻を賜りました．ここに深く感謝し，御礼申し上げます．

また，実際に研究を進めるにあたって，金田教授の率いる研究班の皆様やIT勉強宴会の皆様にも大変お世話になりました．とりわけ渡辺幸三氏，佐野初男氏には，勉強会や電子メールのやり取りを通じてさまざまなご助言やアイデアをいただきました．ありがとうございました．

最後に，私を支え，励ましてくれた家族に心から感謝いたします．

2019年（令和元年）7月8日

井田 明男

本論文に含まれる発表文献

□ 論文誌論文

[i] Akio Ida and Shigeo Kaneda. Applying Multidimensional Scaling for Responsibility Distribution among Objects. *Journal of information processing*, Vol.23, No.3, pp.318-326, 2015（第 4 章）.

[ii] 井田明男, 金田重郎, 熊谷聡志, 藤本明莉. 存在従属性に着目した論理要件ロバストなドメインモデルの作成―ドメインクラス図をユビキタス言語として用いるために. 情報処理学会論文誌, Vol.56, No.5, pp.1340-1350, 2015（第 3 章）.

[iii] 金田重郎, 井田明男, 酒井孝真, 熊谷聡志. 日本語仕様文からの概念モデリングガイドライン―行為文と関数従属性に基づくクラス図の作成. 電子情報通信学会論文誌, Vol.J98-D, No.7, pp.1068-1082, 2015（第 1 章, 第 2 章）.

[iv] 井田明男, 金田重郎, 熊谷聡志, 矢野寛将. エンティティの存在従属性に着目した機能規模の測定―アジャイルで網羅性の高い業務アプリケーションの見積り手法. 情報処理学会論文誌,Vol.57, No.5, pp.1399-1410, 2016（第 6 章）.

[v] 矢野寛将, 中西啓太, 井田明男, 金田重郎. 存在従属性に基づくユースケース図の自動生成手法. 電子情報通信学会論文誌, Vol.J100-D, No.1, pp.14-27, 2017（第 5 章）.

[vi] 金田重郎, 井田明男, 森本悠介, 劉湘涛, 上野洸史. 存在従属クラス図とバージョン管理に基づく正規化アプリケーション構成法. 情報システム学会論文誌, Vol.14, No.2, pp.39-56, 2019（第 7 章）.

[vii] 井田明男, 金田重郎, 森本悠介. エンティティの存在従属分析のためのドメイン特化言語, 情報システム学会論文誌, Vol.15 , No.1, 2019（第 7 章）.

□ ショートノート

[i] 井田明男, 金田重郎. 要求分析のための日本語版ビリヤードボールモデルの提案. FIT2013 RO-009 （FIT 査読付論文, 情報処理学会）, pp.115-120, 2013（第 2 章）.

□ 国際会議（フルペーパー査読）

[i] Shigeo Kaneda and Akio Ida. Understanding of Class Diagrams based on Cognitive Linguistics and Functional Dependency. *Proc. of SERA2014 (IEEE, International Conference on Software Engineering Research, Management and Applications)*, pp.587-591, 2014（第 2 章）.

[ii] Akio Ida and Shigeo Kaneda. Application of Multidimensional Scaling for Responsibility Distribution of Objects. *Proc. of SERA2014 (IEEE, International Conference on Software Engineering Research, Management and Applications)*, pp.580-586, 2014（第 4 章）.

[iii] Shigeo Kaneda, Akio Ida and Takamasa Sakai. Understanding Class Diagrams based on Cognitive Linguistics for Japanese Students. *Proc., of JCKBSE (Joint Conference of Knowledge-Based Software Engineering)*,CCIS, Springer, 2014, pp.77-86, 2014（第 2 章）.

□ 外部表彰

[i] 井田明男, 金田重郎, 森本悠介. 存在従属クラス図から RESTful Web サービスの生成. 第 12 回 情報システム学会 全国大会・研究発表大会, ベストペーパー特別賞, P014, 2016（第 7 章）.

参考文献

[1] Peter P. Chen. English Sentence Structure and Entity-Relationship Diagrams. *Information Sciences*, Vol. 29, No. 2, pp. 127 – 149, 1983.

