ハッセ図としてのクラス図・ER 図について

Class Diagram and ER Diagram as Hassse Graph

金田重郎 †, 井田明男 †, 森本悠介 †
Shigeo Kanedda†, Akio Ida†, and Yusuke Morimoto†
† 同志社大学大学院・情報工学専攻
†Graduate School of Science and Engineering, Doshisha Univ.

要旨

クラス図・ER 図は概念モデリング・DB 設計等において多用されている．しかし，初学者には，どうやってクラス図・ER 図を描くのかが分かりにくい．そこで，クラス図を書きやすいものとする事を目指し,「業務が処理される順番でクラス図を作成してゆけばクラス図が完成する」方法論を追究する．一般には，関連の両端に存在するクラス間において作成順序（時間的前後関係）は含意されない．しかし，1 対多関連は例外であり，両端のクラスに時間的前後関係を含意する．そして，時間的前後関係が無い 2 つのクラス間でデータ処理が必要となり，関連が構成される必要がある場合でも，関連をクラスで表現すれば，時間的前後関係が保持される．結局，記述方法に配慮すると，クラス図は既約非巡回有向グラフ（ハッセ図）となる．ハッセ図であるので，他インスタンスの存在に無関係にインスタンスを生成できる独立型クラスをインプットとして，それを，処理時間の進行に従って，次々と処理を実行するプロセスとして，クラス図全体を記述してゆけばクラス図は完成する．具体的には，処理プロセスの時間的推移を記述する RPDD(Resource Process Dependency Diagram) を利用したガイドラインを提示する．

1. はじめに

クラス図・ER 図は，アプリケーションプログラム (AP) 開発において重要なツールである．要求分析段階の概念モデリングをクラス図で行い，それを実装を考慮して ER 図に書き直すのが一般的と思われる．ただし，どちらか一方で全てを行うことも可能である[1]．

クラス図のテキストは，海外の名著とされるものの翻訳を含めて，多数出版されている．しかし，表記法上の記述は見られるが，肝心の「どうやってクラス図を描くか?」が書かれていない．著者らは，この様な問題意識から,「クラス図は英語である」との分析結果を得て，それをクラス図教育に適用することを既に提案している [1]．しかし，このガイドラインでは，具体的なクラス図の書き方へのガイドラインとしては不十分である．

そこで，本稿では，クラス図の作り易さを担保するため,「業務の流れの順番にクラスを次々と追加すれば，クラス図が完成する」ガイドラインを目指す．初学者にとってクラス図が難しいひとつの理由は，どこから手を付けてクラス図を描けばよいのか分からない点にあると考えるからである．具体的には，関連に着目して，大木 [2] の「ライフタイム」の概念と，関数従属の考え方を導入することにより，多対多を含まないクラス図がハッセ図となることを示す．ただし，関連を引きたいが，2 つのクラスに時間的前後関係が存在しないケースについては，関連型クラス[2]で表現する必要がある．この結果，概念モデリングは，独立クラスとその独立クラスを次々と処理するプロセスの 2 つから表現できる．このモデルは，クラス図作成のためのガイドラインとして利用できる[3]．

以下，第 2 章では，ライフタイム分析と関数従属性から，クラス図の関連を分析し，クラス図がハッセ図となることを示す．第 3 章では，この理論的裏付けの元に，RPDD(Resource Process Dependency Diagram) に基づくクラス図作成法（ガイドライン）について紹介する．第 4 章はまとめである．

---

[1]本稿の議論は，クラス図でも ER 図でも適用可能なことから，以下,「クラス図」で代表させる．この場合，クラス図は，オブジェクト ID のみをプライマリーキー（PK）として持つ通常のタイプには限定されない．複数属性から成る PK を持つ，ER 図的クラス図も対象となる．

[2]本稿では，2 つのクラスに関連付けられて新に作成されたクラスのことを「関連型クラス」と呼ぶ．関連型クラスのインスタンスは，2 つの親クラスのインスタンスよりも後に生成される．関連型クラスから見た親クラスの多重度は必然的に 1 である．

[3]本稿は，情報処理学会・SES2018 において発表した著者らの論文（査読付きシンポジウム論文 [3]）の論点を見直したものである．ただし，提案している RPDD(Resource Process Denpendency Diagram) は同じである．

