存在従属クラス図に基づく正規化オブジェクト指向設計手法

Normalized Object-oriented Design based on Existing Dependency Class Diagram

金田重郎 †，劉湘涛 †，森本悠介 †，井田明男 †
Shigeo Kaneda†, Xiangtao Liu†, Yusuke Morimoto†, and Akio Ida†
† 同志社大学大学院　理工学研究科
†Graduate School of Science and Engineering, Doshisha Univ.

要旨

関係モデルではテーブルの正規性を重要視する．一方，クラス図では正規性は重要視されない．アプリケーション（以下「AP」と記す）の実装時には，クラスを非正規化し，他インスタンスの属性値を，当該インスタンスの属性値としてコピーする事が行われている（コピーして持ってきた属性値を以下「View 的属性値」と呼ぶ）．この非正規性は，AP の内部構造・処理を複雑にして，工数を増大させる．この問題を解決するため，本稿では，プライマリーキーに非推移的関数従属する属性のみをクラスの属性とするオブジェクト指向分析・設計手法を提案する．View 的属性値は，多重度 1 の関連を辿り，他インスタンスから取得する．但し，View 的属性値をアクセスするのは，インスタンスの状態が時間的に変化した後である．そこで，インスタンスの履歴はすべて記録する．あるインスタンスの View 的属性値が必要な場合は，タイミングを考慮して，当該インスタンスから多重度 1 の関連を辿って属性値を取り出せば良い．

1. はじめに

RDB（関係データベース）の世界では，関数従属性に基づくテーブルの正規化は重要概念であり，これにより，更新時異常，削除時異常等を回避して，処理を簡明化する．本来，オブジェクト指向分析におけるクラスは，RDB のテーブルに対応する．クラスは，プライマリーキー（例えば，オブジェクト ID) に対して，少なくとも第 3 正規形でなければならない．しかし,「クラスは第 3 正規形であるべきである」との視点は，オブジェクト指向分析のテキストでは見かけない．更に，インスタンスのライフタイムと，インスタンス上の属性値のライフタイムは一致するべき[1]との大木幹雄の主張 [1] も，オブジェクト指向のテキストでは見かけない．

上記の「クラス図は第 3 正規形」とテキストに書かれていない大きな理由のひとつは,「現実の AP では，第 3 正規形を崩した View 的な属性を追記せざるを得ない」為であろう．例えば，固定資産税の納税通知クラス（図 1）について考えると，税額や課税日のみではなく，課税客体（たとえば，固定資産税金の原因となる不動産の所在地），課税主体（自治体名），納税義務者氏名，納税義務者住所等も記載する必要がある．

しかし，納税義務者の氏名や住所は，人クラスの属性であり，固定資産税クラスの属性としては不都合である．この様な属性を，本稿では「View 的属性」と呼ぶ．View 的属性の値は，もともと，元のインスタンスから一意に特定できる，他インスタンスの属性値の筈である．この様な View 的属性値を，インスタンス生成時に，他インスタンスからコピーして持ってくると，時間的推移によってコピー元の値が変化（例えば，改氏名，改住所）した場合，メンテナンスの問題を生じる．その結果，AP は複雑化し，開発・保守の工数は増加する．

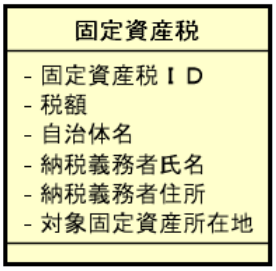

図 1: クラス図の例（固定資産税）

[1]このライフタイムの制約は，非推移的関数従属と表裏一体である．

この問題を回避するため，本稿では，AP を構成するクラスは少なくとも第 3 正規形であることを前提とする．更新時異常・削除時異常等の問題は起こらずに，AP 制御構成は簡明化する．しかし，View 的属性の値は，AP が必要とするタイミングで，インスタンス間の「関連」を辿り取得する必要がある．具体的には，井田明男の存在従属クラス図 [2] における関連の多重度が 1 の性質を用いて追跡する．但し，関連を辿ろうとするタイミングによっては，元のインスタンス間の関係は変化している．このため，本提案手法では，インスタンスの状態変化を全て記録し，追跡が要求されたタイミングでの属性値を問い合わせ元のインスタンスに返すメカニズムを必要とする [3].

