

概念モデルに基づく柔軟な O/R マッピングを実現するフレームワーク DBPowder の提案

村上 直^{1,2,a)} 天笠 俊之^{1,b)} 北川 博之^{1,c)}

概要:

現在、ウェブ、オブジェクト指向、関係データベースの一般化などに伴い、データ永続化に関する開発の効率化を目的とした O/R マッピング技術の重要性が増しており、O/R マッピングのためフレームワークが多数提案されている。これらは、1) オブジェクト指向言語で記述するクラスと関係データベース中のリレーションについて、単純な対応関係を前提とするもの、2) 複雑なクラス定義とリレーションの間の対応関係を記述できるもの、の二種類に大別される。前者は扱いやすい反面、複雑なオブジェクトを扱いづらい欠点がある。一方で後者は、複雑なクラス定義が最初から要求されるため、開発の初期段階でアプリケーションロジックの詳細が定まっていなかった場合に使いづらいという問題がある。本研究では、Extended Entity-Relationship (EER) モデルをベースとしたモデル記述と、有向グラフ併用によりアプリケーションロジックを反映できるクラス定義を組み合わせた、O/R マッピングフレームワーク DBPowder を提案する。DBPowder では、EER モデルでクラスとリレーションの単純な対応関係を扱い、有向グラフ併用によりアプリケーションに固有の性質を捉えるという方針をとることで、初期開発におけるコスト低減と、複雑なアプリケーションロジック対応の両立を図ることができた。

1. はじめに

ウェブ技術の普及、オブジェクト指向技術および関係データベース (RDB) の一般化、アプリケーション開発の一層の迅速化が求められる時勢などが相まって、ソフトウェア開発におけるデータ永続化のサポートと、それに関連する開発の効率化が重要になっている。今日の一般的なシステムにおいては、ソフトウェアそのものはオブジェクト指向言語で記述されることが多い一方、データの永続化には関係データベースを用いることが多い。このため、データ永続化のための関係スキーマ (RS) が設計され、それにしたがってリレーションが作成される。データ永続化の際、オブジェクト指向言語で記述したプログラムから RDB に対してデータの読み書きが行われる。この際に必ず、オブジェクトの状態を SQL に変換する処理や RDB からの返値をオブジェクトに変換する処理が発生する。永続化対象データの種類が増えるにつれ、この変換に関連する開発および保守のコストは高くなる。

O/R マッピング (ORM) フレームワークは、このような問題に対応するために提案され、広く用いられている。その目的は大きく以下に列挙する 3 点の開発の効率化にある。

- 永続化クラス (PC^{*1}): オブジェクト指向言語上で永続化データを扱うためのクラス
- 関係スキーマ (RS): 永続化データを RDB 上で管理するために用いる
- PC で扱うデータを RDB 上で永続化するために必要な処理

ORM フレームワークに求められる要件は様々であるが、本研究では以下に挙げる二つの要件に着目する。

- 近年ますます迅速化が求められるアプリケーション開発において、初期開発におけるデータ永続化の設計や開発のコストの低減
- 複雑なアプリケーションロジックに対応可能な PC の開発

従来の ORM フレームワークは、以下の二通りに大別される。

- (1) クラスとリレーションについて、単純な対応関係を前提とするもの ([10] [7] [13] [21])

¹ 筑波大学大学院システム情報工学研究科 コンピュータサイエンス専攻

² 高エネルギー加速器研究機構 計算科学センター

a) tadashi@kde.cs.tsukuba.ac.jp

b) amagasa@cs.tsukuba.ac.jp

c) kitagawa@cs.tsukuba.ac.jp

*1 PC は persistent class の略

(2) 複雑なクラス定義とリレーションの間の対応関係を記述できるもの ([12] [1] [5])

(1) の場合、比較的単純なマッピングはうまく扱えるものの、リレーションとクラスの単純な対応しか扱うことができない。このため、アプリケーション側の複雑なクラス構成を反映したような複雑なマッピングはうまく扱えない。一方で(2)の場合、複雑なマッピングもうまく扱えるものの、関係スキーマとクラス定義の両方が定まっている必要がある。