表 1: 多重度の扱い一覧

No	関連の多重度	説明	本稿での扱い
1	多対多	一方の端が「0..*」あるいは「1..*」．同様に，他方の端も，「0..*」あるいは「1..*」．	多対多関連は，関連型クラスを用いて 1 対多で表現できる．RDB にマッチしないこともあり，本稿のモデリングから除外する．尚，これにより，多対多の除外がモデルの表現能力を限定することはない．
2	1 対多	1 側は「1」，他端は「0..*」	クラス間の時間的前後関係を含意
3	1 対 1	1 側は「1」，他端は「0..1」	
4	1 対 1	1 側は「1」，他端も「1」	同時にインスタンスを生成すべきだが，現実には実装上，どちらかが先．
5	1 対多	1 側は「1」，他端は「1..*」	本来は順序関係はないが，現実には，1 側が「1」多側が「0..*」の 1 対多と同様に実装
6	1 対多	1 側は「0..1」，他端は「0..*」	
7	1 対 1	1 側は「0..1」，他端も「0..1」	関連型クラスを導入して，前後関係に変換すれば，クラス間の前後関係は成立
8	1 対多	1 側は「0..1」，他端は「1..*」	

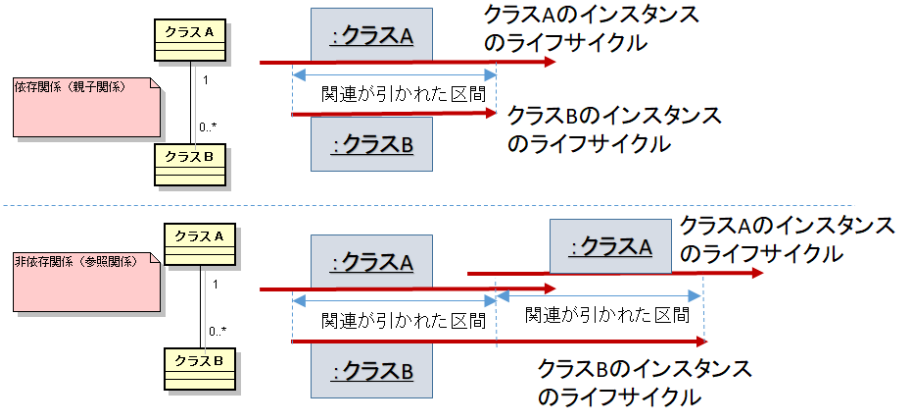

図 1: インスタンスのライフタイムと 1 対多関連

2. ハッセ図としてのクラス図

2.1. 多重度と時間的制約関係

本稿では，関連の多重度と，クラス間の時間的前後関係に着目する．分析結果を表 1 に示す．尚，本稿のモデリングでは，多対多関連は利用しない．自明の方法（関連型クラス）を用いて，1 対多に変換できるため，モデリングには用いない．これによる，モデリング能力の低下はない．

次に「1 対多」を考える.「1」側には 2 通りある.「1」と「0..1」である．まず，最初に，1 側が「1」のケースを考える．図 1 は,「1 対多」で多側が,「0..*」の場合を示している．上段は依存関係，下段は非依存関係である．非依存関係の場合には，クラス A 側に常時インスタンスを必要とするが，リンクを張り替えて，クラス A の異なるインスタンスに変更できる．

図 1 に示すように，事前に「1」側インスタンスが出来ていないと，リンクは張れない．リンクは,「多」側インスタンスが生成されると同時に張られる．逆に,「1」側インスタンスが生成されたからと言って,「多」側インスタンスが存在している必要はない．以上から,「1」側インスタンスの生成は,「多」側インスタンスの生成に先行するか同時であると理解する．

この関係を，一種の「原料—加工結果」「原因—結果」をセマンティクスとして理解する．例えば，従業員クラスと課クラスとの関係では，課側の多重度は「1」であり，従業員側の多重度は「0..*」となる．この場合，配属先である課の名称を準備してから，従業員を割り付けることを意味する．

上記の関係は,「1」側が非依存関係でも変わらない．非依存関係でも，最初のリンク生成の際には,「1」側インスタンスが先か同時に存在している必要がある．このため「1」側が先行する．また，1 対 1 で，片方が「1」，もう一方が「0..1」でも同様であり,「0..1」側のクラスは後から生成できる．時間関係とし