以下，第 2 章では，本提案手法の原理を述べる．第 3 章では，固定資産税のクラス図を例として，実現性を検証する．第 4 章はまとめである．

2. 提案手法

2.1. 構成原理

本稿では，クラス図における「多重度＝ 1」は，意味的に，関数従属性に等しいと考える.「片方のインスタンスが決まれば，他方のインスタンスが決まる」からである．もともと，クラス図の関連は，ポインタの様なものと捉えている．多重＝ 1 なら，相手のインスタンスが一意に決まるから「辿る」事が可能となる．本提案手法の基本原理は，以下の原理 1 である．

【原理 1】インスタンスから関連リンクで一意に辿れるインスタンスの属性値は，View 的属性値でありアクセス可能とする．具体的には「多重度＝ 1」「多重度＝ 0..1」の場合は，リンクがあれば辿り得る．

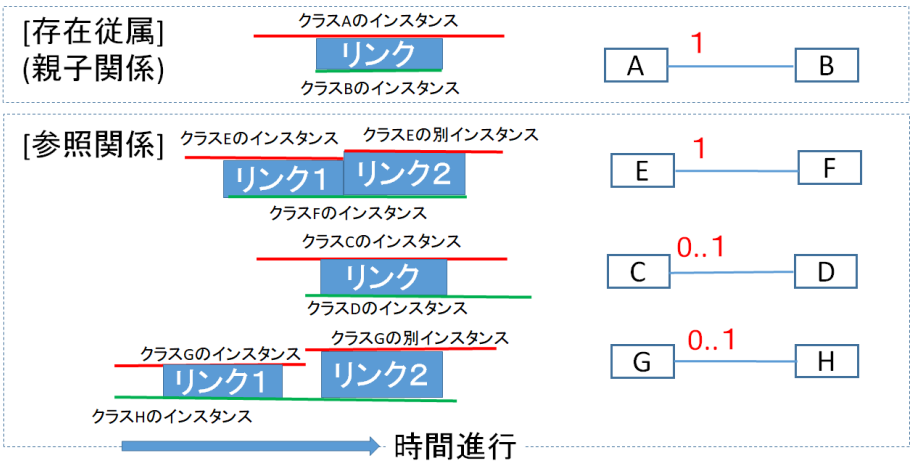

図 2: 多重度 1 の種別

図 2 を用いて，多重度を詳細に見てゆく.「多重度＝ 1」には 2 種類ある．第一は，図 2 の最上部に示した，クラス A,B のケースであり，クラス B のインスタンスが決まれば，クラス A のインスタンスが決まるとの意味で，関数従属性を満たす．クラス B のインスタンスは，クラス A の特定のインスタンスが無いと存在し得ない．井田明男の「存在従属関連 [2]」に該当する．渡辺幸三が「親子関係」と呼ぶものにも近い [4]．この「多重度＝ 1」は，作られても，インスタンス間の関係は一通りしかないことに注意されたい．

一方，図 2 の下 3 つのケースは，それぞれ，クラス図の右側のインスタンスが決まったからと言って，唯一の左側クラスのインスタンスは決まらない．渡辺幸三の「参照関係 [4]」に近い概念なので，参照関係と呼ぶ．参照関係での「多重度＝ 1」は，その上の「親子関係」と一緒のように見えるが，一意に決まるクラス E は，次々と隙間なく切り替えられる可能性がある．更に,「参照関係」の残り 2 つのケースは，多重度が「0..1」であるのでクラス D のインスタンスやクラス H のインスタンスが決まったからと言って，相手のクラス C,G は一意ではなく，切り替わる．

以上の分析から，提案手法では，以下の原理 2 に従う．

【原理 2】提案手法では，存在従属クラス図を用いて，存在従属関連が成立するクラスのみから，クラス図を作成する[2]．これにより，オブジェクト間の関連が固定されるので，何時でも，インスタンス間の関係を辿ることができる．

