［論文］

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

Application Design Methodology with Normalized Classes by using Existence Dependency and Version Management

金田 重郎†, 井田 明男†, 森本 悠介†, 劉 湘涛†, 上野 洸史‡
Shigeo KANEDA, Akio IDA, Yusuke MORIMOTO, Xiangtao LIU, , Kouji UENO,

†同志社大学大学院理工学研究科
‡同志社大学理工学部
†Graduate School of Science and Engineering, Doshisha University
‡Faculty of Science and Engineering, Doshisha University

要旨

オブジェクト指向を採用したアプリケーション設計では, 構築・保守容易化の観点から, クラスの正規性が重要である. クラス正規化のためには, (1)インスタンスの存在期間が, インスタンス中の属性値の存在期間と一致し, (2)インスタンスのプライマリーキー（PK)に, インスタンスの全属性値が非推移的関数従属する, 必要がある. しかし, 現実には, あるクラスのインスタンスに関係した一連の処理のため, 他クラスのインスタンスの属性値を, 当該インスタンスに属性追加してコピーすることがある. この正規形から外れた属性を, 本論文では「View 的属性」と呼ぶ. View 的属性の追加は, クラスの正規性を破壊し, アプリケーションの保守性を低下させる. この問題を解決するため, 本論文では, (1)正規性を満たす存在従属クラスを前提として, (2)View 的属性は「多重度= 1」によりリンクを追跡して得る事とし, (3)任意の時点での属性値を得るため, インスタンスをバージョン管理する, アプリケーション構成法を提案する. 本手法によれば, インスタンス生成時に, 辿り先のインスタンスへのリンクを一意に決定でき, 結果として, インスタンス生成後の任意時点における View 的属性値を, 少ない処理ステップ数で取得できる. 提案手法の実現性を確認するため, 公営住宅管理システムの仕様書を参考に, クラス数 77 の存在従属クラス図を描き, 46 種の帳票に出現する View 的属性が, 存在従属関連のリンクを辿って取得できる事を確認した.

Abstract

The normality of classes is important for improving the capabilities of designing and maintaining application programs (APs). Two key requirements must be met to normalize classes: (1) the lifetime of each attribute value of an instance should be equal to that of the instance itself, and (2) each attribute value of an instance should have non-transitive functional dependency on the primary key (PK) of the instance. However, in the actual design of APs, the attribute value of an instance is often copied to another instance by adding a new attribute that is transitively functionally dependent on the PK. We refer to this kind of normality-destroying attribute as a “View Attribute.” Since the View Attribute destroys the normality of the class, it degrades the AP’s designability and maintainability. As a solution to this problem, this paper proposes a novel application architecture: (1) all classes of the AP’s entity-layer objects are normalized and designed by existence dependency classes, (2) the AP acquires View Attribute values by tracking the link with “multiplicity = 1,” and (3) a version management mechanism is implemented to manage the instances so that the attribute value can be obtained at an arbitrary timing. Using the proposed architecture, the AP can set the pointer to the destination instances as the source instance is created. As the result, the AP can easily and swiftly acquire the View Attribute values at any given timing. The experimental evaluation shows that an existence dependency class diagram can be made for an actual specification document of a public housing management system and that the View Attribute values appearing in the system’s approximately 40 application forms can be acquired by the proposed method.

［論文］2018 年 2 月 27 日受付, 2018 年 9 月 30 日改訂, 2018 年 11 月 29 日受理

1. はじめに

アプリケーションプログラム（以下「AP」と略記する）の開発・保守を容易化するためには，AP を構成するクラス[1]は，関係データベース理論の意味で正規形であるべきである．加えて，属性値の存在期間は，属性値を保持しているインスタンスの存在期間と一致するべきである[1]．しかし，現実の実装設計では，あるインスタンスに関する一連の処理の中で，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しがちである．第 3 正規形から外れた，この種のコピー属性を，本論文では「View 的属性」と呼ぶ．View 的属性は，クラスの正規性を破壊し，アプリケーションの保守性を低下させる．

上記の問題を解決するため，本論文では，存在従属クラス図[2]を用いた[2]，新しい AP 構成法を提案する．具体的には，第 1 に，クラス設計には，正規性を満たす存在従属クラスを用いる．第 2 に，View 的属性は，「多重度=1」のリンクを追跡して得る．更に，第 3 として，任意の時点での属性値を得るため，インスタンスをバージョン管理システムにより管理する．これらにより，「多重度=1」によりユニークに辿り得るインスタンスについて，任意の時点における属性値を，少ない処理量で取得・利用できる．

以下，第 2 章では提案手法の準備として若干の用語説明を行う．第 3 章では，提案手法を述べる．第 4 章では，提案手法の有効性を検証する．第 5 章では，View 的属性値を取得するメソッドを自動生成できるプロトタイプシステムについて述べる．第 6 章は，まとめである．

2. クラスの正規性と View 的属性

2.1. クラスの正規性

関係データベース（RDB）の世界では，関数従属性に基づくテーブル正規化は重要であり，第 3 正規形にとどまらず，第 4・第 5 正規形が要求されることも多い．これにより，更新異常，削除異常などの矛盾点を回避して，処理が簡明化する．オブジェクト指向設計におけるエンティティクラスは，RDB におけるテーブルに対応する．結果的に，ひとつのクラスは，オブジェクト ID[3]をキー属性として，少なくとも第 3 正規形でなければならない．更に，大木[1]はインスタンスの存在期間と，その属性値の存在期間が一致するべきとしている．

以上から，本論文では，正規化されたクラスとは，(1)オブジェクト ID 等の主キー（Primary Key，以下「PK」と略称する）に対して，すべての属性値が非推移的に関数従属し，(2)属性値の存在期間は，その属性を持つインスタンスの存在期間と一致する事をいう．例を示す．図 1 のクラスでは，属性「父親犬の名前」と「母親犬の名前」が正規形から外れている，この 2 つの属性の存在期間は，（子どもである）「犬」インスタンスの存在期間とは異なる．父親犬も，母親犬も，子どもが生まれる前から生存している．父親犬の名前は，父親犬のインスタンスの属性値であるべきであり，図 1 には，その本来のインスタンスからコピーして持って来ている事になる．その意味で，「父親犬の名前」「母親犬の名前」は，異なるインスタンスの値を反映した View 的属性である．

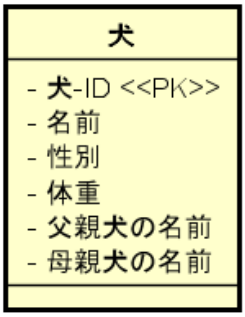


図 1「犬」クラスの例

[1]本論文で扱うクラスとは，バウンダリーレイヤー，コントロールレイヤー，エンティティレイヤーで分けると，エンティティレイヤーのクラスである．通常，データベースで永続化される．

[2]本論文における，「存在従属クラス図」とは，存在従属クラスのみから構成される，エンティティレイヤーの概念データモデルのことを言う．

[3]本論文では，「オブジェクトには 1 個だけオブジェクト ID がある」とするオブジェクト指向の考え方は取らない．一般的には，クラスは，複数の属性からなるプライマリーキーを有しているが，ここでは，理解を容易にするため，オブジェクト ID と呼ぶ．

2.2. 多重度=1 による View 的属性値の取得

上記の様な，正規形を破壊するコピー属性をクラスが持つことは，AP の開発・保守の観点から望ましくない．そこで，本論文の提案手法では，AP 設計において，View 的属性をクラスには作らず，完全な正規形でクラス設計を行う．ただし，この場合，AP 処理上必要なデータである，現在着目しているインスタンスに関連した View 的属性値をどうやって取り出すかが課題となる．

この課題を実現するため，本論文では，クラス図における「多重度=1」を辿って，関連した View 的属性を取得する[4]．例を示す．図 2 には，自動車税のドメイン知識を UML クラス図でモデリングした例である[5]．クラスは正規化されている．「自動車税納税通知」クラスに関する処理では，納税義務者の氏名や住所が必要である．これら View 的属性値は，「自動車税納税通知」インスタンスから「多重度=1」でリンクを辿って（「人」インスタンスの属性である）氏名や住所からコピーして取得できる．この例から見てもわかる様に，「多重度=1」による View 的属性の取得は，（本論文で採用する存在従属クラスで無くても）既存の一般的な UML クラス図でも実現可能である．

しかし，一般の UML クラス図を用いて辿る場合には，一つの問題点がある．図 2 の例で言えば，自動税納税通知から自動税賦課を辿るのは，自動車税納税通知インスタンスが生成された日付（例えば，4 月 10 日）以降のデータ状態で辿る必要がある．一方，自動車税の賦課は 4 月 1 日に行われたとする．その場合，自動税賦課から自動車を逆に辿る処理は，4 月 1 日現在のデータ状態で行う必要がある．4 月 2 日以降において，自動車は譲渡されているかも知れないからである．「多重度=1」で辿る場合には，リンクを辿るごとに AP のドメイン知識に基づいて，適切な時点を考慮しなければならない．結果的に，AP の負担が大きい．

