存在従属に基づくソフトウェア仕様記述のイベント順序検証手法
Event Sequence Verification by using Existence Dependency

金田重郎† 井田明男†
Shigeo KANEDA Akio Ida
†同志社大学・理工学部
†Faculty of Science and Engineering, Doshisha University.

要旨

著者らは,「存在従属」をオブジェクト指向分析(OOA)に導入することを提案している．本稿では，存在従属の応用の一つとして，情報処理システムの処理シーケンス（ユースケースのイベントフロー）を検証・生成する手法を提案する．具体的には，クラス間の存在従属関係を非サイクリック有向グラフ（DAG）と見なし，外部から与えられた自然言語による仕様記述とは無関係に，DAG の上流から下流に流れるイベントフローを生成する．そして，この生成されたイベントフローを外部から与えられた仕様記述と比較・照合することにより，自然言語から生成されたイベントフローが持つ無駄・矛盾を検証し，仕様記述を修正する．

1. はじめに

あるクラスに属するオブジェクトの生成･存在が,他クラスのオブジェクトの存在を前提とする場合，2 つのクラスの間には「存在従属」関係が存在する．例えば,「商品を製造社に発注する」との仕様記述に基づいて,「発注」オブジェクトを作成する場合を考える．「発注」オブジェクトは,「商品」「製造社」の両オブジェクトが既に存在しないと作成できない．この際，通常のデータベース設計法に従えば,「発注」オブジェクト（テーブル）の主キーは,「商品 ID＋製造社 ID」（実際には,「イベント」なので，これにタイムスタンプが加わる）となる[4]．ところが，オブジェクト指向分析(OOA)では，伝統的に，上記の様な複数属性からなる識別子は使われない．一般的には，オブジェクトにはクラス特有のオブジェクト ID があると見なされる．つまり，各クラスの主キーは単一属性である．

クラス図のロバスト性を担保するには，クラスは第 5 正規形でなければならない[5][6]．しかし，各クラスにそれぞれ固有の ID を与えて識別子（主キー）としてしまうと，複数属性による主キーを許すデータベース設計法では避けられるはずの多値従属性や論理的一貫性要件を見過ごす恐れがある．オブジェクト指向分析においても，存在従属が存在する場合には，存在従属の上流側の主キーを，下流側のオブジェクトの主キー（またはその一部）として取り込み，より高度な正規形であることを保証するべきである[1][2].

存在従属とは，オブジェクト間の制約関係である．上流側のオブジェクトを先に作らないと，下流側のオブジェクトが生成できない．そうであるなら，存在従属関係は，情報システムのユースケースの処理手順（イベントフロー）と密接に関連しているはずである．そこで，本稿では，クラス間の存在従属関係を非サイクリック有向グラフとみなし，これによってイベントシーケンスを生成する手法を提案する．これによって，クラス図を作成する元になったイベントフロー（ユースケース記述）の妥当性を検証でき，処理順序の修正や抜けの補完が可能となる．

以下，第 2 章では存在従属について説明し，第 3 章では，そのイベントフロー生成への適用方法を示す．ここでは，存在従属関係を非サイクリック有向グラフ(DAG: Directed Acyclic Graph)と認識し，オブジェクト CRUD(Created Read Update Delete)処理における制約関係を分析する．第 4 章では，一例として簡単な税務処理に関する実例を示す．第 5 章はまとめである．

2. 存在従属

最初に図 1 の例を用いて，存在従属関係を説明しておく.「仕様記述としては,「(会社は）商品を製造社に発注する」である.「モノーコトーモノ」パタンの例である．図 1 は，椿正明[4]の TH モデルにおける「リソース型」と「イベント型」のクラス（エンティティ）を含む，この中で，リソース型クラスで