近年のアプリケーション開発の現場では、開発スピードの向上が著しく、アジャイル開発 [14] などが注目を集めている。このような開発においては、従来型のウォーターフォールモデル [18] のような、開発初期からの詳細な設計をせず、比較的小機能なプロトタイプの実装とデプロイ、機能の拡張をスパイラル的に繰り返すことによって、より大規模なシステムを構築する。この場合、開発初期から詳細な設計は得られないので、(2) のカテゴリの ORM フレームワークは利用しづらい。一方で開発が進むと、アプリケーション側のコードが複雑化するため、(1) のカテゴリが扱う単純なマッピングでは、アプリケーション側の複雑な要求に十分対応することができない。

本研究では上記の問題の克服を目的として、概念モデルに基づく柔軟な ORM を実現するフレームワーク DBPowder を提案する。DBPowder で提案する開発スタイルは以下の通りである。

- 初期開発では、プログラマは EER モデル [19] に基づく概念モデルを記述する。DBPowder は記述内容に基づいて RS と PC を生成する。ここで記述する概念モデルは、実体、非キー属性、実体間の関連のみの簡素な構成とする。
- アプリケーションに適した PC の開発に資するために、DBPowder では ObjectView という仕組みを導入する。プログラマは ObjectView を記述することで、PC のデータ構造を柔軟に記述できる

DBPowder では、EER モデルでクラスとリレーションの単純な対応関係を扱うことで初期開発における設計や開発のコスト低減を可能とし、有向グラフの併用により利用アプリケーションに固有の性質を捉えることで複雑なアプリケーションロジックに対応することを可能にする。

本論文の構成は以下の通りである。2 節で関連研究について述べ、3 節で本研究の対象とするオブジェクトモデルと関係モデルについて述べる。4 節で DBPowder を提案し、5 節でその内容について議論する。6 節でまとめを述べる。

2. 関連研究

既存の ORM フレームワークは、永続化クラスとリレー

ションの単純な対応関係を前提とするもの(2.1 節)と、複雑なクラス定義とリレーションの間の対応関係を記述できるもの(2.2 節)に大別できる。本節ではまずそれらについて述べ、次に、ORM に関連する技術として、関係モデルにおけるビューを取り上げる(2.3 節)。

2.1 永続化クラスとリレーションについて、単純な対応関係を前提とするもの

ActiveRecord [10] は、本カテゴリの代表的アプローチのひとつである。本アプローチでは、ActiveRecord クラスが PC を担う。ActiveRecord クラス内で対となる RS 上のリレーション名を指定すると、指定したリレーションの保持する属性名がそのまま ActiveRecord クラスの属性名として定義される。このように、簡潔な記述で ORM を実現できることが、本アプローチの特徴である。本アプローチは、プログラム実行時にクラスの型を変更できる動的プログラミング言語との相性が良い。特に Ruby on Rails (RoR) [11] における実装では、スキーマ設計時の規約を多く設ける*2 ことで必要な記述量を削減する Convention over Configuration (CoC) [9] [17] という考え方を併せて導入することで、初期開発時の簡素化に大いに貢献している。しかし CoC が定める規約から逸脱した開発が必要になると、その生産性は大きく下がる。例えば、リレーションとクラスの対応が単純な 1 対 1 で収まらない場合や、複合キーが必要となる場合などが挙げられる。

別のアプローチとして、プログラマが ER モデル [8] などの概念モデルを設計し、フレームワークが概念モデルに基づいて RS、PC および永続化に必要な処理コードを生成する手法がある。代表例として WebML [7] [3] や Enterprise Objects Framework [13] を挙げることができる。いずれもプログラマは概念モデル上で関連、継承、参加制約を記述することができ、その上でフレームワークが記述内容を RS と PC へ適切に変換する。しかし、フレームワークの範疇ではアプリケーションの利用状況に応じて複数の PC を取ることはできない。