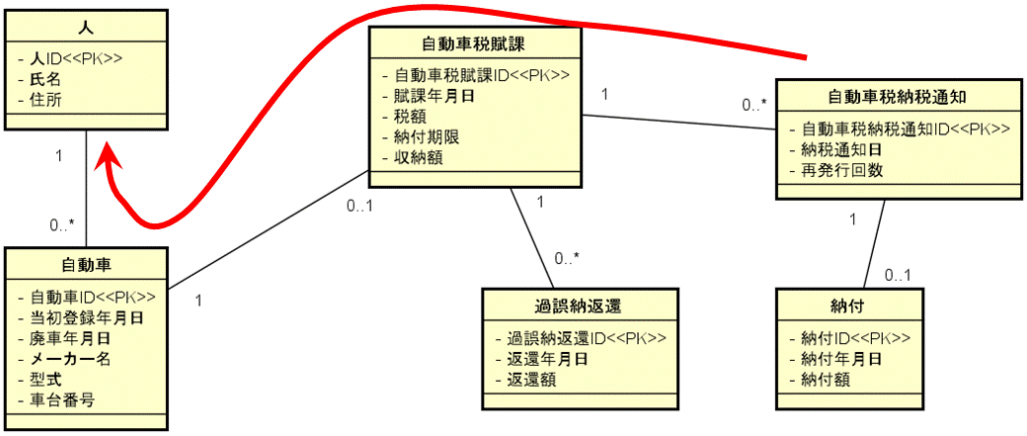


図 2「多重度=1」による追跡可能性

2.3. 2 種類の「多重度=1」

前節で論じた「辿ることの難しさ」は，UML クラス図の「多重度=1」の持つ意味が 2 通りあることに起因する．図 3 に，「多重度=1」の 2 つのケースを示す．ただし，図 3 において，上側のクラスが，「多重度=1」側のクラスである．

第一の意味は，「多重度=1」で接続されるインスタンスが唯一存在する場合である（図 3 左）．この場合，下側のインスタンスのライフタイムは，必ず，上側インスタンスのライフタイムに包含される．この場合の関連は，井田らによる存在従属に等しく[2]，渡辺[3]が「親子関係」と呼ぶケースに該当する．

これに対して，図 3 右では，「多重度=1」であるが，対象となる上側インスタンスは，次々と切り替わる．確かに，ある時点で見れば，上側のインスタンスは 1 個であるが，時間が経過すると対象が切り替わる．渡辺が「参照関係」と呼ぶケースである．図 2 の例で言えば，自動車の所有者は常に一人だけであるが，時間の経過により，所有者が変わる．この様な「参照元インスタンスの切り替わり」があるため，アプリケーションドメインの業務知識を使って，「いつの時点で辿れば良いのか」を AP が意識する必要がある．

この様に，UML クラス図を用いて設計すると，「いつの時点で辿るのか？」を逐一，対象クラスのビジネス上の業務規則から判断し，場合によっては，必要な時点にデータ状態をバックトラックする必要

[4] AP が処理上で必要となる関連した属性値を，「多重度＝1」追跡法で漏れなく取得できるか否かは別途確認が必要となる．この点は，後述の評価の部分で論じる．

[5] ただし，業務知識としては簡単化している．

が生じる．結果的に処理が複雑化する．

一方において，上記を嫌って，従来の様に，注目しているインスタンスに，正規形を壊す View 的属性を追加してしまうと，設計・保守が複雑化する．上記のいずれのアプローチを用いても，AP の設計・保守は複雑化する．

上記の 2 通りの解釈は，「多重度=0..1」についても成立する[6]．この場合では，多重度=0..1 が付与されている側とは反対側のインスタンスが生成されていても，必ずしもリンクが張られているとは限らない．それに対して，関連リンクが張られている場合には，ひとつの時点ではリンクは 1 本に限定される．しかし，図 3 と同様に，相手のインスタンスが 1 個に限定される場合と，相手が切り替わり，複数個存在するケースがある．

但し，図 3 の左側のケースと右側のケースの差異は，もともと，ER 図では意識して区別されて来た．渡辺は図 3 の左側を「親子関係」，右側を「参照関係」と呼ぶことは既に述べた [3]．井田らの存在従属クラス図[2]におけるリンクは，親子関係に相当する．存在従属クラスでは，リンク先のインスタンスは唯一（図 3 左）に限られる．

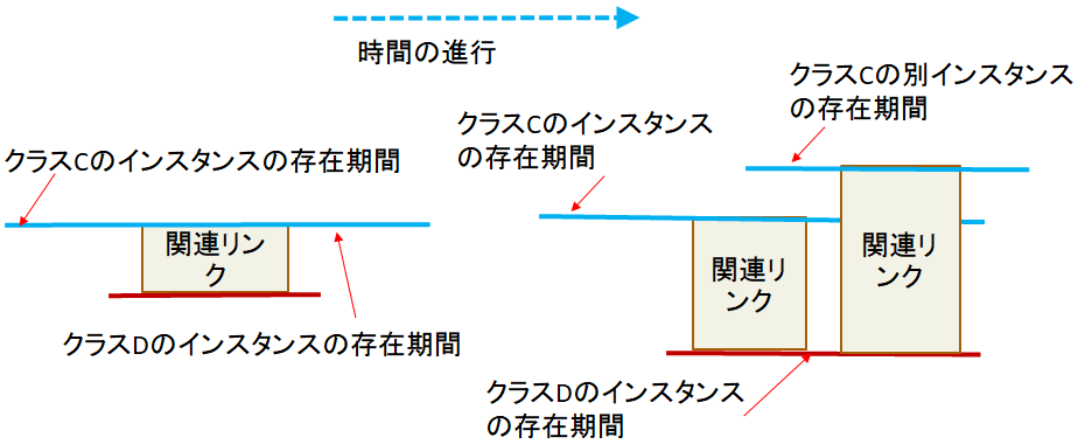


図 3「多重度=1」の 2 通りの意味

以上の分析から，以下のことが分かる．

(1) 存在従属クラス図のみでクラス設計が可能なのであれば，「多重度=1」で辿るべきインスタンスは，図 3 の下側のインスタンスが生成された際に一意に決まる．一度追跡できれば，直接に辿り先インスタンスをアクセス可能なリンク情報を取得できる．これにより，以後，何度でも，容易・高速に，辿り先となるインスタンスに到達できる．

(2) 辿り先となるインスタンスの何時の時点でのデータが必要かは AP 仕様に依存する．従って，各インスタンスはバージョン管理システムで管理し，任意の時点の属性値を取り出せる様にすべきである．住民の住所でいえば，固定資産の賦課時点である 1 月 1 日の住所なのか，AP 処理時点の住所なのかと言った問題である．

一方で，課題も残る．存在従属クラス図は，リンクを存在従属関係に制限している．結果的に，存在従属クラス図を書くことは容易なのか？あるいは，クラス数が増加しないのか？と言った課題が存在する．従来のクラス図は，存在従属関係（親子関係）のみで記述されているわけではない．例えば，渡辺のテキストにおける ER 図作成例[3]では，個数的に，親子関係はリンク全体の 3 分の 1 程度に過ぎない．残りの大半が単なる参照関係であるが，存在従属クラス図では，関連クラスで表現されることになる．結果的に，存在従属クラス図では，クラス数が一般的なクラス図に比して多少増加する可能性はある．しかし，単なる参照関係を用いた場合と比べて，クラス図としての表現能力が異なるわけではない．

以上から，本提案手法では，is-a 階層を除いて，全関連が存在従属関連となる．「多重度=1」での追跡だけで，本来 AP が必要とする属性値を網羅的に取得できるのかと言う課題も生じる．これらの点は，後ほど，評価の章で明らかにする．

---

[6] 「多重度=0..1」を敢えて論じているのは，多重度=0..1 でも，リンクが存在すれば追跡ができるからである．その具体的なケースについては，後述する．

3. 提案手法

3.1 存在従属クラス図

提案手法の第 1 の特徴は，存在従属クラス[3]の全面採用である．これにより，「多重度=1」の追跡を簡明化する．存在従属クラスでは，図 3 左と同様に，「インスタンスが生成された時点において，多重度=１による辿り先インスタンスを一意にポインティングできる．リンクが多段であっても，この性質は保証される．