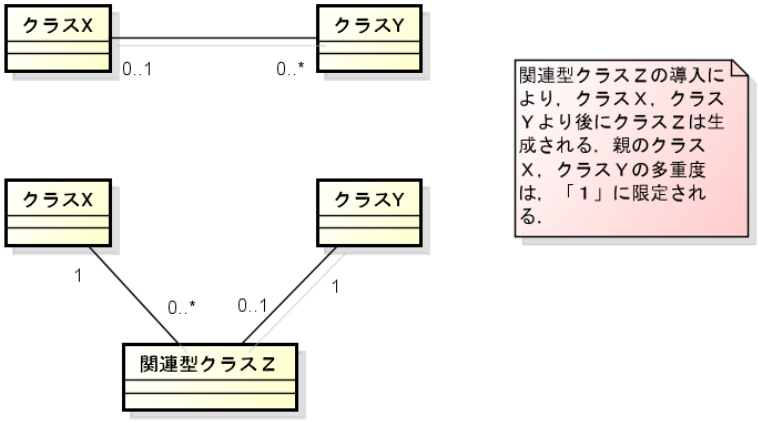

図 2: 関連型クラス導入による前後関係の保持

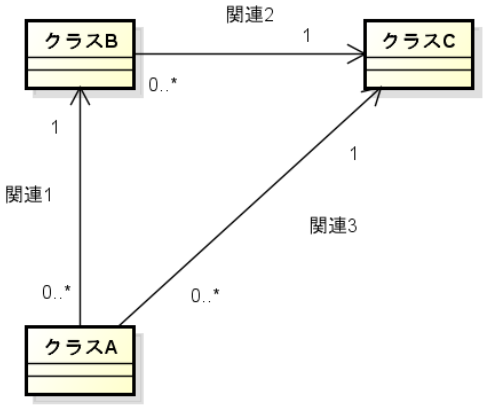

図 3: 関数従属としての 1 対多関連

て,「1」側が早く,「0..1」側は遅いことを含意する．

次に，1 側が「1」で，多側が「1..*」の場合を考える．この場合には,「1」側クラスのインスタンスの生成と,「多」側クラスのインスタンスの生成は，同時である必要がある．しかし，これも,「多」側が「0..*」である場合と同様に，現実には「1」側が先行すると見なせる．例えば，受注クラスと受注明細クラスを考える．この場合，受注明細側の多重度は「1..*」であり，後から追記できるが，最初の受注クラスは，最初に作ってあるだけである．この例でも,「1..*」側のクラスは後から生成される．両端が「1」である場合には，実装上は，どちらかのインスタンスを先に作り，あとから残りを作成する．

以上，見てくると，以下の性質 1 が成立する．

【性質 1】1 対多（1 対 1 を含めて）において，1 側の多重度が「0..1」ではなく「1」ならば，関連リンクの貼られた 2 つのクラスの処理順序は,「1」側が先である．

更に，1 側が「0..1」のケースを考える．表 1 の項番 6,7,8 である．この場合，関連の両端のクラス間での時間的前後関係を保証できない．しかし，図 2 に示すように，クラス間の関連を，関連型クラスで表現することにより，親クラスとの間に前後関係が生じる．以下の性質がある．

【性質 2】1 対多（1 対 1 を含む）の 1 側の多重度が「0..1」の場合には，2 つのクラス間に時間的前後関係は含意できない．しかし，関連型クラスを導入することで，関連型クラスの元となる 2 つのクラスとを前とし，関連型クラスを後とする，時間的前後関係が保持される．

時間的前後関係はノードであるクラス間の半順序と見なせる．1 側が「1」である 1 対多関連以外では，関連型クラスを導入する様に，クラス図作成をガイドできれば，本稿で扱うクラス図では，すべてのクラス間関連に，時間的前後関係（半順序）が存在する．以下の性質 3 が成立する．

【性質 3】1 対多関連（1 対 1 を含む）以外の関連を関連型クラスで表現したクラス図は，有向非巡回グラフ（DAG:Directed Acyclic Graph) である

クラス図は DAG であるので，必ず，1) 他インスタンスの状態に無関係にインスタンスを追加できる「独立型クラス」及び，2) 他のクラス（複数の時もある）の情報を加工して生成された「従属型クラス」に分かれる．また，独立型クラスから，時間的前後関係のリンクを辿って行けば，最後は，そのクラスの後には何もない終端のクラスにたどり着く．