但し，AP のすべての関連を存在従属関連で記述できる保証はない．そこで，次の原理 3 を用いる．

【原理 3】図 2 の参照関係を存在従属関連で記述するためには，関連を関連クラスとして表現する[3]．但し，すべての関連を関連クラス化するのではなく，存在従属関連で，極力記述し，どうしても扱えないものを関連クラスとする．関連クラスは存在従属クラスである．

上記の原理 3 から，どの様なクラス図でも，存在従属関連で表現できることが担保される．但し，関連クラスがやたら増えそうな印象を受ける．この点については，今後の実証が必要であるが，最初から存在従属関連でクラス図を描く場合，関連クラスにしなければならないクラスは多くなく，意味的にも無理は感じない[4]．

2.2. 関数従属関連の追跡

本提案手法では，次の原理 4 を原則とする．

【原理 4】本提案手法では，View 的属性値は,「多重度＝ 1 」の関連を辿って値を得る」[5]．

存在従属の設計思想では，インスタンス間のリンクの構造は固定的である．図 2 の最上部に示した様に，相手にしているインスタンスは 1 個である為である．AP の処理進捗とともに，インスタンスのデータ状態は変化するが，リンク構造は変化しない．但し,「View 的属性値」を取得する際には，すでに，消えてしまったインスタンスの状態を利用する必要が生じる．図 1 の例で言えば，納税義務者の氏名，住所は最新のものでなければならない．しかし，課税主体である土地・建物の所有関係は，過去のある時点で，その人が所有していたにすぎず，現在は，他人との間の所有関係となっている恐れがある．このため，最新のオブジェクト間の関連を用いると，誤った人に納税通知を送りかねない．逆に古いタイミングでは所有者（人）は住民だったが，その後転出している可能性もある．この問題を回避するため，本提案の手法では，以下の原理 5 を採用する．

【原理 5】クラスには，ある期間内が有効となる「期間型クラス」と，ある特定のタイミングの状態を記録する「イベント型クラス」を設ける[6]．本稿では，期間型には有効期間開始タイミングと有効期間終了タイミングを表す属性を設ける[7]．イベント型では，生成時点を示すタイミングを 1 個持たせれば十分である．これにより，AP 上で消去されたインスタンス，変更前のインスタンス状態は，すべて記録される．期間型では，属性値が変更されることがあるが，本稿では，変更時点でそれまでのオブジェクト

[2]実際にドメインが要求するすべてのクラスを存在従属クラス図で表現できるか否かの検証は，今後の課題である．しかし，世の中に独立して存在するオブジェクトは，独立クラスとして表現でき，その独立クラスに従属したイベントである従属クラスで，ほとんどオブジェクトはカバーできると考えたい．尚，独立クラス，従属クラスは，存在従属の用語であるが，椿正明のリソースクラス，イベントクラス [6] に近い概念である．

[3]佐藤正美のアプローチ [5] に近い．

[4]例えば，図 4 で，固定資産の所有関係は，固定資産と人との間の「関連」である．固定資産の所有者は切れ目なく変わるので多重度=1 となる．しかし，税金賦課の原因は「所有している事実」であって，関連クラスとした方が自然である．

[5]本稿では，is-a 関連と has-a 関連（集約とコンポジション）については触れていない．但し，has-a 関連は，状態動詞に対応する典型的な関連の一つであり，本稿と同様の議論が成立する．is-a 関連については，スーパークラスの関連をサブクラスに読み替える必要はあるが，基本的な多重度の関係は，スーパークラスとサブクラスで変化しないと考える．

[6]イベント型，期間型の区別は本質的なものではない．期間型のみでモデル化してもよい．しかし，モデリングする際に，2 種類を意識して使い分けることにより，モデリングが支援される．イベント型は，過去のあるタイミングでのデータ状態であり，後から属性値を修正することは許されない．