図 4 は存在従属クラス図の簡単な例である．クラス C のインスタンスは，クラス A のインスタンスとクラス B のインスタンスを特定して初めて生成できる．また，クラス A，B の様に，他インスタンスの状況に無関係に，当該クラスのインスタンスを生成できるクラスを「独立クラス」と呼ぶ．一方，クラス C は「従属クラス」であり，そのインスタンスは，(1 個以上の)他クラスのインスタンスの存在を前提として生成される．リンクの矢印はそれを示している．リンクの矢印が向う側を「上流側」，反対側を「下流側」と呼ぶ．存在従属クラス図では，上流側の多重度は表記しないが，図 4 では，リンクを辿れる事を示すために，リンクの上流側の「多重度=1」を表示している．

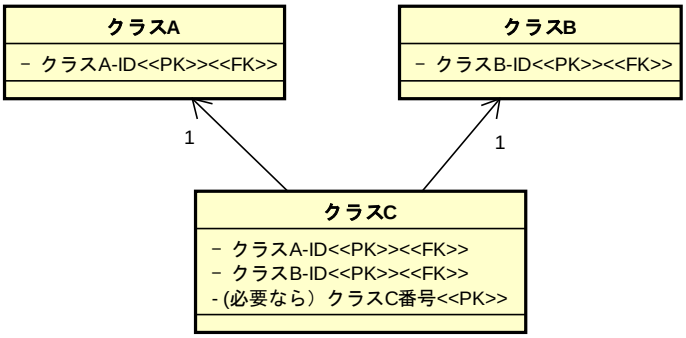


図 4 存在従属クラス図の簡単な例

存在従属クラス図では，下流側のクラスのPKは上流側のクラスのPKを全て受け継いだものとなる．図4の例では，クラスAのPKをクラスA-ID，クラスBのPKをクラスB-IDとするとき，クラスCのPKはクラスA-IDとクラスB-IDを併せたものとなる．ここで，A-ID,B-IDは，外部キー（Foreign Key，以下「FK」と略称する）となる．この場合には，クラスAの特定のインスタンスと，クラスBの特定のインスタンスの配下には，唯一のクラスCのインスタンスしか生成できない．複数個生成する必要がある場合には，クラスCのPKとして，クラスA-ID，クラスB-ID，に加えて，インスタンス生成時のタイムスタンプをPKに追加すれば良い（シーケンス番号等でもかまわない）．結果的に従属クラスの主キーは必ず複合主キーとなる．

3.2 DAG(サイクルのない方向性グラフ)

存在従属クラス図における存在従属リンクは，一度，2 つのインスタンスに対して張られると，下流側のインスタンスが消えるまでは，上流側インスタンスを消去できない．逆に言えば，インスタンスを下流側で作成した時点で，上流側へのリンクはすべて存在している．その結果，存在従属クラス図は，クラス図全体が DAG(サイクルを持たない方向性グラフ)を形成する．DAG 構造[7]のイメージを図 5 に示した．このため，下流側のインスタンスから，辿り先となるインスタンスを追跡でき，最後は，最上流，すなわち，独立クラスのインスタンスに到達する．「多重度=1」で辿り得るクラスを，容易に読み取ることができる．

ここで，存在従属クラス図における，多重度=0..1 での追跡可能性を確認しておく．図 5 では，クラス D からクラス F への逆方向の辿りのケースがあり得る．存在従属クラス図では，すべての関連が，上流側に「多重度=1」を持っている．このため，「多重度=0..1」が存在し得るのは，図 5 の様に，存在従属関連の下流側へ向いた時のみである．この場合，存在従属関連リンクが存在する間だけ，下流側クラスのインスタンスが存在する．上流側への存在従属リンクが発生した段階で，下流側インスタンスは一意に決まる．

言い換えると，たとえば，図 5 で，クラス E のインスタンスを作成した時に，当該インスタンスから

[7]再帰的関連であっても，インスタンスは異なるためインスタンスレベルでは非循環リンクとなる．

辿り得る辿り先インスタンスとして，ABCD は認知できるが，クラス F のインスタンスが当該インスタンスから辿り得ることが分かるのは，クラス F のインスタンスの生成時点となる．

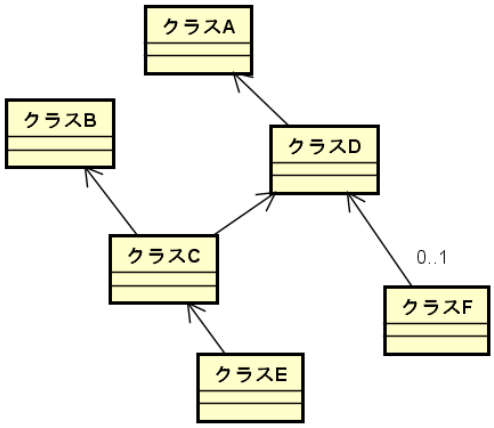


図 5　存在従属クラス図における追跡

3.3　提案手法

以上の分析から，以下の AP 構成法を提案する[4][5]．図 6 に提案手法の概要イメージを示す．本提案手法では，以下の 3 つの要素機能を結合させる．

（1）　クラスを正規化した，存在従属クラスでクラス設計
（2）　「多重度＝1」あるいは「多重度＝0..1」による View 的属性の追跡
（3）　インスタンスをバージョン管理して，任意の時点での属性値を取得可能

以下，これら要素の連携について説明する．

提案手法では，エンティティレイヤーのクラスは，存在従属関連（親子関係）のみで記述する[8]．存在従属クラス図では，辿り先のインスタンスが（着目しているクラスのインスタンスの存在期間中に）削除されることはない．また，上流側インスタンスも**他のインスタンスに挿げ替えられる**こともない．このため，図 6 に示す様に，着目しているインスタンス生成時に，時点情報なしに，辿り先インスタンスを特定できる．

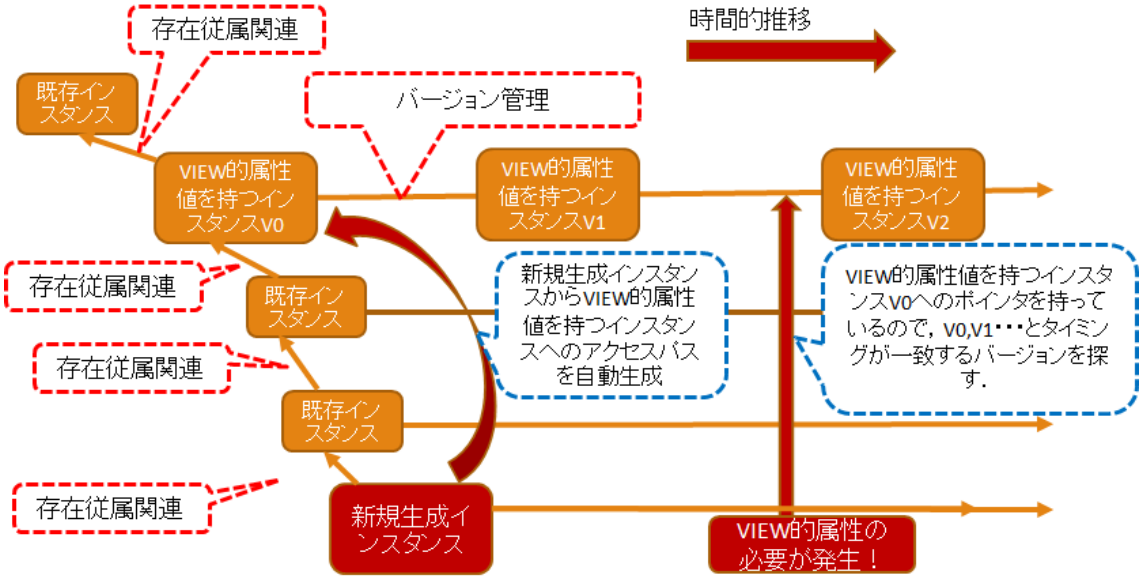


図 6　提案手法の処理方式

[8] is-a 階層も許される．is-a 階層が存在しても「多重度=1」の追跡は可能である．本論文では is-a 階層に関する議論は割愛する．has-a 階層は，通常の関連リンクと変わるものではない．

本論文で提案するのは，処理方式であり，実装には関知していない．図 6 を具体的にどう実装するかは今後の課題である．ただし，例えば，インスタンスがすべてインコア（主記憶上に展開されている）と仮定すれば，各インスタンスは仮想空間上のアドレスを持つ．従って，「新規生成インスタンス」が生成された時点で，存在従属リンクで辿り得る全インスタンスのアドレスが確定し，以後，この値は変化しない．即ち，一旦，ポインタアドレスが確定すれば，辿り先のインスタンスを極めて高速にアクセスできる．その点が本提案手法の特長である．それに対して，図 2 の様な従来型クラス図で辿ろうとすると，処理は複雑化しており，処理の高速化は難しいと考えられる．