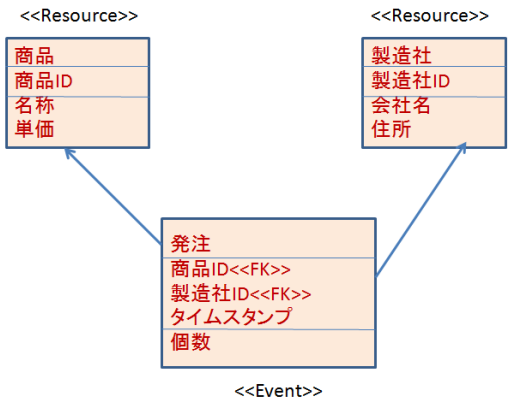

図 1　「商品を製造社に発注する」クラス図

は，他オブジェクトの有無に関係なく，オブジェクトを生成できる．具体的には，「商品[1]」「製造社」はリソース型であり，「発注」オブジェクトはイベント型である．リソース型クラスの識別子（主キー）は，当該クラス固有のものであるので，図 1 に示すように，ID を個別に与えている．

「発注」クラスはイベント型であり，事前に商品と製造社が決まっていないと生成できないオブジェクトである．このイベントオブジェクトは，同じ商品・製造社の組み合わせに対して，何度も発注できるので，オブジェクト生成には，発注時点のタイプスタンプが必要である．従って，識別子（主キー）は，「商品 ID+製造社 ID+タイムスタンプ」となる．この場合，商品オブジェクトと製造社オブジェクトがそれぞれないと，発注オブジェクトは生成できない．この事を図 1 では，方向性（矢印）の関連表記で示している．「矢じり」が付いたほうが，存在従属の上流側（存在が前提とされるオブジェクト側）である．

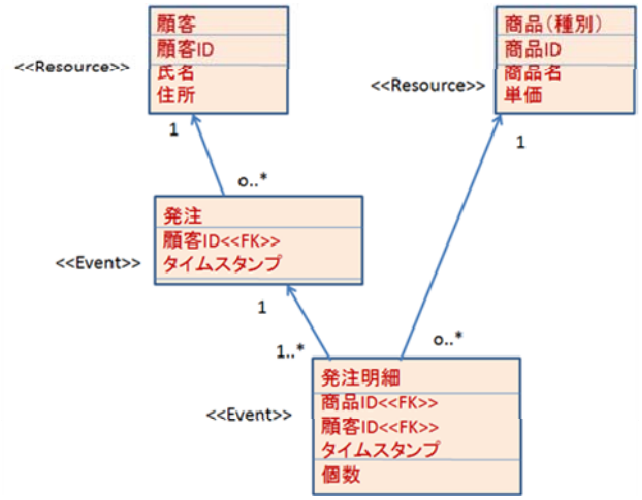

図 2 存在従属関係を持つクラス図

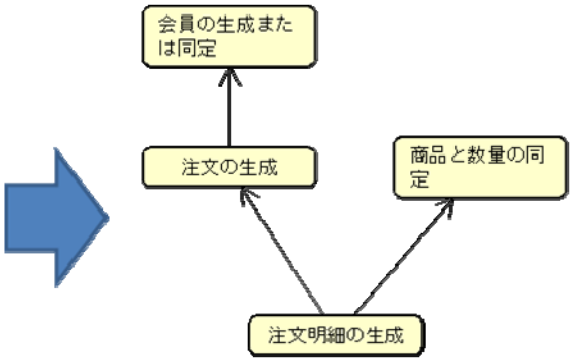

図 3　生成されたアクティビティ・ディペンデンシー・グラフ（活動依存グラフ）

3.　存在従属に基づくイベントフローの生成

3.1.　イベントフロー生成の原理

図 2 を用いて，クラス図とイベントフローの関係を考察する．図 2 は，ユースケース「会員が商品を（当社に）注文する」が正常終了した際に作成されるクラス図である．ユースケースの事後条件として，ユースケース完了後は，クラスの構造が図 2 の様になっていることが要求される．事後状態に持って行

[1] ここでは，戸別の商品では無く，商品の種類を表現するクラスである．

く為には

（1）注文の生成に先立って，会員の生成，または，同定が完了している必要がある．

（2）注文明細の生成に先立って，注文の生成，かつ，商品と数量の同定が完了している必要がある．

ことが分かる．従って，それぞれのクラスから，それを生成するためのアクション（動作動詞）を取り出し，それらを（1）（2）を満たすように，トポロジカルソートすると図 3 の様なアクションの依存グラフが得られる．図 3 から，最初に処理するべきなのは，会員の同定である，それができれば，発注を確定，次に，発注明細を作るが，そのためには，予め，商品と個数を同定する必要があることが自動的に分かる．あとは，ユースケース記述のイベントフローを確認して，自然言語のユースケースの持っているイベントフローに対して，矛盾や抜けはないか，を検証すればよい．不合理な部分があれば，元となっているユースケース記述そのものを修正することになる．

