情報処理学会論文誌 Vol.56 No.5 1340–1350 (May 2015)

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

井田 明男[1,a)] 金田 重郎[2] 熊谷 聡志[2] 藤本 明莉[1]

受付日 2014年8月16日, 採録日 2015年2月4日

概要：今日ビジネスアプリケーション開発の現場では空前のアジリティが求められている．また，システムはよりいっそうデータ指向的になってきている．そのため，アプリケーションの中核となるモデルとして構築されるドメインモデルは，1）問題領域を端的に記述し，2）ドメインの専門家と開発技術者との意思疎通を促進し，3）ドメインの論理要件を満たして，4）データベース設計のデータ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であることが要求される．しかしながら，オブジェクト指向の発想だけでは，このようなドメインモデルを構築することは難しく，かといってデータ項目主導型の正規化理論をそのまま持ち込むことは困難である．そこで，本稿では，存在従属性に着目したドメインモデルの構築手法と UML のクラス図をベースとした表記法を提案する．ホワイトボードに手書きできるくらい簡潔でありながらも識別子の関係が手に取るように理解できるモデルを目指す．存在従属性の概念は理解しやすく，ドメイン中のいたるところで見出されるため，提案手法により，第 4 正規形と同等以上の正規化レベルを持ったドメインモデルが自然に構築できる．提案手法がオブジェクト指向手法において正規化に似た役割を果たし，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．

キーワード：存在従属性，ドメインモデル，クラス図，ユビキタス言語，論理要件頑健性

Creating Domain Model Having Logical Requirement Robustness by Focusing on Existence Dependency —How to use Domain Class Diagram as a Ubiquitous Language

Akio Ida[1,a)] Shigeo Kaneda[2] Satoshi Kumatani[2] Akari Fujimoto[1]

Received: August 16, 2014, Accepted: February 4, 2015

Abstract: Record-breaking agility is demanded today in the scene of the business application development. In addition, the system becomes data-oriented. Therefore we need domain model as follows: 1) It describes the problem domain precisely. 2) It promotes the mutual understanding of an expert and the engineer. 3) It expresses the logic requirements of the domain. 4) It is just usable for a database design. However, it is difficult to build such domain model by only for an object-oriented idea. However also, it is difficult to simply bring in the data item oriented normalization theory to OOA. Therefore, in this paper, we propose a method for creating domain model having logical requirement robustness by focusing on existence dependency and a model notation based on UML class diagram. It is easy to understand the concept of existence dependency. Proposed method plays a similar role to the normalization in object-oriented approach. We hope that our proposed method helps the participating system-engineers and domain-expert in a requirements analysis session.

Keywords: existence dependency, domain model, class diagram, ubiquitous language, logical requirement robustness

1. はじめに

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 [1]．長期的な傾向としては，ビジネスの深層部でよりいっそう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 [2]．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 [3] 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 [3]．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（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 1 つの典型的なスタイルとなっている [4]．

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]．このような識別子の取り扱い方では，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 [1] 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（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，本稿では，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 [5] やこの概念をモデリングに導入する提案 [6] は過去にも存在するため，本稿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，その表記とあわせて提案する．

以下，2 章では，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 章では，本提案の手法を説明する．4 章では，存在従属性の UML クラス図での表記について議論する．5 章では，本提案手法を例題に適用した結果を議論する．6 章はまとめである．

2. 概念の定義と関連研究

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・エヴァンスのドメイン駆動設計」[3] 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．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

[1] 同志社大学
Doshisha University, Kyotanabe, Kyoto 610–0394, Japan
[2] 同志社大学大学院・理工学研究科
Graduate School of Science and Engineering, Doshisha University, Kyotanabe, Kyoto 610–0394, Japan
[a)] ideaworks@kcn.jp

[*1] 本稿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 [7]．
[*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 [8]．

表 1 DOA と OOA の大局的な対比
Table 1 Rough comparison of OOA and DOA.

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

語 [3] として機能することが求められるといえよう．

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 [9]．

OOA は，第 2 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 [10] 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筆者らは考える．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きわ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ごと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 1 は DOA と OOA の特徴を筆者らが大局的に対比したものである．

2.3 オブジェクト指向と正規化理論

表 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いまだ存在しない．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，前述のとおり 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 [1] 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．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 1 つとなっている．

2.4 存在従属性

存在従属性の概念は P. Chen が用いている [8]．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うるとき，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 [6]．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文献 [9] だけが存在従属性について言及している．

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

[*4] 弱実体とも呼ばれる [8].
[*5] 強実体とも呼ばれる [8].