上記の様に，提案手法で辿り得るのは，「あるインスタンスから見て，存在従属の上流側にある，他のインスタンスまでのルート」である．辿り先のインスタンスが決まれば，その属性値は容易に読み出せることとなり，View 的属性値を取得できる．ただし，この定義では，例えば「申請人の家族一覧」のような，複数のインスタンスから構成される辿り先は特定できない．しかしながら，例えば，「申請人」インスタンスを辿り先として一意に決定できれば，その後は，AP の内部処理として，「申請人」との関係で，家族一覧を出力することは容易と思われる．

一方，各インスタンスは，図 6 に示すように，何らかの方法で，バージョン管理され，任意の時点の値を取り出せるようにするべきである．これによって，例えば，納税義務者の住所について，AP 処理時点の住所が知りたいのか，過去のある時点（例えば，4 月 1 日）の住所が知りたいのかに応じて，バージョン管理システムに渡す時点を制御すれば良い．しかも，辿る際には，着目している下流側インスタンスから，辿り先のインスタンスへ直接ジャンプできる．これに対して，図 2 の様な既存のクラス図では，ある辿り先が必要となる毎に，毎回，中間に位置するインスタンスを辿る必要があることになる．

更に，提案手法では，図 5 に示した様に，辿れるクラスがどれかを一目で判断できる．単なるビジュアル的な側面であるが，図 2 に比べると，利用しやすいクラス図と考える．

4. 提案手法の検証

以上の説明から分かる様に，図 6 の提案手法では，（1）制約されたリンクしか持たない存在従属クラス図で，実用規模のアプリケーションのクラス図が描けるのか，（2）「多重度=1」による追跡で，AP 処理が必要とする View 的属性を網羅的に取得できるのか，という問題がある．本章では，この点を評価した結果を示す．

4.1 「多重度=1」による追跡

図 7 は，提案手法の原理を説明するために作成した，簡単化された自動車税のクラス図である．この中で，「自動車所有」クラスは，関連クラスに類似したクラスであり，所有関係をクラス化している．存在従属クラス図でモデリングする場合，この種のクラスを用いることにより，存在従属関連のみで，モデリングできる．

図 7 において，「自動車税納税通知」インスタンスから「多重度=1」で辿れる属性は，以下の通りである．

所有者の氏名，住所
自動車の初期登録年月日，廃車年月日，メーカー名，型式，台車番号
自動車所有の開始年月日，終了年月日
自動車税賦課の賦課年月日，税額，納付期限，収納額

納税通知書を送るために必要な項目は網羅されている．逆に言えば，納税通知書の発行業務に関係した処理を行うために必要な情報が得られない (そのインスタンスから一意にポイントできない) 場合には，モデリング自体に不備があると考えられる．

ここで，注意を必要とするのは，「多重度=1」による追跡は，「見つかるものが，追跡ルート毎に 1 個だけ」との制約を意味することである．たとえば，家族情報（一般に，答えが複数ある）の様な複数個の情報を，この方法だけで見つけることはできない．図 8 は，後述する，ある自治体の住宅管理システムの調達仕様書から作成した存在従属クラス図の一部である．この場合，「入居申し込み」インスタンスからは，申し込み名義人の氏名，氏名ふりがな等を追跡できる．しかし，家族一覧は得られない．家族一覧を作るには，AP は名義人の「人 ID」を外部キーとして，入居人クラスから検索する必要がある．ただし，「入居申し込み」インスタンスから，少なくとも，名義人は特定できており，辿り先となるインスタンスの直前までは，追跡できている．その意味では，辿ることはできている．

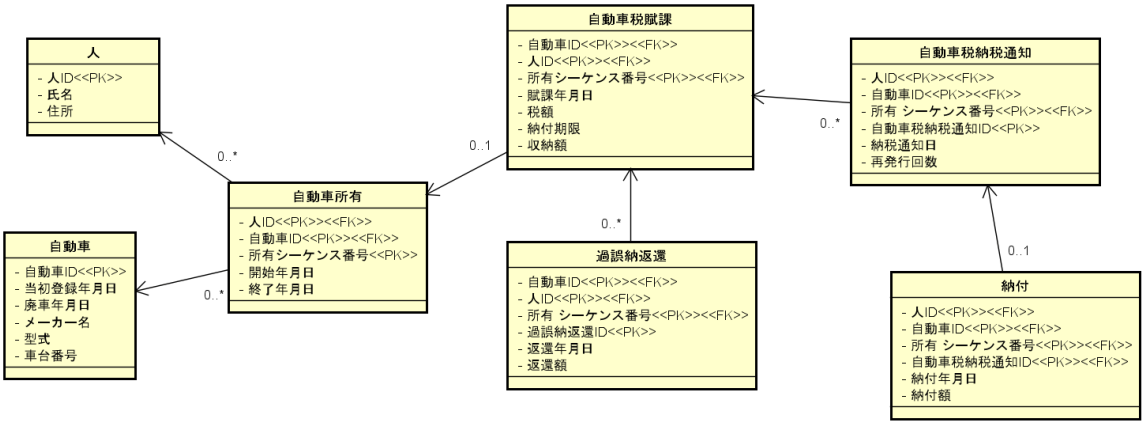


図 7　存在従属クラス図を用いた View 的属性値の追跡

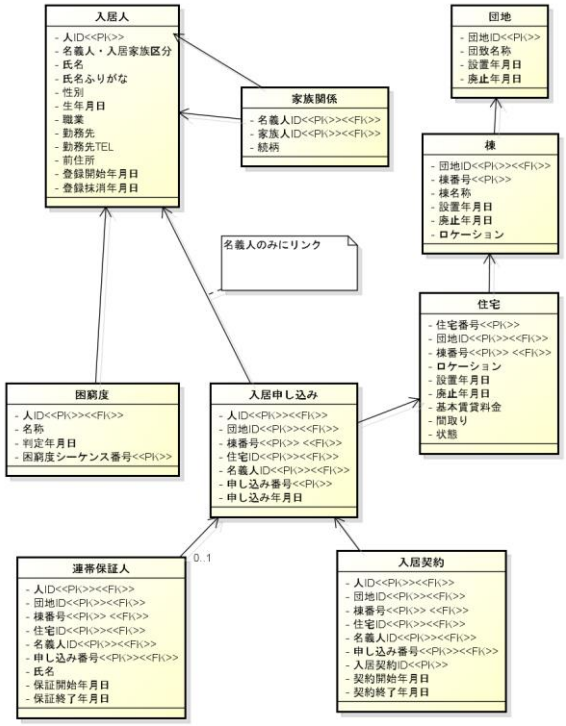


図 8　住宅管理システムの存在従属クラス図（部分）

4.2 存在従属クラスの記述性

次に問題になるのは，実用規模 AP においても，存在従属クラス図がコンパクトにモデルを記述し得るかという課題である．また，実際の業務処理で必要となる View 的属性が「多重度=1」で網羅的にアクセス可能なのかも確認を要する．

上記の問題は，実用規模のシステム設計で評価しないと意味がない．そこで，ある自治体（以下，A 自治体と呼ぶ）の住宅管理システムの調達仕様書と帳票類に基づいて，存在従属クラス図によるエンティティレイヤーのモデリングを行った．具体的には，以下の 2 つの評価を実施した．

STEP1: A 自治体の公営住宅管理システムの調達仕様書(概要機能仕様書)に従って，エンティティレイヤーの存在従属クラス図を作成.

STEP2: A 自治体から入手した当該システム全帳票 43 種類に対して，帳票上に表示されるべきデータ項目を「多重度=1」で追跡可能か否かを存在従属クラス図上で追跡・確認.

STEP3: 追跡できない場合には，クラス追加・属性追加.