[7]有効性開始タイミングは，当該インスタンスのプライマリーキー属性（集合）に含めることも考えられるが，そのような実装設計については，本稿では議論を省略する．

の有効期間終了タイミングを記入して，論理的には AP から消去する．そして，変更後のデータ状態は，新しい期間型オブジェクトを生成して記入する

以上の構成原理から，以下のことが分かる．

- 存在従属クラス図では，関連は存在従属関連しか存在しない．この場合，オブジェクト間の存在従属関連は，固定であり変化しない．したがって，タイミングに無関係に，リンクは辿り得る．
- 存在従属関連を辿る場合，それが実際にリンクしているか否かは，関連が生きている必要があり，タイミングに依存する．よって，関連を辿る場合には，辿るタイミングを指定して処理する必要がある．

存在従属クラス図において，図 2 の存在従属（親子関係）では,「多重度＝ 1 」であり，設計時点でリンク先は確定している．時間経過とともに従属先のインスタンスが切り替わることもない．このため，存在従属クラス図から，インスタンスの存在従属上で上流側にある（推移的に関数従属性を持つ）各インスタンスを特定できる．結果として，各インスタンスの属性にアクセスして値を取得するメソッドを（インスタンス生成時に）自動生成できる．このメソッドは，各インスタンスに設定でき，これにより，View 的属性の属性値の取得を保証する．

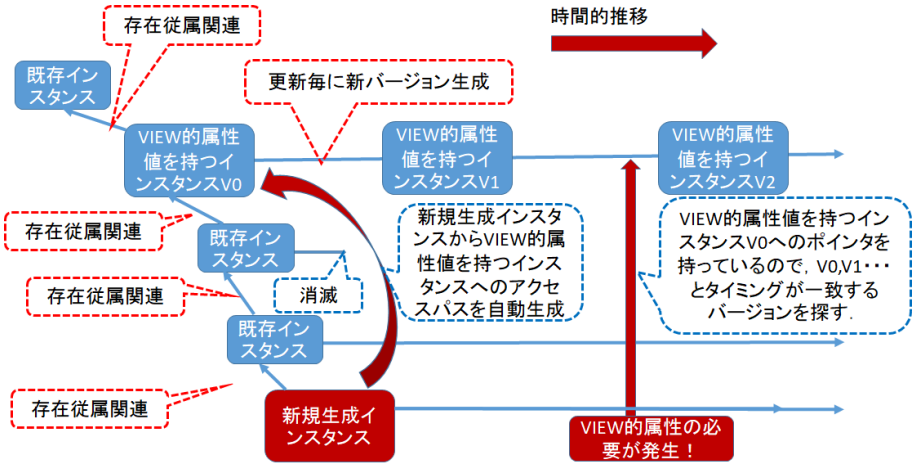

図 3: 実際に View 的属性値を得るメカニズムのイメージ

但し，View 的属性の属性値の取得関数は，かならず，アクセスするタイミングを指定する必要がある．インスタンスの AP 上での論理的な存在/不存在を含めて，当該インスタンスの生成以後，消滅までのデータ状態はすべてログとして残す（図 3 参照）．このため，メソッドに引数として渡されたタイミングと，ログに残された有効期間開始タイミング，有効期間終了タイミングを比較して，タイミングが合致したログから属性値を取得する必要がある．生成されていなかったり，すでに AP 上は消去されている場合には，エラーを返す．但し，この関連を辿る処理は，実際に関連を次々と辿る必要はない．

辿る元となる下流側インスタンスが生成されたときには，すでに，存在従属の上流側のインスタンスはすべて生成された後であり，上流側インスタンスへのポインタは分かっている．このポインタで，ターゲットインスタンスの最も古いものが得られる（万が一，既に，上流側は消去されていた[8]としても，実際にはログは残してある）ので，タイミングを確認しながら，同一インスタンスのログを調べて，属性値を得ればよい．

尚，上流側インスタンスから，多重度 1 でリンクされているインスタンスが存在する場合も，辿ることができる．図 2 の参照関係 3 種類に相当する．但し,「多重度=0..1」では，リンクが常に存在する保証はない．また，参照関係の「多重度＝ 1 」では，実際に辿るタイミングが来るまではどのインスタンスをポインティングできるかは確定しない．