別のアプローチとして、Thiran らはラッパー指向の ORM を提案している [21] [20]。提案では、RS を出発点として、フレームワークが定めた 8 種類のスキーマ変換ルールをプログラマが適宜適用することで、プログラマが望む PC を取得できるとしている。PC を RS 上で永続化するために必要な処理は、適用した変換ルールに沿うことで定義できる。RS と PC の構造上の違いに着目できるのは利点だが、変換ルールに基づく変換のみしかサポートしない。

*2 クラス名は英単語の単数形、リレーション名はクラス名の複数形、主キーは代理キーとして id という名前にする、など

2.2 複雑なクラス定義とリレーションの間の対応関係を記述できるもの

Hibernate [12] は、本カテゴリの代表的アプローチのひとつである。本アプローチでは、.hbm という xml ファイル上に PC を記述するとともに、記述した PC の各要素ごとに RS の要素との対応を記述するという方式を基本とする。 .hbm ファイルをツールに解釈させると、RS が構築されるとともに、Java の永続化クラスに対応するソースコードが生成される。 Hibernate では PC と RS と要素の対応を全て記述する必要があるため、初期開発の迅速化は望めないが、その一方で PC と RS の複雑な対応関係を記述できる。 なお Hibernate annotations を用いると、.hbm ファイルと同内容を Java の永続化クラス上にアノテーションで記述できる上に、RoR の ActiveRecord と類似の CoC も備える。 しかし Java の言語仕様上、Java で永続化クラスを記述することそのものが初期開発の迅速化のためには足枷となる。 また Hibernate や Hibernate annotations では、実行時モデルの制約から、RS と PC の属性は 1 対 1 に対応することが前提となっており、フレームワークの範疇ではアプリケーションの利用状況に応じて複数の PC を取ることはできない。

別のアプローチとして、Microsoft ADO.NET Entity Data Model (EDM) [2] [1] を挙げることができる。 EDM では、概念スキーマ定義言語 (CSDL)、ストアスキーマ定義言語 (SSDL)、およびマッピング仕様言語 (MSL) という XML ベースの 3 種類の言語を用いて ORM を実現する。 3 種類の言語を扱うことでアプリケーションに適した PC 開発が可能だが、初期開発の迅速化は困難である。 初期開発の迅速化のためには、3 種類の言語を同時に扱うための GUI の援用が必須となる。 なお MSL においては、関係する PC と RS 間の制約を SQL の拡張言語を用いて列挙することで ORM を実現するアプローチが提案されている [15]。 このアプローチでは、列挙した制約文はコンパイルされて Query View と Update View に分解され、それぞれがデータ参照と更新を担う。 このアプローチにより、非常に複雑な RS と PC の対応関係を、制約の列挙により記述することができる。

このほか、RS と PC が別途与えられていることを前提として、その対応関係の記述を効率化しようとするアプローチとして、M²ORM² [5] [4] が提案されている。

2.3 関係モデルにおけるビュー

関係モデルにおけるビューを用いると、リレーションをアプリケーションに適した構造に再構成した別の仮想的なリレーションを、RS 上に作成できる。 しかし、生成した仮想的なリレーションと PC の対応づけは別途必要である。 また、作成したビューへの値の挿入、更新、削除について、定義元のリレーションに対する操作を一意に定めら

れない場合や、副作用を引き起こす場合があることが知られている (ビュー更新問題)。 また、ビューを用いてリレーションを再構成すれば RS と PC の複雑なマッピングを実現できる可能性があるが、複雑なマッピングを実現する際のビュー展開時に、サブクエリを含むなど非効率な SQL しか生成できない恐れがある。

3. 準備: 対象とするデータモデル

本節では、本研究が対象とするオブジェクトモデルと関係モデルについて述べる。

3.1 対象とするオブジェクトモデル (OM)