ただし，大阪府の Web サイトで公開された類似業務のための申請帳票[6]には，A 自治体の帳票に含まれないものが 3 種類（その内のひとつを図 10 に例示）存在したため，3 帳票を追加した．実験的なクラス図作成結果は以下の通りである．

・調達仕様書から作成したモデル
41 クラス，162 属性（プライマリーキーを構成するものを除外）
・帳票 46 種類に基づく追加後のモデル
36 クラス追加，149 属性追加（既存クラスへ追記 44 属性，新規追加クラスの属性 105）

図が細かいので，イメージのみとなるが，図 9 には作成された存在従属クラス図の全体イメージを示す．図 9 の作成時には，存在従属クラス図の作成が難しいとの印象は無かった．また，「関連をクラス化しているために冗長である」との印象もなかった．むしろ，「所有関係」の様な，通常ならば関連でも表現できそうなリンクであっても，クラス化した方が分かり易く自然であった．

存在従属クラス図は，上流側クラスからスタートして，つぎつぎと，従属するクラスが追記される形となっている．このため，一連の業務処理が，時間経過とともに，段階を経てゆくような性質の応用に向いている可能性がある．今回の評価対象である，公営住宅管理システムも，その様な性質を持つ業務であり，本論文で例示した税務処理も似たような傾向がある．応用ドメインのどのような傾向が，存在従属クラス図に向いているかは，更なる検証を必要とする．

更に，46 種類の帳票の View 的属性値の取得についても，属性追加等により，（家族情報の様な繰り返しを持つデータ項目を除いて）取得可能である事を確認した．図 10 は，その帳票のひとつである．この帳票に記載されている全ての View 的属性値を，ある一つのインスタンスから辿ることが可能か否かを検証した．

しばしば，モデルの不十分さから属性追加が生じた．しかし，関係モデルの特徴として，後から属性を追加することに困難性は感じなかった．更に，今回のアプリケーションドメイン特有かもしれないが，追加したクラスは，ほとんどが既存の上流側のクラスに対する特定の処理（従属している下流側クラス）であり，存在従属クラス図の基幹部分（上流部分）への大きな変更はなかった．

4.3 関連研究

ここで，関連研究について触れる．本提案手法は，存在従属クラス図を前提としている．存在従属クラス図自体は著者らの提案なので，著者らの知る限り，学術的文献には，類似研究はない．ただし，図7を構成するそれぞれの技術に大きな新規性があるわけではない．例えば，多重度=1の親子関係を用いて辿ると，辿り先のインスタンスが一個にとどまることは，教科書には記載が無くても，熟練設計者には知られた事実であろう．また，今回示したクラスの正規化条件も，オブジェクト指向の教科書に記載されている例を知らないが，熟練設計者には自然である．そもそも，ER図では，親子関係と参照関係を区別することは常識である．また，存在従属関連のみでクラス図が書けることは，佐藤正美のアプローチからも自明である[7]. 佐藤は, すべての関連リンクを対照表または対応表としてクラス化しているが，この関連リンクに近いクラスでは，リンクを張る双方のインスタンスが存在しなければリンクを張れないので，存在従属クラス図の一種となるからである．ただし，佐藤のアプロ―チは，全てのリンクをクラス化しているので，クラス数の増大は避けられない．

本論文のオリジナリティは，図 6 に示した構成要素を組み合わせた点にあり，結果的に，存在従属クラスのみで対象を記述する．これにより，クラス図全体の流れが読み取りやすく，しかも，下流側でインスタンスが生成された時点で，すべての辿り先インスタンスへのリンクを確定できる．以後，View 的属性値が必要となれば，このリンク情報に基づいて，極めて少ない処理ステップで高速に View 的属性を取得できると思われる．

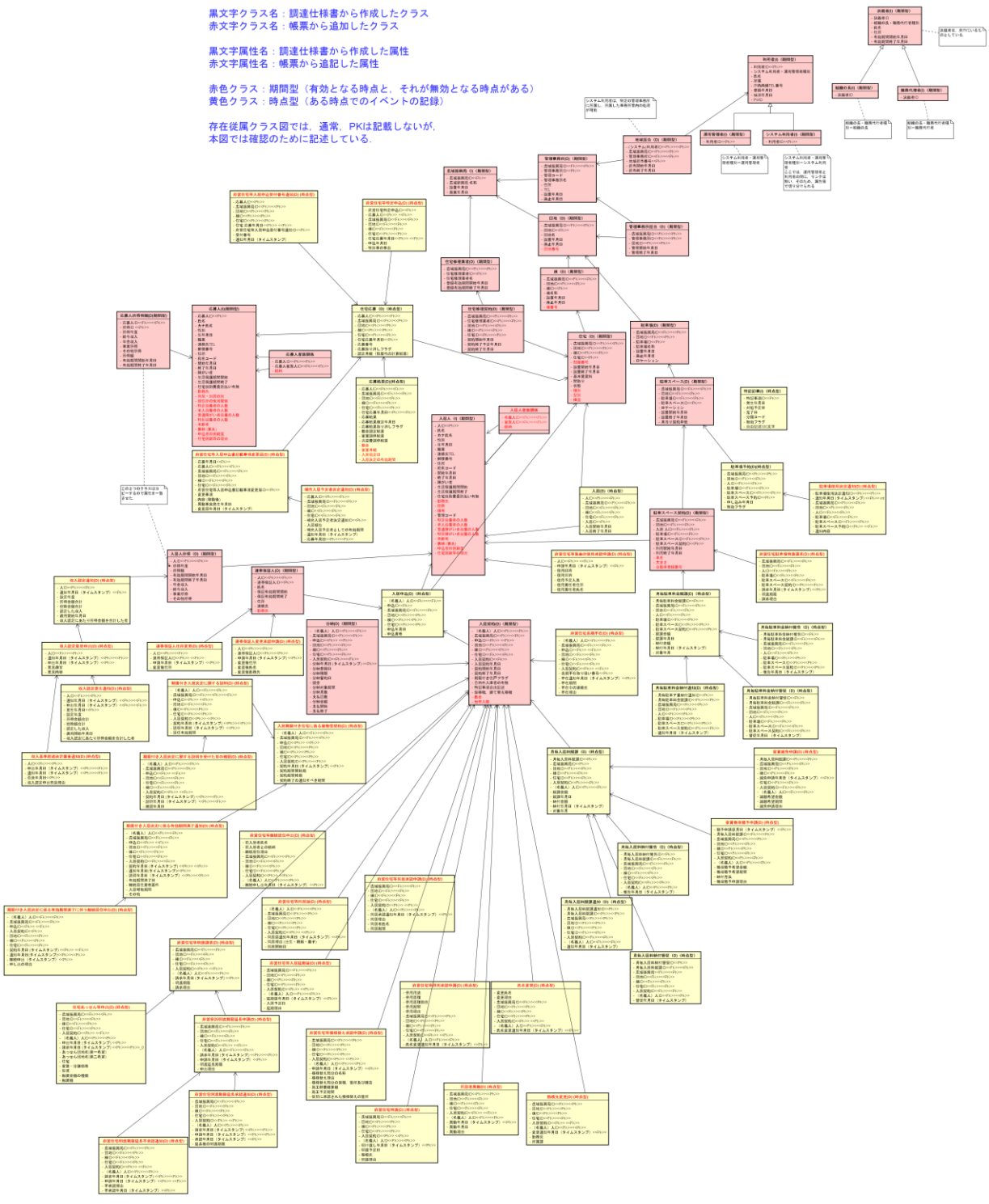
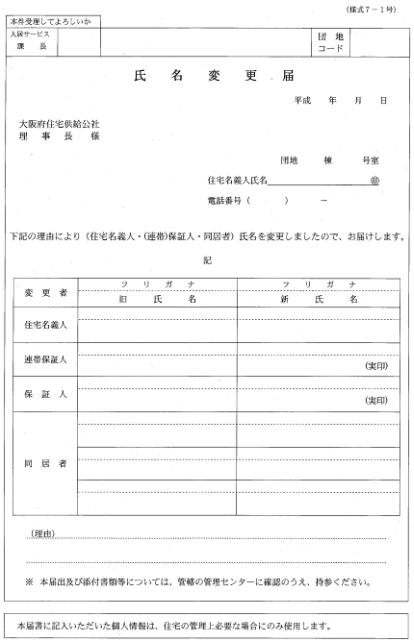


図 9　完成した存在従属クラス図[9]

[9] 図 9 では，（存在従属クラス図では記述しない）各クラスの PK 属性も表示している．また，ある期間有効な期間型クラスと，特定時点で発生したイベントを記録するイベント型クラスを色で区別している．ピンク色が期間型，黄色がイベント型である．更に，独立クラスか従属クラスかも，「I」「D」で示している．クラス名が赤字は帳票による評価により追加されたクラスであり，クラス名が黒字になっているクラスは，調達仕様書段階で作成したクラスである．属性が赤字なのは，調達仕様書段階で作成したクラスに，帳票によるモデル修正の際に属性追記されたことを示す．尚，帳票によるモデル修正で新規追加されたクラスに所属する属性は（すべてが追記を意味するが）黒字で示している．



（様式7－1号）

本件受理してよろしいか

入居サービス課長

団地コード

氏　名　変　更　届

平成　年　月　日

大阪府住宅供給公社
理　事　長　様

団地　棟　号室
住宅名義人氏名　㊞
電話番号（　）　－

下記の理由により（住宅名義人・(連帯)保証人・同居者）氏名を変更しましたので、お届けします。

記

変更者	フリガナ 旧氏名	フリガナ 新氏名
住宅名義人		
連帯保証人		(実印)
保証人		(実印)
同居者		

（理由）

※ 本届出及び添付書類等については、管轄の管理センターに確認のうえ、持参ください。

本届書に記入いただいた個人情報は、住宅の管理上必要な場合にのみ使用します。

図 10　帳票の例（大阪府の申請帳票から例示[6]）

4.4 考察と今後の課題

以上の評価によって，(1) 存在従属関連（リンク）のみで，対象ドメインは十分にモデリング可能であること，及び，(2)「多重度＝1」による追跡で，十分，業務処理に必要な属性値を取り出し得ることを確認できた．ただし，あくまでも今回はワンポイント評価であり，プログラム記述量がどこまで減るかとの実装評価，あるいは，本提案の趣旨を反映したプラットフォームの設計・構築等の更なる検証が必要と考える．