[8]無論，実際に消去するのではなく，有効期間終了タイミングの属性に値を入れ，AP の論理上は消去する事になる．

いずれのケースでも，提案手法ではすべての参照関係は関連クラスを用いた存在従属関連に置き換えられる．このため，インスタンス間に参照のためのリンクが構築または変更されるタイミングで，関連クラスのインスタンスが生成される．そのインスタンスの識別子は参照元のインスタンス ID ＋参照先のインスタンス ID ＋リンク開始時点であり，属性としてリンク終了時点を持つ．そのため，任意の時点の参照元と参照先のインスタンスの属性値からなる View を取得できる。なお，存在従属関連であれば参照先のインスタンスが参照元のライフサイクルを通じて固定であるため，このような時点情報は捨象可能である。従って，関連クラスを用いて，存在従属化することが望ましい．

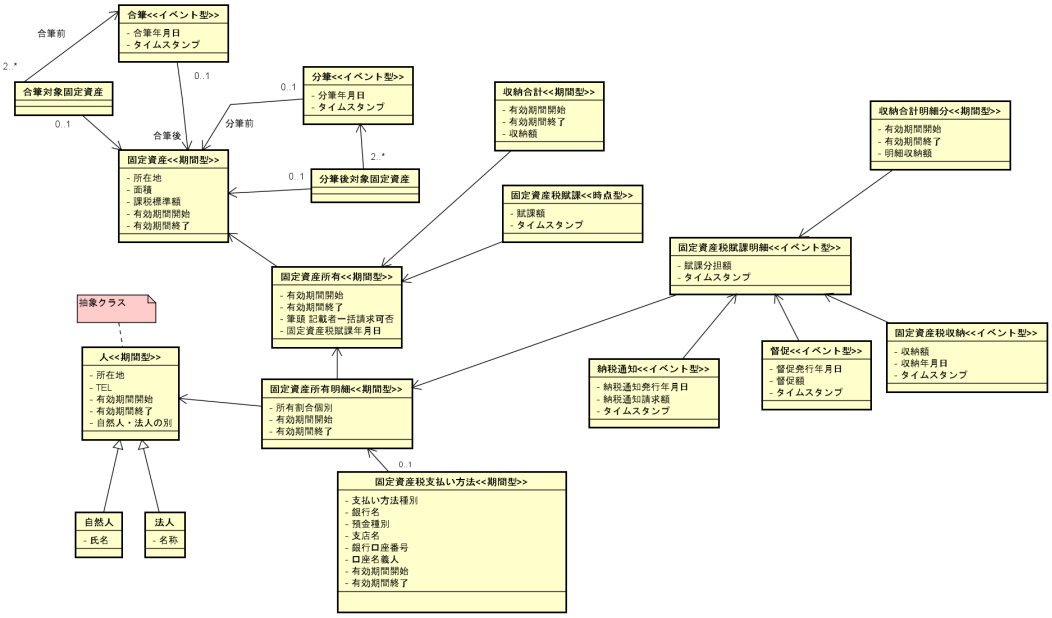

図 4: 固定資産税の存在従属クラス図の例

3. 存在従属クラス図による確認

上記提案の有効性については，今後の評価を待つ必要がある．ここで，一例として，図 4 には，固定資産税に関する存在従属クラス図を示す．例えば，納税通知を出そうとした際には,「納税通知」インスタンスから存在従属の矢印型の関連でポイントできるあらゆるインスタンスの属性がアクセス可能である．具体的には，以下に示す項目がアクセスできる（クラス名と属性名で示す）．おそらくは，納税通知を出すには十分であろう．

- 固定資産税賦課明細（賦課分担額）
- 固定資産所有明細（所有割合個別）
- 人（法人・自然人の別，所在地，氏名または名称，TEL）
- 固定資産所有（所有会館開始タイミング，所有期間終了タイミング，筆頭記載者一括請求可否，固定資産賦課年月日）
- 固定資産（所在地，面積，課税標準,）