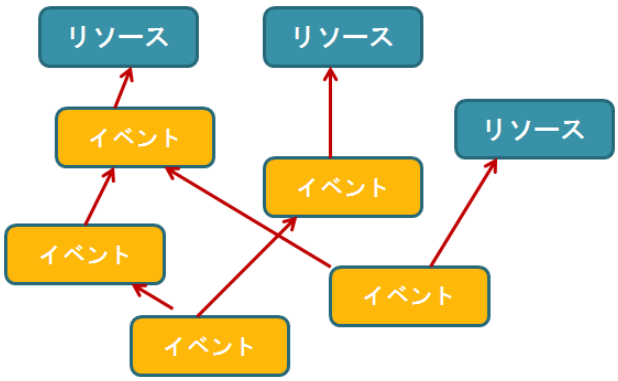

図 4 存在従属が形成する非サイクリック有向グラフ

3.2. 非サイクリック有向グラフ(DAG)

一般的な議論をするために，図 3 をより抽象的なレベルで捉える．存在従属は，クラス間に関連を持つが，この関連は，有向であり，しかもサイクルを持たない．図 4 は存在従属関係が構成するグラフのイメージを摸したものである．このグラフに出現する，全てのクラス（オブジェクト）が処理の対象となり得る．ただし，オブジェクトの状態を変化させる操作は，以下の 3 種類しかない．CRUD のリード（R）は，オブジェクトに影響を与えないので，除外した．

C（CREATE）：オブジェクトまたは属性値の生成
U（UPDATE）：オブジェクト中の属性値の変化
D（DELETE）：オブジェクトの消去

オブジェクトに対する CUD 操作では，以下のイベント追加が要求されることは自明である．

(1) CREATE は，リソースオブジェクト生成として行われる場合と，（上流側の）既存オブジェクトに存在従属したイベントオブジェクト生成（上流側オブジェクトは既に存在していなければならない）の 2 通りがある．イベントオブジェクトの場合には，必要となる上流側の既存オブジェクトの同定（あるいは必要なら作成）を行ってから，ターゲットオブジェクトの生成を行うイベントフローとなる（図 2，図 3 はその一例である）．

(2) UPDATE は，既存のオブジェクトに対してのみ実行できるが，当該修正が，当該既存オブジェクトに従属したオブジェクトに影響を与える場合には，下流側オブジェクトの処置が必要である．アップデートされる属性値がどのような影響を下流側に与えるかは，業務内容と下流側オブジェクトの状況にも依存するので一概には言えない．しかし，アップデート処理がユースケースとして行われ

る場合には，ユースケース記述には書かれていなくても，下流側のオブジェクトの有無確認と，（存在している場合には）影響調査と処置をイベントとして行う事が要求される．

(3) DELETE は，UPDATE と同様であり，下流側のオブジェクトをすべて消去してからでないと，当該オブジェクトを消去できない．これも，下流側のオブジェクトの状況によりイベントフローは変化する可能性はある．しかし，ユースケース記述に書かれていなくても，下流側オブジェクトの存在確認と必要であれば消去は，イベントフローに取り込む必要がある．

4. 存在従属に基づくイベントフローの生成

提案手法を簡単な事例に適用して，妥当性を検討する．図 5 は，自動車税と軽自動車税の処理のためのクラス図（極めて，簡単化された例）である．ただし，自動車税は道府県税，軽自動車税は市町村税であるので，ひとつの自治体の税務システムにおいて，この 2 つの税務処理を並行処理することは通常ではおきない[2]．ここでは，あくまで，分析用事例である．