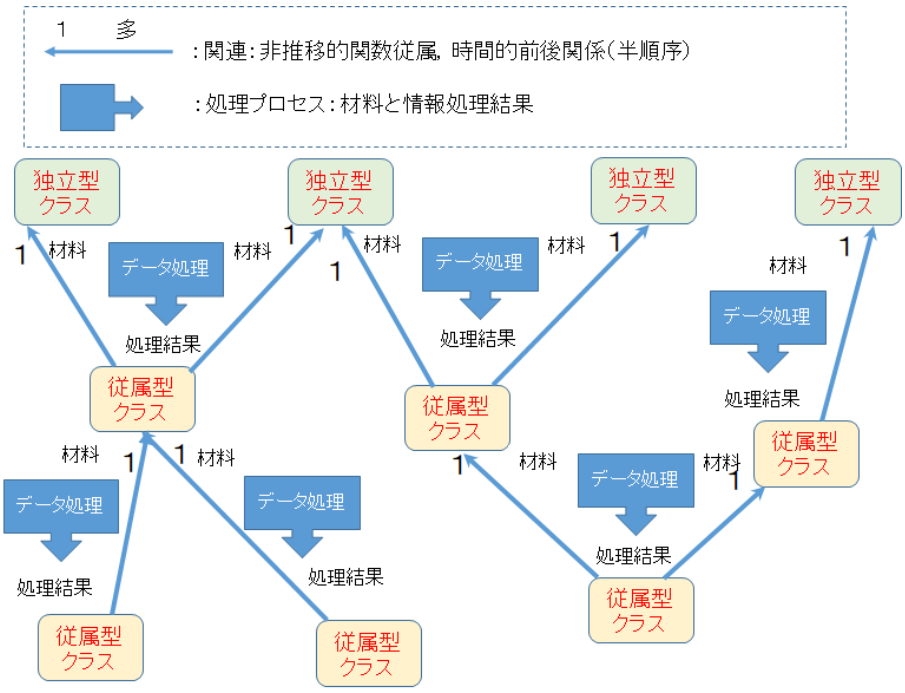

図 4: ハッセ図としてのクラス図

2.2. 関数従属性と 1 対多関連

表 1 に示した様に，本稿で議論しているクラス図では，関連は有向であり，時間的に先行したクラスへ多重度「1」で向かっている．つまり,「多」側のインスタンスを決めると，ひとつ，かならず,「1」側のインスタンスが決まる．本稿では，これを関係 DB モデルの関数従属性と同じものと理解する．そう理解すると，図 3 のクラス A からクラス C への直接の関連は，推移的な関数従属であると見なせる．従って，クラス図作成では，クラス A からクラス C の様な関連は張るべきではない．つまり，クラス図における関連とは，直接的因果関係で，それ以上分解できない基本単位（半順序）を制約関係として表記すべきである．これは，クラス図作成のガイドラインの一つである．

前節では，1 対多関連は半順序関係であることを論じた．そうなると，図 3 においてクラス A からクラス C への直接的な関連を取り除いて，非推移的な関数従属のみから DAG を構成することになる．これは，DAG の既約化である．明らかに以下の性質が成立する．

【性質 4】1 対多関連を関数従属性と理解すると，非推移的関数従属のみから構成されるクラス図は，既約な DAG，つまり，ハッセ図である．

以上から，クラス図は，図 4 の様な構造を持っている．独立クラスのインスタンスを材料として，データ処理が行われ,「多」側クラスのインスタンスが生成される．つまり，データ処理ステップが関連である．処理結果として，従属型クラスが設置される．材料となる側が，多重度「1」なのは暗示的である．材料だから，処理の前に必要となる原料の個数が変わったりしないのである．

加えて，図 4 を見ると，1) 単一のクラスから単一のクラスに処理を行うパスと，2) 2 個のクラスから 1 つのクラスを生成するパスがある．前者では，時間的に前となっているクラス側の多重度は「1」でなければならない．例えば,「0..1」となればどこかに誤りがある．一方，多重度が「0..1」となる可能性があるクラスについては，関連づけられるもう一つのクラスと時間的前後関係は規定されないので，既に，別のクラスとして準備されているはずである．その場合，図 4 に示すように，関連型クラスとなるので，自動的に，親となる 2 つのクラスに向いた多重度は「1」である．結局，図 4 のハッセ図の親側の多重度は全て「1」となり，親側が先で，子側が後となることと整合する．