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 [11]」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 3.3 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．次章では OOA の単純さ，直感的な分か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. 提案手法

3.1 ガイドライン

提案の核心はきわ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．

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」時点を表す時刻または版（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 [11]．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．各々の授業には，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けで，この問題は劇的に考えやすくなる．以下では，その手順とモデルの表記について説明する．

3.2 手順とモデルの表記

3.2.1 Step1：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．ここであるいは，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しれないが，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．

[*6] 佐藤 [12] 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．

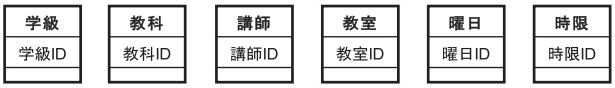

図 1 例題ドメインの独立クラス群
Fig. 1 Master classes for example domain.

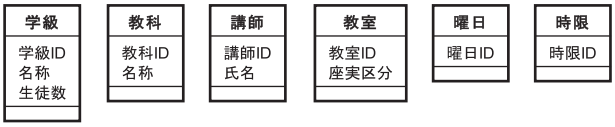

図 2 属性が追加された例題ドメインの独立クラス群
Fig. 2 Master classes with attributes for example domain.

図 1 は Step1 実施後のドメインモデルである．図 1 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．

3.2.2 Step2：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

次に，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）に従って，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どおり「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 2 は Step2 実施後のドメインモデルである．

3.2.3 Step3：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 は Step3 実施後の例題のドメインモデルである．図 3 では，前述のガイドライン（ハ）に従って，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 とを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合せに限定される．

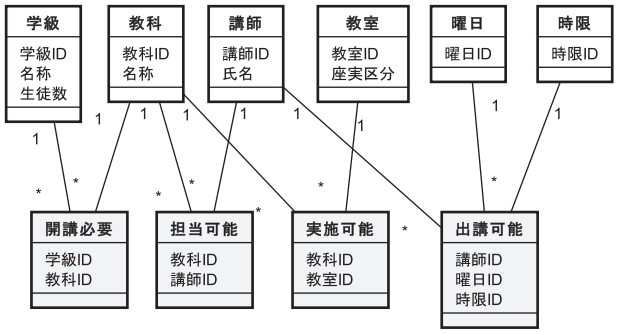

図 3 Step3 実施後の例題のドメインモデル
Fig. 3 Obtained domain model for example after Step3.

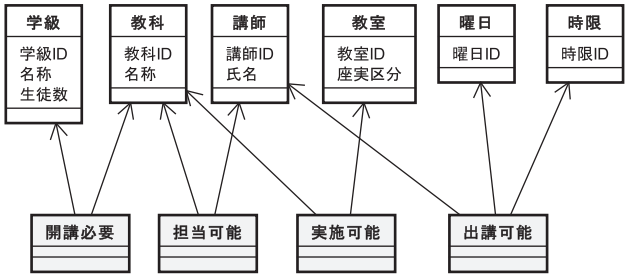

図 4 図 3 の提案記法による表記
Fig. 4 Domain model by proposed notation for Fig. 3.

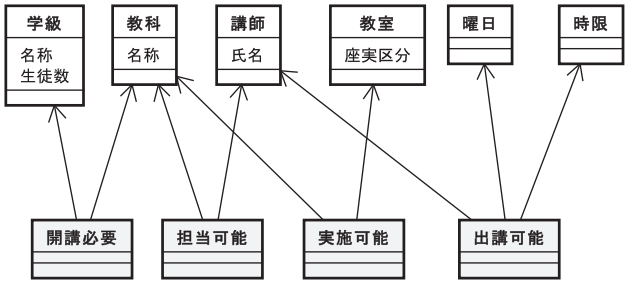

図 5 図 3 の提案記法による表記
Fig. 5 Another drawing by the proposed notation for Fig. 3.

ここで，表記法についても検討したい．図 3 では，従属クラスを淡い灰色で表記し，各々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，従属クラスから見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 のモデルは図 4 または図 5 のように表記できることもあわせて提案する．図 5 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

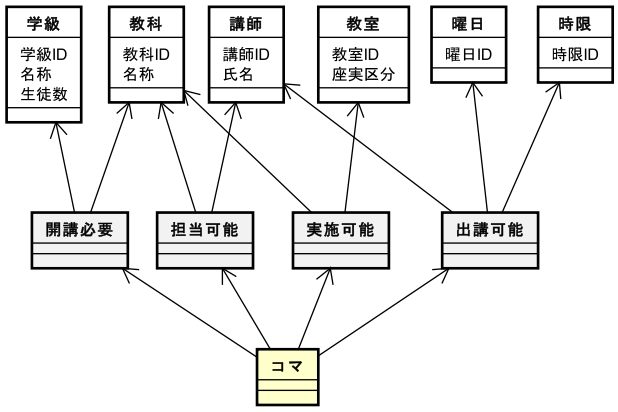

図 6 例題ドメインの Step4 実施後のモデル
Fig. 6 Obtained domain model for example after Step4.