図 5 を見ると，存在従属関係によって，自然にイベントフローが生成される．たとえば，何らかの理由で，自動車税オブジェクトを消去する場合には，督促状（自動車税督促）が存在すれば，それを消去してからでないと消去できない．ただし，督促状に更に，催告状（自動車税催告）がぶら下がっている場合には，これも消去する必要がある．更に，自動車税消去のイベントの前には，必ず，住民（自動車の所有者）の同定と自動車の同定を必要とすることが分かる．

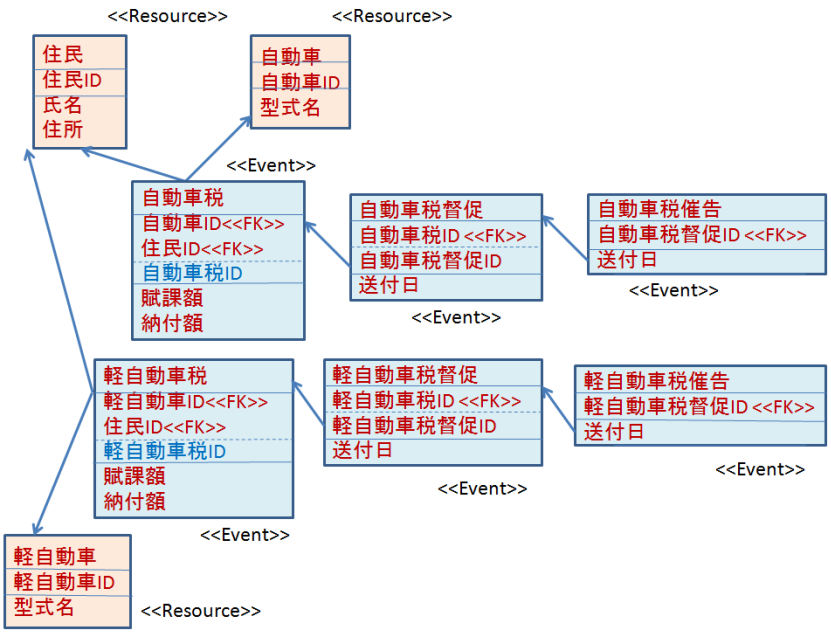

図 5　自動車税・軽自動車のクラス図（例）

課題は，自動車の確認イベント，及び住民の確認イベントを，自動車税消去のイベントフローに入れるかどうかである．自動車のほうは，図 5 を見る限り，自動車税の処理には必ず利用して，他には利用されるパスがないので，自動車税消去のイベントフローに入れておいても良いと思われる．しかし，住民の同定イベントは，自動車税と軽自動車税の双方に利用されるので，別のユースケースとして独立させるべきと思われる．即ち，存在従属の有向グラフ（DAG）が存在すれば，以下のことが，半自動的に導出できる．

（1） どの部分を共通したイベントフローとして独立させるべきか，あるいは，あるイベントを単一の

[2] 理論的には東京都では起こり得る．

イベントフローに取り込むべきかを判断できる．

（2） 大まかなイベントフローが自動的に生成されるので，これを，分析のもととなっているユースケース記述と照らし合わせ，ユースケース記述自体の抜けや矛盾を指摘できる．結果として，シンプルでロバスト性の高い，イベントフローにユースケース記述のイベントフローを改善できる可能性がある．

（3） 内部統制に優れたイベントフローを記述できる．すなわち，架空のリソースやイベントに存在従属するような取引を登録することははじめから不可能である．

図 5 では，実装上の課題として，イベントクラスでも，別途，オブジェクト ID を表記している．上流側のクラスもイベントクラスである場合については，上流側の持っている複数属性を下流側が引き継ぐのではなくて，上流側イベントクラスのオブジェクト ID を引き継いでいる．これによって，どのクラスが上流側（存在従属の元となるクラス）なのかを特定できるようにしている．これをしないと，例えば，図 5 で，「自動車催告」の主キー属性として，「自動車 ID+住民 ID」となって，「自動車税督促」の上流側が「自動車税」ではなく，2 つのクラス，住民と自動車に見えてしまうからである．存在従属関係は，あくまで，本当に意味的に従属し得ているクラス間でポインタを張るべきと考える．