そうであるなら，クラス図を構築するには，独立型クラスをインプットとして準備し，それを材料と

する単位となる処理（=動作動詞）を時間的な発生順序に従って，並べて置き，あとは，このハッセ図のノードの部分に，適切なクラスを追記すれば，クラス図は完成することになる．これに基づいて，クラス図のガイドラインを提案する．

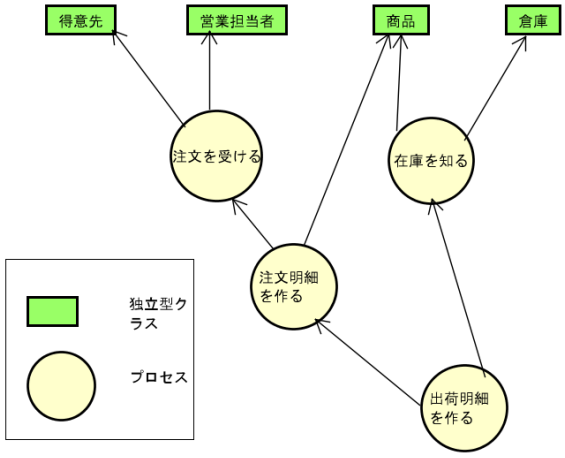

図 5: RPDD の例

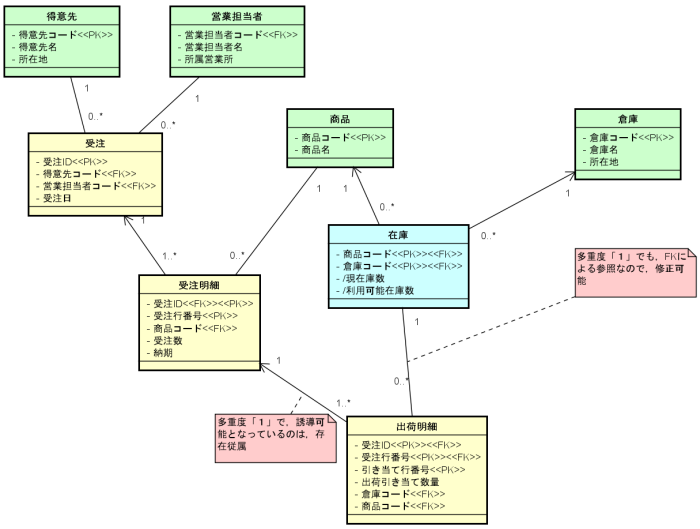

図 6: 渡辺による設計例

3. クラス図作成のガイドライン

以上の分析から，本稿では，クラス図作成のガイドラインとして，RPDD（Resource Process Dependency Diagram) を用いる．図 5 は簡単な RPDD の例である．RPDD では，最上位に，他インスタンスとは独立してインスタンスを追加できる独立型クラスをドメインから抽出して並べる．それを最大値をもつノードとして，後は，業務処理の時間的前後関係に着目しながら，この「材料」をつぎつぎと加工するプロセスを並べてゆく．プロセスの結果や記録が，次のノードに位置するクラスである．この出力された加工結果であるクラスは，次の段階の材料となる．

RPDD の構造の妥当性は，以下の理由から説明される．

【RPDD の妥当性】RPDD 作成時において，あるクラス X の後に，プロセス P が処理され，結果としてクラス Y が生成されるとする．多重度が 1 対多（表 1 の項番 2,3,4,5）の場合である．この場合，クラス X が存在しても，プロセス P が実行されないと，クラス Y は存在しない（まだ生成されていない）．そこで，クラス Y を RPDD に追記して，クラス X からクラス Y にアークを張り，そこに，プロセス P を割り付けなければならない．この場合，RPDD に追記されるのは，一本の流れである．関連は，結果的に，クラス間に張られる．クラス X 側の多重度「1」は「プロセスの材料」を暗示する．

これに対して，関連リンクが結ぶべき 2 つのクラス X,Y の間に，時間的前後関係がない場合を考える (表 1 の項番 6,7,8 である).「1」側の多重度が「0..1」であるケースである．この場合，そのタイミング以前の RPDD 作成が正しければ，あるプロセス P を追記するべきタイミングでは，すでに，関係した 2 つのクラスは RPDD 上に存在している．そうなると，RPDD の構造上，材料となる 2 つのクラス X と Y の間に張られる関連ではなく，関連を表現する関連型クラス Z が生成され，X 側,Y 側の多重度は 1 となる．出来上がったクラス図では，2 つのクラスから 1 つのクラスへ向かうアークである．この場合も，材料側の多重度は,「1」である．ただし，本稿では，2 つのクラスから関連型クラスを作成することは考える．3 クラス以上の関連については，2 クラスに対する関連型クラスを用いて，分解するものとする．