可能であることを示唆している．

さて，図 4 と図 5 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，1 章で述べたとおり RDB への実装を前提としている以上，妥当である考える．ただし，この表記法の提案に至った経緯については 4 章で議論する．

3.2.4 Step4：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 6** 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 6 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うえ，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 3.3 節で論じる．

3.3 提案手法と正規化レベルの対応

表 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．正規化レベルごとの概要については渡辺 [13] を参照した．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 2.5 節で述べたとおり，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，各々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] 「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 2 提案手法の正規化レベルへの寄与
Table 2 Contribution to normalization level by proposed method.

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，結合従属性がある関係を 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はずであるが，分離されたおのおののクラスに結合従属性が残存しないことを保証するものではない

3.3.1 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，$a \to c$ が成立する場合はガイドラインによって，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．次に $\{a, c\} \to b$ 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．

3.3.2 提案手法と第 4 正規形

次に，第 4 正規形を検証する．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

[*9] 実際には，これ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する．

合，D の識別子は基本的に $\{a, b, c\}$ であり[*9]，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，$\{a, b, c\} \to d$ 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，3.3.1 項と同様であるため，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．

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．

3.3.3 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ものにならないと述べている [13] 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[*10]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．

4. 表記法についての議論

3 章では，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．この章ではその可能性について，若干の検討を加えておく[*11]．

4.1 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 7**）．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ふさわしい．

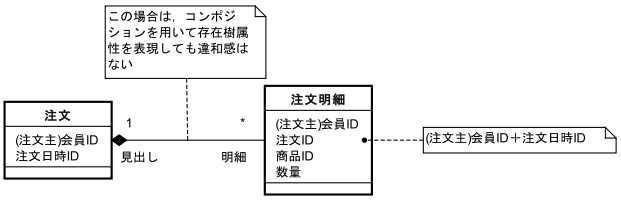

図 7 コンポジションを用いた存在従属性の表記（適切な例）
Fig. 7 Notation for existence dependency by composition (acceptable case).

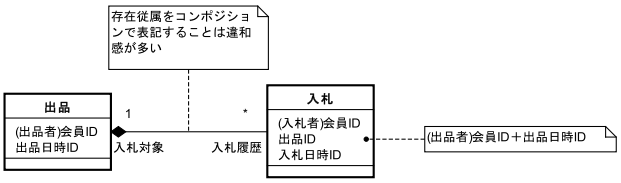

図 8 コンポジションを用いた存在従属性の表記（不適切な例）
Fig. 8 Notation for existence dependency by composition (unacceptable case).

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 8** 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 [14] 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．

4.2 存在従属性を関連クラスで表記する

次に，存在従属性を関連クラスで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 [14] 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なけれ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

[*10] 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．
[*11] 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．

をオブジェクト図で表現すれは，**図 9** のようになる．図 9 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 10** 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．

4.3 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，結論からいえば，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 11** 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付けられない．

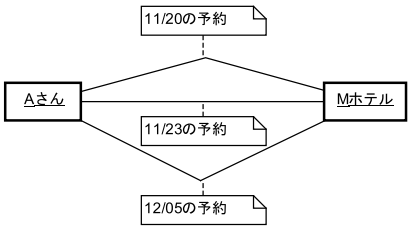

図 9 ホテル予約ドメインのオブジェクト図
Fig. 9 An object diagram for a hotel reservation domain.

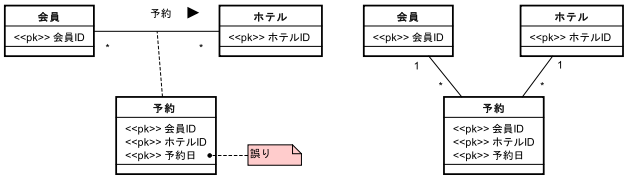

図 10 ホテル予約ドメインのクラス図
Fig. 10 Class diagrams for a hotel reservation domain.

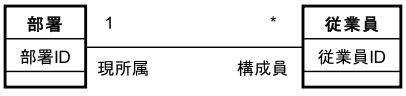

図 11 存在従属ではない関連の例
Fig. 11 Example for association not existence dependency.