尚，図 5 において，オブジェクト ID の代わりに，オブジェクトへのポインタ（仮想空間上のアドレス）を利用できる状況になった場合の実装についても，今後は，考察が必要と思われる．図 5 においては，オブジェクト ID をポインタとすれば，少なくとも，下流側から上流側はコンスタントタイムでアクセスできる．ただし，これだけでは，上流側から下流側へのポインタが存在しない．しかし，これも，実装上，容易に対応できる．下流側のインスタンスを生成したタイミングで，そのポインタを上流側のインスタンスにアタッチすればよいだけである[3]．双方向のポインタアクセスが可能であるなら，3.1 節において示した DAG 構造の追跡は極めて短い処理時間で実現できる．近い将来，メインメモリーが PRAM や MRAM で不揮発化した場合には，検討に値するアプローチと考えられる．

5. まとめ

クラス図における存在従属グラフを利用して，ユースケースのイベントフローを生成することを提案した．これは，著者らの「事務手続きは，人間が持っているものではなくて，処理されるべきデータが持っている意味的制約関係から生成されるものである」とのひとつの考え方に基づくものである．しかし，提案手法について，実用レベルの適用実験は行われておらず，今後の課題である．

なお，本稿では，椿正明の言う意味で，「リソース」「イベント」クラスのみを対象としている．しかし，椿正明による TH モデル[4]では，一種の View とも理解される在庫型，要約，断面のクラスが必要となる．従来の業務システム設計では，リソース/イベントクラスに関連した処理の中に，これら View 的なイベントも組み込むことがしばしば行われている．ユーザビリティを重要視する意味から，結果として，ひとつのユースケースの中に，これら集約的な処理を混ぜることは，止む負えないものとしなければならないのかもしれない．しかし，たとえ，そうであっても，内部処理としては，View 的なイベントの実装は，区別して考えるべきと思われる．そして，明らかに，要約，断面のクラスは，イベントクラス（場合によっては，リソースクラス）に存在従属するものと思われる．

尚，上記のようなイベントフローの自動生成を行うと，まず，問題となるのは，「ユーザビリティが悪いのではないか」との懸念であろう．「快適なフローは人の側にある」とするのが，一般的な業務処理システムの考え方だからである．しかし，これも本当にそうなのであろうか．オブジェクト指向の黎明期に，オブジェクト的な捉え方を支えた，極めて有名な認知心理学の理論に，ミンスキーの「フレーム理

[3] 実際，Observer パタンでは，新しく生成された View（Observer）は，オブザーブする対象である Model(Observable)に対して，自分自身のポインタ（self あるいは this）を差し出して，aModel.attach(self)を要求する．それに応じて aModel（観察対象のインスタンス）は，受け取った self を自分が持っている Observer 型のポインタキューに追加する．

論」がある[7]．フレーム理論では，人間の連想は，あるオブジェクトの属性や属性値に視点が移動し，その属性値が示すオブジェクトに興味が移り，というのが人間の認知モデルであったはずである．仮にそれが正しいのなら，むしろ，オブジェクトの存在従属関係に着目して処理した方が，人間の認知的負荷は少ない事になる．即ち，既存の業務処理のやり方をそのまま写し取ることが，本当にユーザビリティの優れた業務システムを開発することにつながるのだろうか．これも，今後の課題である．

参考文献

[1] 井田明男，金田重郎，熊谷聡志，藤本明莉，“整合性要件のための存在従属性に着目したオブジェクト識別子の構成”，電子情報通信学会，知能ソフトウェア研究会，2014 年 11 月

[2] 井田明男，金田重郎，“存在従属性によるオブジェクト識別子の設計”，情報システム学会，第 10 回全国大会・研究大会，2014 年 11 月

[3] 渡辺幸三，“販売管理システムで学ぶモデリング講座”，翔泳社，2008 年 5 月

[4] 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

[5] 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ー英語 7 文型と関数従属性に基づくクラス図の作成ー”，情報処理学会・第 128 回・情報システムと社会環境研究会，2014 年 6 月

[6] 金田重郎，井田明男，酒井孝真,“英語 7 文型と関数従属性に基づくクラス図の理解”，電子情報通信学会，知能ソフトウェア研究会，2014 年 3 月

[7] 上野晴樹，“知識工学入門 改訂 2 版”，オーム社，1989 年 3 月