尚，本提案方式に基づいて，存在従属クラス図でエンティティレイヤーのクラス図を作成した経験からすると，「多重度=1 の存在従属関連では辿れないことが，作成されているモデルに対するチェック機能として働くのではないか」「ある業務処理を行うときに，どのクラスのインスタンスを中心に設計するべきかを容易に読みとれる」「あるクラスから辿ることのできる範囲を直感的に把握しやすい」等の点を確認できた．これらの性質は，AP 実装上，メリットとなる可能性がある．このあたりの検証も今後の課題である．

5. プロトタイプシステムによる評価

本論文は，実装については関知していない．AP 開発者の選択の範囲内にある．ただし，実装手法の提案なので，実装例を示すことには，提案手法の実現性を確認する意味で意義があると考える．また，本提案方式において，着目しているインスタンスから出発して，リンクを辿って他のインスタンスの属性値を取得する関数は，着目しているインスタンスのメソッドとして実装できる．更に，この関数はクラス図から自動生成できる．その意味では，提案手法では，View 的属性を取得するための関数群作成へのプログラマへの負担は一切無い．これに対して，従来方式では，辿り先のインスタンスが変更される事があるので，インスタンス間のリンクを辿る際には，タイミング（年月日時刻）を意識する必要がある．このため，従来手法では，現在のリンク状態で辿ることに限定する場合を除いて，アクセス関数の自動生成はできない．その意味では，従来手法では，プログラマへの負担が大きい．

以下，提案手法のプロトタイプとして，アクセス関数を自動生成するツールを構築して，ごく簡単なクラス図の例について，評価実験を行った結果を報告する．

5.1　既存クラス図による View 的属性の取得

最初に，従来方式による View 的属性の取得例を示す．図 11 は，既存の手法によるクラス図の例である．クラス図の表現内容については，説明は要しないと思われる．各クラスは，それぞれオブジェクト ID を持っている．2 個のクラスのインスタンス間に張られているリンクについては，リンクを表現する

ための外部キー（FK）を属性値としてインスタンスに持たせている．尚，クラス名にすべて「[既]」がついているのは，「既存方式」を意味しており，後述する存在従属クラス図（図 12）との対応を分かり易くするためである．

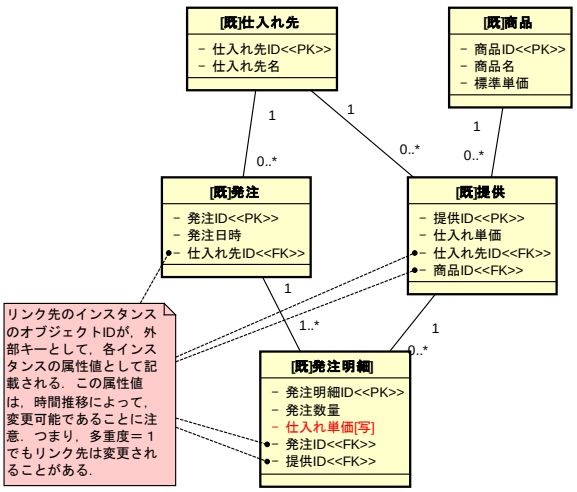


図 11　従来のクラス図の簡単な例

尚，図 11 では，「[既]発注明細」に，コピーされた属性である「仕入れ単価」が追記されている．この様なことはしばしば行われる．しかし，このコピー属性の存在は，正規形を破壊する．本章の議論では，完全に正規化されたクラス群を対象としており，図 11 の「仕入れ単価」は，図 11 では表示されているが，実際には無いものとして分析する．

5.2　既存方式における View 的属性の取得

次に，この正規化されたクラス構造の中で，「View 的属性」を取得する手間を見てみたい．まず，着目しているインスタンスは，「[既]発注明細クラス」に属するものとする．この着目しているクラス中の特定インスタンスを指定するために，「発注明細 ID」が所与であるとする．具体的には，ID 番号等である．この「発注明細 ID」により指定されたインスタンスから多重度=1 を辿って得られるところの，以下の View 的属性を用いて画面を生成する必要が AP 上で生じたとする．

・商品 ID，商品名，仕入れ単価，仕入先名，発注日時，発注数量

上記 View 的属性を取得するためには，AP 設計者は，以下の処理を実現するプログラムを書かなくてはならない．尚，着目しているインスタンスを「[既]発注明細インスタンス」と呼ぶこととする．

■商品 ID 及び商品名の取得（既存手法）■

(Step1) 「発注明細 ID」の指定に基づいて，「[既]発注明細クラス」から，「[既]発注明細インスタンス」を取り出す．第 1 回目のサーチ処理が実行される．
(Step2) 上記で特定された「[既] 発注明細インスタンス」が持つ「提供 ID」属性の属性値（FK）を読出し，「[既]提供クラス」から，その FK 値を PK 値として持つインスタンスを取り出す．第 2 回目のサーチ処理が実行される．
(Step3) 上記で特定された「[既]提供クラス」のインスタンスが持つ「商品 ID」属性の属性値（FK）を読み出し，「[既]商品クラス」から，その FK 値を PK 値として持つインスタンスを取り出す．第 3 回目のサーチ処理が実行される．
(Step4) 上記で特定された「[既]商品クラス」のインスタンスが持つ「商品名」属性から商品名を得る．AP はこの商品名を得られた結果として表示する．

以上のステップでは，合計 3 回のクラス内サーチを実行している．同じ View 的属性値であっても，「仕入れ単価」「発注日時」では，サーチは 2 回である．一方，「仕入れ先名」では，サーチは 3 回必要である．「発注数量」では，「[既]発注明細クラス」のインスタンスを探すだけなので，サーチは 1 回である．

5.3 既存手法の問題点

図 11 の既存クラス図におけるリンクの「辿り」には一つ課題がある． View 的属性を辿る作業は，「いつの時点でのリンク関係かを意識する必要があり，プログラマへの負担となる」点である．たとえば，上記図 11 で，「[既]提供クラス」の「仕入れ単価」属性が時間的に適宜変更されているとする．この場合，リンクを辿る作業を現在のデータ状態における「[既]提供クラス」のインスタンスで行うとおかしなことになる．本来は，発注明細を作成したタイミングにおける「[既]提供クラス」のインスタンスの属性を求めないといけない．これが，正規形を崩しても，View 的属性のために，コピー値を持つ属性を作りたくなる理由の一つと思われる．コピー値を持つ属性を作らないで正規形を担保すると，以下のような状況が生じる．

・リンクを辿るコードを書く場合，AP に関するドメイン知識を使って「いつの時点のデータ状態で辿るか」を意識してコーディングする必要があり，実際には反映が難しい．
・だからと言って，現在のデータ状態で辿ると，リンク状態が変わってしまっているので，不正な属性値を返すことになりかねない．
・一方，非正規化して，属性値をコピーすると，例えば，コピー元が修正された場合の処置が必要となり，開発者の負担となる．

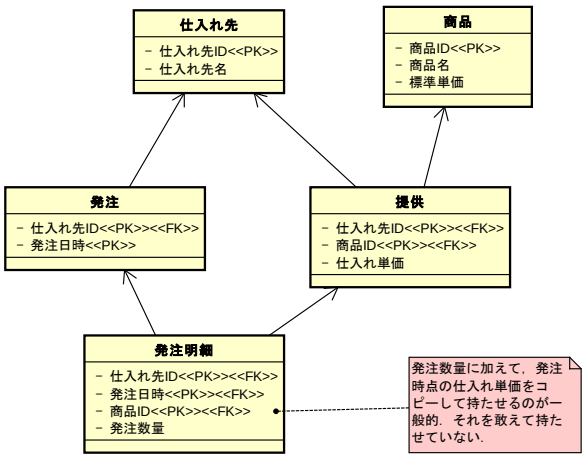


図 12 存在従属クラス図のサンプル

5.4 本提案のクラス図による VIEW 的属性の取得

図 12 は，同じモデルを存在従属クラス図で表現した結果である．着目しているインスタンスから，リンクを辿って一意にたどり着けるインスタンス（一般には複数）を指定するための外部キーを，各インスタンスは最初から保有している．図 11 と同様に，図 12 において，以下の VIEW 的属性を取得する必要があるとする．

・商品 ID，商品名，仕入れ単価，仕入先名，発注日時，発注数量

上記の View 的属性を取得するためには，AP プログラマは，以下の処理を実現するプログラムを書かなくてはならない．ただし，これは，定型処理なので，存在従属クラス図から自動生成できる．着目しているインスタンスを，「発注明細インスタンス」と呼ぶ．