以上から，RPDD では，正しくタイミングを順を追って RPDD を作り上げてさえすれば，1 対多関連と，それ以外の関連に相当するデータ処理が RPDD 上に配置され，ハッセ図としてのクラス図の構築がガイドされる．

以下，ガイドラインに基づくクラス図作成のステップを示す．

Step-1 独立クラスをドメインから抽出して，RPDD(Resource Process Diagram) の最上部に置く．この図は，DFD に似ているが，事務処理の表面上のフローではなく，あくまでも概念モデリングレベルの要の「もの・こと」を分析対象とする．

Step-2 以下，上にあるクラス図を材料として，業務処理を行うプロセスを書いてゆく．上部のクラス複数個を材料として使う場合には，複数のクラスから，出力側のクラスへプロセスを 1 個設置する．

Step-3 上記を時間的順序に従って繰り返し，それ以上，業務の連鎖が下に伸びなくなるまで，プロセスの前後関係を守りつつ，追加してゆく．

Step-4 プロセスの部分を 1 対多（1 対 1 を含む）関連として，ネットワークの位置に，従属型クラスを設ける．

Step-5 クラス間の関連が，1 対 1 となった場合には，両端に存在する 2 つのクラスの統合を検討しておく．

図 6 は，実際に上記を実行した際に，得られるモデルの候補である．渡辺幸三の文献を参考にしている [4]．図 5 の RPDD からは，自然にこの様なクラス図を作成できると考える．

4. 終わりに

本稿では，クラス図における「関連」の両端に存在するクラスの相互には，時間的前後関係（原因・結果関係，材料・製品関係等）を含意することを問題提起した．その結果，クラス図は，有向非巡回グラフ（DAG) となる．加えて，1 対多関連を一種の関数従属性とみなせば，一つ一つの関連は，非推移的な関数従属性を満たす必要がある．結果的に，クラス図はハッセ図となる．

そうであるなら，クラス図は，最上部に他のインスタンスの有無に無関係にインスタンスを追加できる独立型クラスを配置し，それを原料の様にして，つぎつぎと時間的順序に従って，プロセスが起動される様子を描いたモデル RPDD(Resource Process Denpendency Diagram) を描き，そのノードをクラスとして，クラス図を作成すればよい．

本稿の提案手法では，1 対多（多は「0..*」) となる関連が，両端のクラスの時間的前後関係を含意している性質を利用している．一方で，その他の「0..1」等の多重度を持つ関連では，クラスの時間的前後関係が含意されないので，関連型クラスで表現している．結果的に，本ガイドラインに沿って，クラス図を作成する場合，クラス数が増加するものと思われる．しかし，一方で，関連ではなく，クラス（インスタンス）として，プロセスの処理結果をインスタンスとして生成するので（当該インスタンスが無効となった後でも），容易にログとして残すことができる．従って，関連型クラスを増やしてでクラス図を記述することには，大きな問題点はなく，むしろ，要のもの・ことを DFD の様に，時間的推移にそって想起するだけで，クラス図を構成できるので，初学者にも，扱い易いガイドラインと考える．逆に言えば，業務知識が無い限り，クラス図は描けないことを意味している．

参考文献

[1] 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 D，Vol.J98-D，No.7，pp.1068-1082，2015 年 7 月

[2] 大木幹雄,「ライフタイム分析に基づくクラス構造抽出の定式化と構造に関するデザインパターンの抽出実験」，情報処理学会論文誌 Vol.45, No.6, pp.1554-1568, 2004 年 6 月

[3] 金田重郎，井田明男，森本悠介,「正規化クラス図（ER 図）作成のガイドライン －要のもの・こと間のハッセ図としてのクラス図－」，ソフトウェアエンジニアリングシンポジウム 2018, シンポジウム論文，2018 年 9 月

[4] 渡辺幸三,「データモデリング入門」，日本実業出版社，2001 年 7 月，P.150，図 2

[5] 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 年 11 月