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たどり着いた．

5. データモデルによる検証

前章までで，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うえ，その論理要件を満たす理想的な識別子について考慮も無意識に実施してしまえることが明らかとなった．

この章では，別の例題を用いて提案手法を検証する．

5.1 例題 2 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** は清掃当番表の例である．

5.2 ドメインモデルの構築と実験

例題 2 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．清掃当番表の割当てをバージョン管理するために，ガイドラインの（ニ）と，割当て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 12** は，成果物を UML 表記に忠実に表記したものである．また，**図 13** 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 13 のほうであると考えられる．

さらに，**図 14** は，図 13 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．

5.3 試行結果

試行の結果，このモデルは，次のあたりまえの業務要件

表 3 例題 2 の清掃当番表
Table 3 Cleaning rotation table for example 2.

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

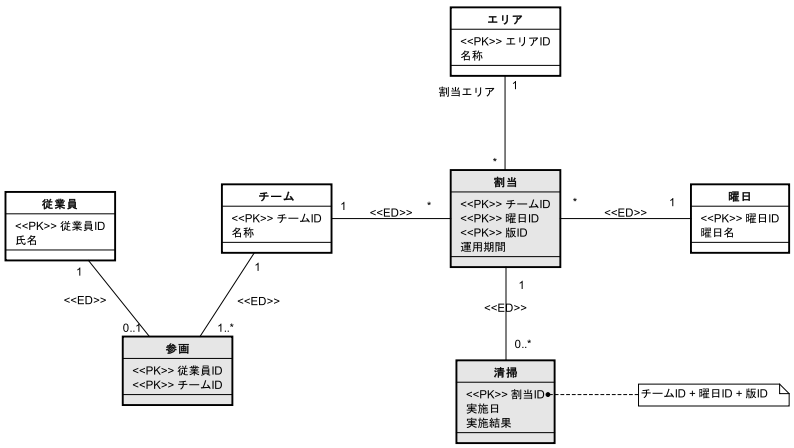

図 12 例題 2 のモデル（クラス図の表記法に忠実な表記）

Fig. 12 Domain model for example 2 by UML original notation.

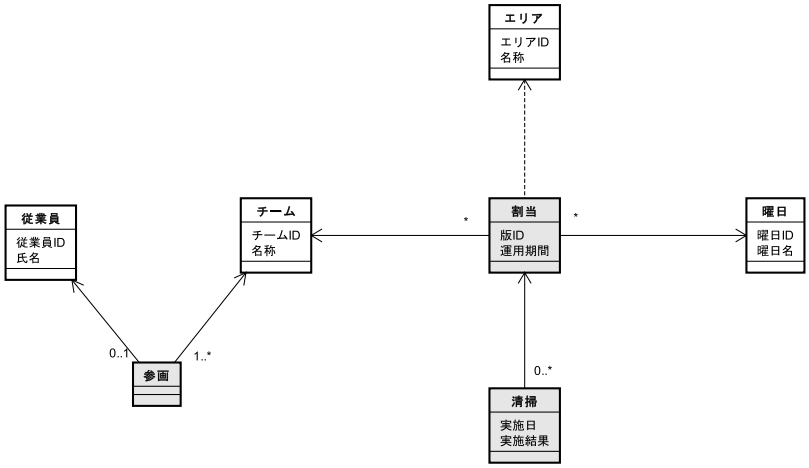

図 13 例題 2 のモデル（提案の表記法による表記）

Fig. 13 Domain model for example 2 by our proposed notation.