[2] Peter P. Chen. English, Chinese and ER diagrams. *Data and Knowledge Engineering*, Vol. 23, No. 1, pp. 5 – 16, 1997. Natural Language for Data Bases (Workshop 1996).

[3] 金田重郎, 井田明男, 酒井孝真. 日本語仕様文からの概念モデリングプロセス－英語 7 文型と関数従属性に基づくクラス図の作成－. 研究報告情報システムと社会環境（IS）, Vol. 2014-IS-128, No. 3, pp. 1–8, 2014.

[4] 金田重郎, 井田明男, 酒井孝真, 熊谷聡志. 日本語仕様文からの概念モデリングガイドライン—行為文と関数従属性に基づくクラス図の作成. 電子情報通信学会論文誌, Vol. J98-D, No. 7, pp. 1068–1082, 2015.

[5] Ronald W. Langacker. *Foundation of Cognitive Grammar Volume II—Descriptive Application*. Stanford University Press, 1991.

[6] 川上誓作. 認知言語学の基礎. 研究社, 1996.

[7] 水口志乃扶, 小川暁夫, 定延利之. 受動文の形式と意味—ヴォイスの統合的研究に向けて. 国際文化学研究：神戸大学国際文化学部紀要, No. 1, pp. 1*–32*, 1994.

[8] John R. Taylor, 瀬戸賢一. 認知文法のエッセンス. 大修館書店, 2008.

[9] William Croft. *Syntactic Categories and Grammatical Relations—The Cognitive Organizations of Information*. The University of Chicago Press, 1991.

[10] 中村善太郎. もの・こと分析で成功するシンプルな仕事の構想法. 日刊工業新聞社, 2003.

[11] Michael Tomasello, 大堀 壽夫 (翻訳), 秋田 喜美 (翻訳), 古賀 裕章 (翻訳), 山泉 実 (翻訳). 認知・機能言語学—言語構造への 10 のアプローチ. 研究社, 2011.

[12] 金谷武洋. 日本語に主語はいらない. 講談社, 2002.

[13] 株式会社豆蔵. サンプル RFP 日経 IT プロフェッショナル—業務システム分析のための UML モデリング演習 Ver.1.2. http://itpro.nikkeibp.co.jp/NIP/modeling/NIP06041.pdf, 2004. (2017 年 2 月 8 日確認).

[14] 金谷武洋. 日本語文法の謎を解く—「ある」日本語と「する」英語. ちくま新書, 2003.

[15] 金田重郎, 世古龍郎. 認知文法に基づくオブジェクト指向の理解. 電子情報通信学会技術研究報告：信学技報, Vol. 111, No. 396, pp. 61–66, 2012.

[16] 金田重郎, 井田明男, 酒井孝真. 「クラス図は英語である」との観点に基づく仕様文・クラス図変換メソッドの提案. 電子情報通信学会技術研究報告：信学技報, Vol. 112, No. 419, pp. 13–18, 2013.

[17] 岡智之. 日本語研究への認知言語学の応用—多義語，特に格助詞を中心に. 東京学芸大学紀要，総合教育科学系, Vol. 58, pp. 467–481, 2007.

[18] 森山新. 認知言語学から見た日本語格助詞の意味構造と習得. ひつじ書房, 2008.

[19] 森山新. 研究領域への扉 (第 6 回) 認知言語学 (後編) 教授法研究の活性化のために—認知言語学からの提言, 第 22 巻. アルク, 2009.

[20] 一般社団法人 ICT 経営パートナーズ協会, 関隆明 (監修). 超高速開発が企業システムに革命を起こす. 日経 BP 社, 2014.

[21] Scott Ambler, 越智典子 (翻訳), オージス総研 (監訳). オブジェクト開発の神髄—UML 2.0 を使ったアジャイルモデル駆動開発のすべて. 日経 BP 社, 2005.

[22] Eric Evans, 今関剛 (監修), 和智右桂 (翻訳), 牧野祐子 (翻訳). エリック・エヴァンスのドメイン駆動設計 (IT Architects ’Archive ソフトウェア開発の実践). 翔泳社, 2011.

[23] 牛尾剛, 長瀬嘉秀. オブジェクト脳のつくり方—Java・UML・EJB をマスターするための究極の基礎講座. 翔泳社, 2003.