更に，多重度＝ 0..1 の場合も追跡できる．図 3 の例では，以下の項目である．

- 固定資産税支払方法（支払方法種別，銀行名，預金種別，銀行口座番号，口座名義人）
- 合筆（合筆年月日）
- 分筆（分筆年月日）

次に，固定資産税の賦課後に，所有者が変わった場合を考える．この場合，それまでの「固定資産所有」インスタンスは無効化され（データは残る），新たな「固定資産所有」インスタンスが，異なる「人」との間に結ばれる．しかし,「固定資産税賦課明細」等の課税データは，影響を受けない．

一般に，存在従属クラス図に基づく設計では，下流側にインスタンスが存在する場合，上流側のインスタンスは消去できない．本提案手法でも，クラス図の段階では，そのクラス間の制約は保持されるべきである．しかし，図 3 からもわかる様に，本提案手法では，下流側にインスタンスが存在しても，上流側インスタンスが現実世界の状態に合致しなくなれば消すことができる．

尚,「固定資産所有」インスタンスを無効化したところで，下流側の「固定資産税賦課明細」からの当該インスタンスへのポインタは消えていない．一度作ったインスタンスは消さないのが，提案手法の大原則であるため，固定的な存在従属関連を辿って，リンクはいつでも辿れる．但し，有効期間への注意は必要であり，リンクを辿るタイミングがこの期間内になければ，インスタンスのデータを取り出すことはゆるされない．

4. 今後の課題とまとめ

本稿では，関係モデルと同様にして，オブジェクトの正規性（少なくとも第 3 正規形）を前提とする AP 構築法を提案した．着目しているインスタンスと関係した「View 的属性値」は多重度＝ 1 あるいは 0..1 のリンクをたどって取得する．但し，多重度＝ 1 で追跡することは，一般のクラス図でモデル構築しても可能である．しかし，提案手法では，あえて存在従属クラス図を採用した．その理由は以下の 3 項目である．

1. 存在従属クラス図であれば，多重度＝ 1 しかないので追跡できる範囲が広い．
2. 存在従属関連は，上流側を原因として，下流側が結果となっている．保証されているわけではないが，下流側インスタンスの存在原因を上流側に持っていると考えられるので，View 的属性を辿るのに適している可能性がある．
3. 下流側インスタンスを生成した際に，ポインタアドレスとして追跡可能である．上流側インスタンスを指定可能となるため，View 的属性を取得するメソッドの高速性が期待できる．

逆に，目的とする View 的属性が多重度＝ 1 で追跡できないことは，推移的に関数従属な値を，モデル上では取得できないことになり，モデルの正当性が疑われる．現実の AP では，渡辺幸三の「参照関係」の様に，存在従属関連とは見なせない関連もある．しかし，この問題は，関連クラスを持ち込めば解決する．存在従属クラス図を前提とすると，全関連を関連クラスにする必要は無く，モデルの可読性が悪くなるとも思われない．但し，提案手法には未検証の問題が残されている．1) 実際にリンクを辿るだけで十分な種別の属性値が得られるのか，2) 十分な処理速度でリンクを辿れるのか，3) クラス図の設計上，制約にならないか等である．今後の課題としたい．

参考文献

[1] 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, 2004, pp.1554-1568
[2] 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, 2015, pp.1340-1350
[3] 金田重郎, 井田明男, 森本悠介,「オブジェクトの正規性を重視して存在従属関連を用いたドメインモデリング手法」, 電子情報通信学会，知能ソフトウェア研究会，研究報告，2017 年 7 月
[4] 渡辺幸三,「業務別データベース設計のためのデータデリング入門」, 日本実業出版社，2001 年
[5] 佐藤正美,「T 字形 ER データベース設計技法」ソフト・リサーチ・センター，　1998 年
[6] 椿正明,「名人椿正明が教えるデータモデリングの技」, 翔泳社, 2005 年
[7] 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.7，2015, pp.1068-1082