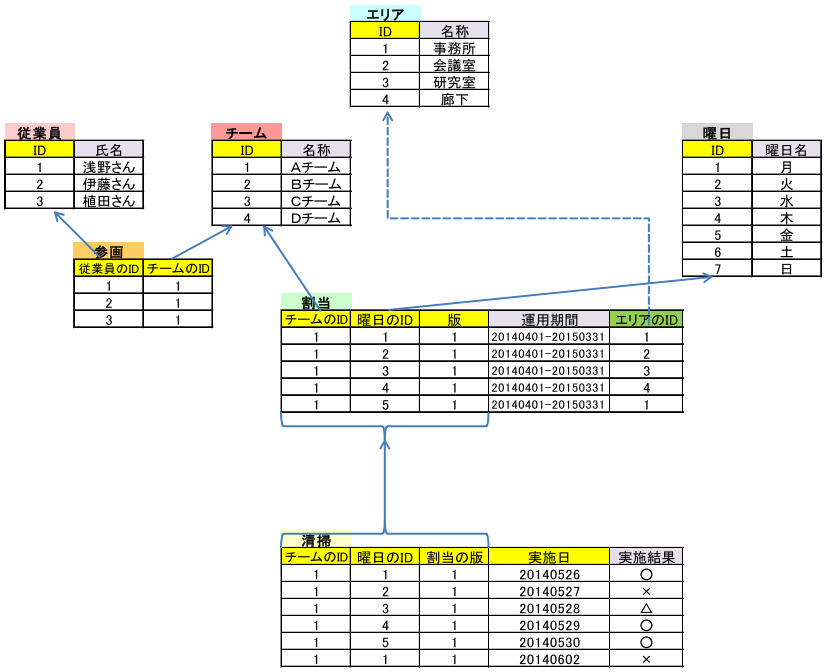

図 14 検証のために用いた例題 2 のデータモデル

Fig. 14 Data model used for verification for example 2.

を愚直にクリアした．

- 割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
- 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
- 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
- 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
- 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

6. まとめ

本稿では，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．Jacobsonら [15] 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．

筆者らは，かね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 [16], [17]．それらの概略は，1 つは，英語，すなわち，主語が目的語を動詞で制御する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 1 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多次元尺度構成法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．

参考文献

[1] 一般社団法人 ICT 経営パートナーズ協会：超高速開発が企業システムに革命を起こす，日経 BP 社 (2014).
[2] Ambler, S.，越智典子，オージス総研：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 出版センター (2005).
[3] エリック・エヴァンス，今関　剛：エリック・エヴァンスのドメイン駆動設計，翔泳社 (2011).
[4] 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).
[5] Snoeck, M. and Dedene, G.: Existence dependency: The key to semantic integrity between structural and behavioral aspects of object types, *IEEE Trans. Softw. Eng.*, Vol.24, No.4, pp.233–251 (1998).
[6] Haesen, R., Snoeck, M., Lemahieu, W. and Poelmans, S.: *Existence dependency-based domain modeling for improving stateless process enactment*, Hogeschool-Universiteit Brussel, Belgium (2009).
[7] 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).
[8] Chen, P.: The Entity Relationship Approach to logical Database Design, *QED Information Science*, Wellesley (1977).
[9] James Rumbaugh：オブジェクト指向方法論 OMT—モデル化と設計，トッパン (1992).
[10] Coad, P. and Yourdon, E.：オブジェクト指向分析—OOA 第 2 版，トッパン (1993).
[11] Date, C.J.: *Database In Depth*, Oreilly & Associates Inc. (2005).
[12] 佐藤正美：論理データベース論考，ソフト・リサーチ・センター (2000).
[13] 渡辺幸三：データモデリング入門，日本実業出版社 (2001).
[14] OMG: UML Superstructure Specification, v2.4.1, available from ⟨http://www.omg.org/spec/UML/2.4.1/⟩.
[15] Jacobson, I., Jonsson, P., Christerson, M. and Övergaard, G.：オブジェクト指向ソフトウェア工学 OOSE—use-case によるアプローチ，トッパン (1995).
[16] 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 (2014).
[17] 井田明男，金田重郎：オブジェクトへの責務配分のための多次元尺度構成法の適用—ソフトシステムズ方法論の成果物からの接近，電子情報通信学会，知能ソフトウェア研究会 (2014).

井田 明男　（正会員）

1959 年生．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，1989 年 4 月株式会社オージス総研入社．業務系および技術系のシステム開発やオブジェクト指向と UML を用いた開発のトレーニングやコンサルティングに従事．2012 年に独立．現在，有限会社井田代表取締役，同志社大学理工学部講師．要求モデリング，構造モデリング，コード自動生成の研究に従事．特種情報処理技術者．電子情報通信学会会員．

金田 重郎 （正会員）

1951 年生．1974 年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．同年 4 月日本電信電話公社・武蔵野電気通信研究所入所．大型汎用電子計算機の実用化，ならびに，誤り検出訂正符号の研究に従事．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．現在は，同理工学研究科・情報工学専攻教授．要求分析・センシング応用技術の研究に従事．工学博士（京都大学），技術士（情報処理部門）．IEEE 会員．

熊谷 聡志

1990 年生．2014 年 3 月同志社大学理工学部インテリジェント情報工学科卒業．2014 年 4 月同大学大学院理工学研究科情報工学専攻博士前期課程入学．要求モデリング手法の研究に従事．

藤本 明莉

1993 年生．2011 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．要求モデリング手法の研究に従事．