[24] 萩本順三, 不破康人, 福村真奈美. 最新オブジェクト指向技術応用実践—Java によるビジネスアプリケーション開発モデルと実装技法. エーアイ出版, 1998.

[25] Peter P. Chen. *The entity-relationship approach to logical data base design (Data base management: no. 6)*. Q.E.D. Information Sciences Inc., 1st edition, 1977.

[26] Monique Snoeck and Guido Dedene. Existence dependency—The key to semantic integrity between structural and behavioral aspects of object types. *IEEE Transactions of Software Engineering*, Vol. 24, No. 4, pp. 233–251, 1998.

[27] Raf Haesen, Monique Snoeck, Wilfried Lemahieu, and Stephan Poelmans. Existence dependency-based domain modeling for improving stateless process enactment. *2009 Congress on Services - I*, pp. 515–521, 2009.

[28] James Rumbaugh, Michael Blaha, William Lorensen, 羽生田栄一（監訳）. オブジェクト指向方法論 OMT—モデル化と設計. トッパン, 1992.

[29] Peter Coad and Edward Yourdon. *Object Oriented Analysis 2nd Edition.* Pearson Education, 1991.

[30] Peter P. Chen. The entity-relationship model–toward a unified view of date. *ACM Transactions on Database Systems*, Vol. 1, No. 1, pp. 9–36, 1976.

[31] James Rumbaugh, Michael Blaha, William Lorensen, and Frederick Eddy. *Object-Oriented Modeling and Design.* Prentice-Hall, 1990.

[32] Christopher Date, 株式会社クイーブ（訳）. データベース実践講義—エンジニアのためのリレーショナル理論. オーム社, 2006.

[33] 佐藤正美. 論理データベース論考—データ設計の方法:数学の基礎と T 字形 ER 手法. ソフトリサーチセンター, 2000.

[34] 渡辺幸三. 業務別データベース設計のためのデータモデリング入門. 日本実業出版社, 2001.

[35] Object Management Group. OMG：UML Superstructure Specification, v2.4.1. `http://www.omg.org/spec/UML/2.4.1/`, 2011. (2017 年 2 月 8 日確認).

[36] Ivar Jacobson, Patrik Jonsson, Magnus Christerson, and Gunnar Övergaard. *Object Oriented Software Engineering—A Use Case Driven Approach.* Addison-Wesley Professional, 1992.

[37] Akio Ida and Shigeo Kaneda. Applying Multidimensional Scaling for Responsibility Distribution between Objects—Linking Conceptual Activity Model and Class Diagram. *Journal of Information Processing*, Vol. 23, No. 3, pp. 318–326, 2015.

[38] 飯島正, 永田守男. オブジェクト指向ソフトウェアの構造パターンに対するデータ解析手法による分析評価の試み. 電子情報通信学会技術研究報告 知能ソフトウェア工学, 第 95(86) 巻, pp. 9–16, 1995.

[39] Peter Checkland and Jim Scholes. *Soft Systems Methodology in Action 1st Edition.* Wiley, 1990.

[40] Brian Wilson and Leslie Wilson. *Systems: Concepts, Methodologies, and Applications 2nd Edition.* John Wiley and Sons, 1990.

[41] Rebecca Wirfs-Brock and Alan McKean. *Object Design: Roles, Responsibilities, and Collaborations (Addison-Wesley Object Technologiey Series) 1st Edition.* Addison-Wesley Professional, 2002.

[42] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development 3rd Edition.* Prentice Hall, 2004.

[43] Robert C. Martin, 瀬谷啓介 (翻訳). アジャイルソフトウェア開発の奥義 第2版オブジェクト指向開発の神髄と匠の技. SB クリエイティブ, 2008.

[44] 杉本富利. 多次元尺度構成法. インフォーメーションサイエンス誌8月号, pp. 125–130, 1983.

[45] Joseph B. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrica*, Vol. 29, pp. 115–129, 1964.

[46] 中村永友, 金明哲. 多次元データ解析法 (R で学ぶデータサイエンス 2). 共立出版, 2009.