■商品 ID 及び商品名の取得（提案手法）■

(Step1) 「発注明細 ID」に基づいて，「発注明細クラス」から特定のインスタンスを取り出す．第 1 回目のサーチ処理が実行される．
(Step2)上記で特定された発注明細インスタンスが持つ「商品 ID」属性の属性値（FK）を読み出し，「商品クラス」から，その FK 値を PK 値として持つインスタンスを取り出す．第 2 回目のサーチ処理が実行される．
(Step4) 上記で特定された「商品クラス」のインスタンスが持つ「商品名」属性から商品名を得る．AP はこの商品名を得られた結果として表示する．

今回は，サーチは 2 回であり，既存手法の 3 回から 2 回に削減されている．リンクを辿る上で，中間に存在するクラスが 1 個だったので，サーチ回数は 1 削減されただけであるが，一般にリンク上に n 個のインスタンスがあれば，n-2 回のサーチが削減される．尚，「仕入れ先名」についても，2 回に削減される．

以上の分析から，以下のことが分かる．

・ターゲットとなるインスタンスに直接ジャンプしているので，中間のリンクを辿るタイミングを意識する必要がない．ターゲットとなるクラスにおいてバージョン管理が行われていれば，そのタイミングのみを指定してデータを取得すればよい．
・各クラスにおいて有効な時間的区間が示されたバージョン管理が行われていれば，各インスタンスに辿り先のインスタンスの属性値を読み出すメソッドを準備し，属性値を必要となるタイミングを指定可能とすれば，任意の時点での View 的属性を取得できる．
・上記のメソッドは，AP のドメイン知識が無くても，クラス図から機械的に自動生成できる．

上記の議論からわかるように，存在従属クラスでは，ターゲットインスタンスの属性値を取り出す関数を自動生成できる．DB 定義, DAO は元々自動生成できる．すべてのクラスのインスタンスに関して，履歴管理をして，任意の時点での値を取り出せるようにしておけば良い．以上の分析に基づいて，プロトタイプシステムを構築した．以下にその概要を述べる．

5.5 プロトタイプシステム

提案手法の実現性と処理性能を確認するため，プロトタイプシステムを構築した．ただし，実現法としては種々考えられるが, 本プロトタイプシステムでは, 構築容易性を考慮して, リンクを辿る処理は, DBMS（MySQL）で実現することとした．図 13 には構築したシステムの概要を示す．大きな特徴は，存在従属クラス図を Astah*で描くと，クラス図の情報を XML 形式で取り出し，そこから，（1）DMBS の CreateDB コマンド等の DB 定義，（2）Java で記述された AP のオブジェクト群の Java ソースコード，（3）オブジェクト群と DBMS を連携する DAO のソースコード，（4）DAO がカプセルとして利用する JavaBeans ソースコード，をすべて自動的に生成する．

即ち，前節で説明したメソッド群は，自動的に生成されるので，プログラマは関与する必要がない．リンクを辿る処理は SQL 記述に従って MySQL により行われるが, 必要となる SQL 記述は DAO の中で自動生成され，必要に応じて，DAO から MySQL に渡されて処理される．これにより，従来は煩雑なルーチン群を手書きしなければならなかったのに対して，提案手法では，一切，プログラムする必要はない．各インスタンスには，View 的属性を取得する関数が具備され，しかも，ターゲットインスタンスの値は，必要なら，特定の時刻を指定して，その際のデータが取得できる．

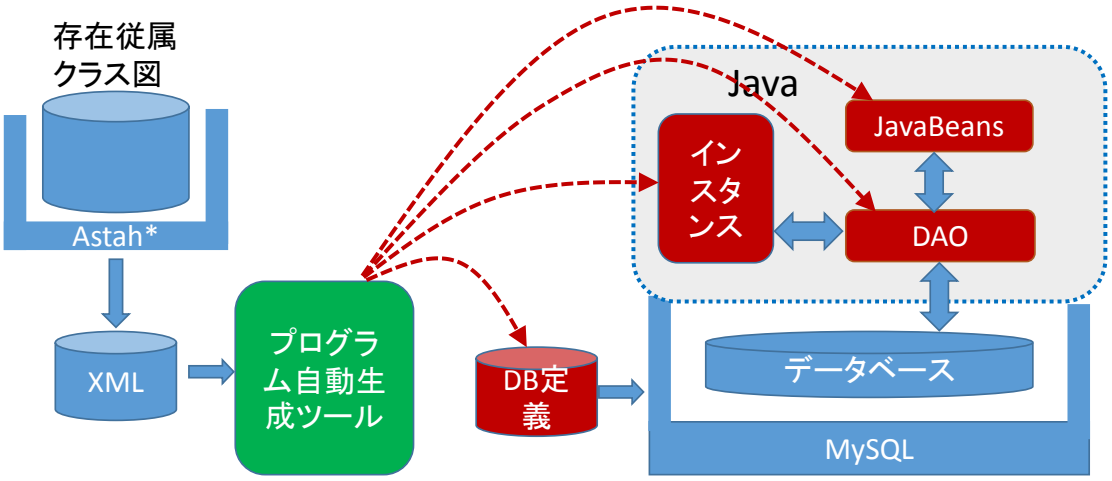


図 13 プロトタイプシステム概要

5.6 性能評価実験

提案手法の性能を評価するため，図 12 のクラス図について，図 13 のシステムで自動的に AP 基盤を生成し，動作するプログラム（メソッド）を自動生成可能であることの確認と，どの程度の性能で動く

かを評価した．ただし，性能評価は，後述の理由で，DBMS 内部での処理時間のみを本論文では示す．

テストデータ（図 12）の作成条件は以下の通りである．但し，バージョン管理されているクラスでは，有効性開始タイミングと有効性終了タイミングをもつインスタンスがつぎつぎと作られている．即ち，実質上 1 個のインスタンスが，多数（730 個）に展開されている．バージョン管理を考えないインスタンス数は以下の通りである．

・「仕入先クラス」のインスタンス：7 個
・「商品クラス」のインスタンス：8 個
・「発注クラス」のインスタンス：7 個
・「提供クラス」のインスタンス：56 個（仕入先と商品のすべての組み合わせ）
・「発注明細クラス」のインスタンス：発注クラスのインスタンス毎に 8 種類の商品を発注

実際には，バージョン管理を導入しているので（「仕入れ先クラス」を除いて），実際のインスタンス数は多い．以上のクラス中で，「仕入れ先クラス」については，バージョンは作らない．「商品クラス」については，2016 年 4 月 1 日から 2018 年 3 月 31 日の間(730 日)、商品名の一部をランダムに変更し，それらを登録した．総レコード数は，5840 件（8×730）となっている．「発注クラス」についても， 2016 年 4 月 1 日から 2018 年 3 月 31 日の期間，毎日 7 社の仕入先に発注したデータを用意した．レコード数は 5,110 件（730×7）である．「提供クラス」は，2016 年 4 月 1 日から 2018 年 3 月 31 日の間(730 日間)，仕入単価は，商品の標準単価から毎日 1 円ずつ値上がりしたデータを用意した．レコード数は，40,880 件(56×730)である．「発注明細クラス」は，「発注クラス」のインスタンス毎に，8 種類の商品を 1 個ずつ発注した発注明細クラスのインスタンスを用意した．レコード数は 40,880 件（5,110×8）である．

5.7 評価実験の結果

プロトタイプシステムの評価環境は，以下の構成である．
OS: Windows10
CPU: Intel Core i5-5400 2.70GHz
メモリ:8GB
DBMS : MySQL

View 的属性値を取得するためのメソッド（自動生成）は，Java で書かれているが，内部で SQL コマンドを生成する．これを DBMS に渡して，SQL コマンドとして実行する．リンクを辿って，属性値を得るための SQL の例を以下に示す[10]．ただし，クラス名はアルファベット名に変更されている．「発注明細クラス」の「発注明細 ID」を所与の具体値として指定している．この SQL は，あくまでも，特定のタイムスタンプにおける，辿り先インスタンスの属性値を要求している．クラス中でのサーチは 2 回発生する．以下の SQL を含む，自動生成されたメソッドが正常に動作することを確認した．

```
select goodsNAME
  from goods, orderdetails
  where orderdetailsID = 20400
  and goods.goodsID = orderdetails.goodsID
  and goods.validtime <= orderdetails.validtime and orderdetails.validtime < goods.invalidtime;
```

同じ処理を，既存手法の様に，つぎつぎとリンクを辿る SQL で描くと，以下の表現となる．この SQL は手作業で作成したものである．ただし，あくまでも「現在のインスタンス間のリンク構造」に準拠するので，一般的には，AP にはそのままでは適用できない恐れが強いが，評価のためにあえて作成した．クラス内のサーチは 3 回ある．

```
select goods.goodsNAME
  from orderdetails,offers,goods
  where orderdetails.orderdetailsID = 20400
  and orderdetails.offersID = offers.offersID
  and offers.validtime <= orderdetails.validtime and orderdetails.validtime < offers.invalidtime
  and offers.goodsID = goods.goodsID
```

[10] Java のメソッド実行では，この SQL は動的に生成されて，DBMS に渡される．

and goods.validtime <= orderdetails.validtime and orderdetails.validtime < goods.invalidtime;

上記の２個の SQL を含む Java メソッドを起動して所要時間を比較したい．ただし，以下の理由により，Java レベルでは単純には比較できない．

・DBMS のキャッシュを無効化した状態でも，ハードディスクのキャッシュ効果は存在し，システムを再起動した後の SQL 実行の第 1 回目と２回目を比べると，2 回目は明らかに高速であり，1 回目と 2 回目で，CPU 時間も大きく異なる．

・Java によるメソッドからアクセスすると，最初のメソッド実行では，DBMS の開始に 200ms～300ms 程度の時間を必要としており，SQL 部分のみを取り出すことができない．

・逆に Java によるメソッドの 2 回目で SQL の実行時間の測定したいところだが，今度は，ハードディスクのキャッシュが動作した後なので，1 回目の様な，ハードディスクキャシュの効果を外した測定にならない．

以上のことから，上記の 2 個の SQL の DBMS 上での実行時間を調べると以下の通りであった．尚，測定は，それぞれ，再起動後として，ディスクキャッシュの影響を排除している．

従来手法： 94.5ms

提案手法： 62.5ms

サーチの回数が，従来手法は 3 回，提案手法は 2 回であるので，所要時間の比率 3 : 2 は妥当と考える．

5.8 考察

前節の評価結果から分かるように，提案手法と従来手法では，クラス内サーチの回数にほぼ比例した所要時間となる．処理時間的には，オーダーが変わるわけではないので，効果が限定される．むしろ，提案手法のメリットは，バージョン管理が簡単にできて，辿り先のインスタンスの何時の属性値が欲しいかを指定すれば，確実にその時点での属性値が入手できる点にある．

一般に業務処理では，椿正明が「要約」「断面[11]」と呼ぶデータ構造が多用される．例えば，過去や今週の，ある一週間での各支店の売り上げ一覧等である．基本的に，AP 構築では，これらのデータは，その時点で作成しておき，保存するアプローチを取ることが多い．しかし，本論文のアプローチでは，バージョン管理が極めて簡明に実現されるので，「断面」の画面が必要となる時に，仮想テーブルの様に，SQL コマンドを実行すれば良いと思われる．実際，今回のプロトタイプ評価でも，ハードディスクキャシュは，select 文の実行時間に大きな影響を与えている．すなわち，一度データをキャッシュすると，それ以降の SQL 処理は極めて高速である．その意味でも，本提案手法によれば，あらかじめ準備しておいた SQL 記述を，必要なタイミングで起動することで，「断面」「要約」を生成できる可能性がある．

言うまでもなく，「断面・要約」動的生成の性能については，実用規模アプリケーションにおいて，どの程度のレスポンスが得られるかを確認する必要がある．但し，今回の評価では HDD を用いている．これに対して，SSD は HDD よりも 2 桁位高速な読出しアクセスタイムを持つ．また，インタフェースとしても，従来の SATA に比べて大きなスループットを持つ PCI-e インタフェースの SSD が登場している．加えて，SSD の容量は既に 4 TB を越えつつあり，容量的にも十分な値である．SSD の採用により，更にレスポンスタイムを短縮できることが期待される．

本来，「断面・要約」は，RDB の立場からすれば View 的属性の塊である．View 的属性を管理するために，わざわざデータ構造を作ってパーシステント化するのは，本来の正規化設計の立場からすればおかしな話しである．加えて，テーブル構造を非正規化してプログラムステップを増加させた時の開発コストおよび保守コスト（人件費）の増加は，容易に，SSD のコストを超過して行く．その意味でも，本論文に示した様な，徹底的な正規化設計は，今日的な意味があるのではないかと考えている．

6. まとめ

本論文では，(1)正規性を満たす存在従属クラスを前提として，(2)View 的属性は「多重度= 1 」によりリンクを辿って得ることとし，(3)任意の時点での属性値を得るため，インスタンスをバージョン管理する，アプリケーションプログラム（AP）構成法を提案した．提案手法によれば，インスタンス生成時に，辿り先のインスタンスへのリンク（例えばポインタ）を一意に決定でき，以後，中間に位置するインスタンス群を経由せずに，任意の時点における View 的属性値を，辿り先のインスタンスから直接に取得できる．その結果，各クラスのインスタンスをそれぞれバージョン管理しておけば，辿り先のインスタンスから属性値を取りだす時点情報を指定することにより，必要な時点での属性値をアクセス出来る．この枠組みは，AP の構成を簡明にするだけではなく，処理の高速化が期待される．

[11] 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 11 月

提案手法の実現性を確認するため，自治体の公営住宅管理システムの調達仕様書を参考に作成された，存在従属クラス図を用いて，実帳票に出現する View 的属性を取得できるか否かを評価した．存在従属クラス図は最終的に 77 クラス，（主キーを構成する属性を除いて）311 属性となった．作成には特段の困難性を感じることはなかった．また，46 枚の帳票を対象として，記載された View 的属性を，提案方式でアクセスできるか否かを確認した．その結果，一意に特定される属性値については，提案手法で漏れなく取得できることを確認した．一意に特定できなかった View 的属性としては，例えば，家族一覧の様な，繰り返し性があるデータ項目があった．しかし，この場合も，家族を検索するための特定インスタンスを取得できるので，AP 処理として，一覧データを取得することは難しくない．

今後は，提案方式を実現するプラットフォームの検討，アプリケーションを実装したプログラム量の比較検討等が必要と思われる．尚，本研究を進めるに際して，住宅管理システム帳票類のご提供をいただいた，自治体関係者に深謝いたします．

本論文は，電子情報通信学会・知能ソフトウェア工学研究会の資料「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」（信学技報, vol. 117, no. 465, KBSE2017-44, pp. 31-36, 2018, 3 月）に内容を追加して書かれたものである．信学技報から引用された部分については，著作権は電子情報通信学会に帰属するので，ご注意願いたい．末筆ながら，著作権利用許諾（許諾番号：18GB0089, 2018 年 9 月 26 日付）をいただいた，電子情報通信学会に深く感謝いたします．

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, pp.1554-1568, 2004 年 6 月
[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, pp.1340-1350，2015 年 5 月
[3] 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
[4] 金田重郎, 劉湘涛, 森本悠介, 井田明男, “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”，情報システム学会・第 13 回全国大会研究発表大会，S2-B.1，2017 年 12 月
[5] 劉湘涛，森本悠介，井田明男，金田重郎，“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”，信学技報, vol. 117, no. 465, KBSE2017-44, pp. 31-36, 2018 年 3 月
[6] https://www.osaka-kousha.or.jp/customer/kousha.html(2018 年 2 月 17 日確認）
[7] 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 年 4 月
[8] 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 D, Vol.J100-D, No.1, pp.14-27，2017 年 1 月
[9] 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ー行為文と関数従属性に基づくクラス図の作成ー”，電子情報通信学会論文誌 D，Vol.J98-D，No.7，pp.1068-1082，2015 年 7 月
[10] 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 11 月

著者略歴

金田 重郎（かねだ　しげお）

1974 年 3 月京都大学工学部電気第二学科卒業，1976 年 3 月京都大学大学院工学研究科電子工学専攻修士課程修了．1976 年 4 月日本電信電話公社・武蔵野電気通信研究所入所．1997 年 4 月同志社大学大学院総合政策科学研究科教授・同工学部教授．現在，同・理工学研究科・情報工学専攻教授．

井田 明男（いだ　あきお）

1984 年同志社大学文学部文化学科（心理学専攻）卒業．銀行員，大手メーカ系 SE を経て，2012 年に独立．有限会社井田代表取締役，現在に至る．同志社大学理工学部講師．

森本 悠介（もりもと　ゆうすけ）

2013 年 4 月同志社大学理工学部インテリジェント情報工学科入学，2017 年 3 月同卒業，2017 年 4 月同志社大学大学院理工学研究科情報工学専攻入学，現在に至る．

劉 湘涛（LIU Xiangtao）
2016 年 4 月同志社大学大学院理工学研究科情報工学専攻入学，2018 年 3 月同修了.

上野 洸史（うえの　こうじ）
2013 年 4 月同志社大学理工学部インテリジェント情報工学科入学，2018 年 3 月同卒業.