[47] 藤澤裕樹, 岡田裕, 一瀬邦継, 金田重郎. ソフトシステムズ方法論 (SSM) と概念データモデリング (CDM) を用いた業務分析手法の提案. 研究報告情報システムと社会環境 (IS), Vol. 2010-IS-111, No. 6, pp. 1–8, 2010.

[48] International Business Machines Corporation. SPSS-statistics-software. `https://www.ibm.com/jp-ja/analytics/spss-statistics-software`. (2017年2月8日確認).

[49] The R Foundation. The R Project for Statistical Computing. `https://www.r-project.org/`. (2017年2月8日確認).

[50] Frank Buschmann, Hans Rohnert, Michael Stal, Regine Meunier, and Peter Sommerlad. *Pattern-Oriented Software Architecture—A System of Patterns (Wiley Software Patterns Series) 1st Edition.* Wiley Software Patterns Series. John Wiley and Sons, 1996.

[51] Doug Rosenberg, Matt Stephens, 三河淳一 (監訳), 船木健児 [他](訳). ユースケース駆動開発実践ガイド—オブジェクト指向から Spring による実装まで. 翔泳社, 2007.

[52] 井田明男, 金田重郎, 熊谷聡志, 藤本明莉. 存在従属性に着目した論理要件ロバストなドメインモデルの作成—ドメインクラス図をユビキタス言語として用いるために. 情報処理学会論文誌, Vol. 56, No. 5, pp. 1340–1350, 2015.

[53] 金田重郎, 井田明男, 矢野寛将. 存在従属に基づくユースケースの逆生成. 電子情報通信学会技術研究報告 = IEICE technical report : 信学技報, Vol. 115, No. 54, pp. 13–18, 2015.

[54] 椿正明. 名人椿正明が教えるデータモデリングの「技」 (DB Magazine Selection). 翔泳社, 2005.

[55] 渡辺幸三. 販売管理で学ぶモデリング講座. 翔泳社, 2008.

[56] 矢野寛将, 中西啓太, 井田明男, 金田重郎. 存在従属性に基づくユースケース図の自動生成手法—公営住宅管理システムを例として. 電子情報通信学会技術研究報告 = IEICE technical report : 信学技報, Vol. 115, No. 487, pp. 25–30, 2016.

[57] 株式会社チェンジビジョン（編）. astah* professional. http://astah.change-vision.com/ja/product/astah-professional.html, 2016. (2017 年 2 月 8 日確認).

[58] Philippe Kruchten, 藤井 拓 (翻訳). ラショナル統一プロセス入門 第 3 版 (ASCII Software Engineering Series). アスキー, 2004. 藤井拓 (訳).

[59] Deeptimahanti Deva Kumar and Sanyal Ratna. Semi-automatic Generation of UML Models from Natural Language Requirements. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pp. 165–174. ACM, 2011.

[60] 中鉢欣秀, 小林孝弘, 松澤芳昭, 大岩元. シナリオの図解化によるユースケースモデリング. 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理, Vol. 88, No. 4, pp. 813–828, 2005.

[61] Mike Cohn, 安井力 (翻訳), 角谷信太郎 (翻訳). アジャイルな見積りと計画づくり—価値あるソフトウェアを育てる概念と技法. 毎日コミュニケーションズ, 2009.

[62] A.J. Albrecht and John E. Gaffney Jr. Software function, source lines of code, and development effort prediction: A software science validation. *Software Engineering, IEEE Transactions on*, Vol. SE-9, pp. 639 – 648, 12 1983.

[63] The Common Software Measurement International Consortium. Cosmic 機能規模測定法 version 3.0 手法概要編. http://www.jfpug.gr.jp/cosmic/CFFP-index.html, 2007. (2016 年 1 月 15 日確認).

[64] The Common Software Measurement International Consortium. Cosmic 機能規模測定法 version 3.0 測定マニュアル. http://www.jfpug.gr.jp/cosmic/CFFP-index.html, 2007. (2016 年 1 月 15 日確認).

[65] 情報サービス産業協会 REBOK 企画 WG(編集). 要求工学実践ガイド: REBOK シリーズ 2. 近代科学社, 2014.

[66] 藤井拓. オージス総研・オブジェクトの広場—ビジネスアプリ開発者のための機能規模測定手法 COSMIC 法入門. https://www.ogis-ri.co.jp/otc/hiroba/technical/IntroCOSMIC/IntroCOSMICPart1Jun2010.html, 2010. (2016 年 1 月 15 日確認).

[67] 駒木和彦. COSMIC 法向けの早期・概算見積もり : 不完全な要件定義への適用. プロジェクトマネジメント学会研究発表大会予稿集, Vol. 2010.Spring, pp. 188–192, 2010.

[68] 駒木和彦. CRUD 図で CFP を測定する—より簡単で確実な COSMIC 法による機能規模測定. プロジェクトマネジメント学会研究発表大会予稿集, Vol. 2011.Spring, pp. 379–384, 2011.

[69] 木村めぐみ, 藤井拓. Cosmic 概算法による機能規模測定のコスト削減の検討結果. 情報処理学会 研究報告ソフトウェア工学（SE）, Vol. 2012, No. 11, pp. 1–8, 2012.

[70] 渡辺幸三. 設計者の発言—アプリの類型における処理テーブルの形式化. http://http://watanabek.cocolog-nifty.com/blog/2015/01/post-a185.html, 2015. (2016 年 7 月 1 日確認).

[71] L. Richardson, S. Ruby, 山本陽平 (監修), 株式会社クイープ (翻訳). RESTful Web サービス. オライリー・ジャパン, 2007.

[72] 清水響子. ブロックチェーンの基本を理解してみる—知っておいて損はない気になるキーワード解説. https://it.impressbm.co.jp/articles/-/14647. (2017 年 7 月 28 日確認).

[73] 渡辺幸三. 設計者の発言—マイクロサービスで周辺アプリと連係する. http://watanabek.cocolog-nifty.com/blog/2015/12/post-7a2e.html, 2015. (2017 年 5 月 8 日確認).

[74] Martin Fowler, ウルシステムズ株式会社 (監訳), 大塚庸史 [他](訳). ドメイン特化言語 パターンで学ぶ DSL のベストプラクティス 46 項目. ピアソン桐原, 2012.

[75] Yasuo Honda. Active Record - Object-relational mapping in Rails. https://github.com/rails/rails/tree/master/activerecord. (2018 年 9 月 8 日確認).

[76] Eloquent: Relationships—The PHP Framework For Web Development. https://laravel.com/docs/5.7/eloquent-relationships. (2018 年 9 月 8 日確認).

[77] Extended Backus-Naur Form. https://en.wikipedia.org/wiki/Extended_Backus_Naur_form. (2018 年 9 月 8 日確認).

[78] Bill Karwin, 和田卓人 (監訳), 和田省二 (監訳), 児島修 (翻訳). SQL アンチパターン. オライリージャパン, 2013.

[79] MongoDB. https://www.mongodb.com/. (2018 年 9 月 8 日確認).

[80] PlantUML. http://plantuml.com/ja/. (2018 年 11 月 28 日確認).

[81] Treetop - A Parsing Expression Grammar (PEG) Parser generator DSL for Ruby. https://rubygems.org/gems/treetop/versions/1.6.8. (2018 年 9 月 8 日確認).

[82] Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. *ACM SIGPLAN Notices*, Vol. 39, pp. 111–122, 2004.

[83] OpenAPI Initiative. Swagger The Best APIs are Built with Swagger Tools. https://swagger.io/. (2018 年 9 月 8 日確認).

[84] 井田明男, 金田重郎, 森本悠介. 存在従属グラフから RESTful Web サービスの生成. 第 12 回 情報システム学会 全国大会・研究発表大会, 2016.

[85] 水野貴明. *Web API: The Good Parts*. オライリー・ジャパン, 2014.

[86] 増永良文. リレーショナルデータベース入門 [第 3 版]. サイエンス社, 2017.

[87] 児玉公信. UML モデリングの本質. 日経 BP 社, 2011.

[88] NPO 法人 IT 勉強宴会. 花束問題 V1.2. https://www.benkyoenkai.org/コンテンツ/花束問題 v1-2. (2018 年 9 月 8 日確認).

[89] 佐藤正美. データベース設計論—T 字形 ER. ソフト・リサーチ・センター, 2005.