DBPowder が扱うオブジェクトモデル (OM) は、ODMG3.0 [6] のサブセットを基本とし、Java を参考にした拡張を含む。

OM は、オブジェクトとリテラルで構成される。 オブジェクトは識別子を持ち、リテラルは識別子を持たない。 オブジェクトは状態とふるまいをもつ。 状態はプロパティの集合により定義され、ふるまいは、操作の集合により定義される。 プロパティは、属性または関連 (後述) である。

型は、オブジェクトやリテラルを分類する。 オブジェクト型は、インタフェース定義とクラス定義により構成される。 リテラル型は、Java のものに準じる。 オブジェクト型のふるまいを定義した仕様をインタフェース定義と呼び、オブジェクト型の状態とふるまいを定義した仕様をクラス定義と呼ぶ。

次に、具象クラスと抽象クラスを以下のように定める。 仕様を実装済のクラスを具象クラスと呼ぶ。 記述された具象クラスにおいて、状態は属性や関連として全て実装されており、ふるまいはメソッドとして全て実装されている。 仕様の全てまたは一部が実装済ではないクラスを、抽象クラスと呼ぶ。 抽象クラスでは、クラス定義を構成する属性、関連、メソッドのいずれかについて、実装されていないものがある。 以降、断り無くクラスと呼ぶときには、具象クラスを指すものとする。

型 A , B があるとき、型 A がもつ状態やふるまいを型 B が全て備え、型 B のオブジェクトの存在が同時に型 A のオブジェクトの存在を意味するとき、「型 B は型 A を継承する」という。 クラスは、一つのクラスと一つ以上のインタフェースを継承できるが、複数クラスを多重継承することはできない。 インタフェースは、一つ以上のインタフェースを継承できるが、クラスを継承することはできない。

二つの型について、相方のオブジェクトを参照できる関係のことを、関連と呼ぶ。 相互に参照できる関連を双方向関連とよび、片方向のみに参照できる関連を、片方向関連と呼ぶ。 片方向関連は、型 A , B について、 $A \rightarrow B$ の参照は可能だが、 $B \rightarrow A$ の参照はできないといったものである。 関連により参照できるオブジェクトの個数に応じて、

1 または多の多重度を定義する．双方向関連の場合の多重度は，1:1, 1:n, n:1, n:m のいずれかとなる．

関連プロパティの値として null を許すかどうかで，参加制約を定義する．

オブジェクトのプロパティを保存することで，オブジェクトを別のプロセスや計算機から復元可能にする操作のことを，永続化と呼ぶ．

3.2 対象とする関係モデル (RM)

本節では，DBPowder が扱う関係モデル (RM) について述べる．

属性 A_1, A_2, \dots, A_n に対して，各々の属性のドメインの直積 $D_1 \times D_2 \times \dots \times D_n$ の有限部分集合を，リレーション T とする．リレーションの各要素を，タプルと呼ぶ．

次に，候補キー，主キー，外部キーを以下のように定める．リレーション中のタプルを一意に識別できる属性の部分集合で，要素数が極小のものを，候補キーと呼ぶ．候補キーからある一つを選択したものを，主キーと呼ぶ．主キーにおいては，各属性値は空値 (null) を許さない．リレーション $T_1(\dots, A_{FK}, \dots)$ と $T_2(A_{PK}, \dots)$ について，任意の A_{FK} の値が，null または A_{PK} 上の ある値に一致するとき， A_{FK} を外部キーと呼ぶ．ここで特に $T_1 = R_2$ の場合，自己参照と呼ぶ．

DBPowder が扱う関係代数演算は，射影，選択，自然結合 (等結合) である．データ定義言語およびデータ操作言語としては，SQL を用いる．

4. 提案: DBPowder O/R マッピングフレームワーク

4.1 概要

本研究では，DBPowder O/R マッピングフレームワークを提案する．DBPowder は，初期開発の簡素化とアプリケーションに適した PC 開発の両方を実現するために，EER モデルと有向グラフをフレームワークの中核に据える．

初期開発では，プログラマは EER モデルに基づく概念モデルを記述する．DBPowder は記述内容に基づいて RS と PC を生成する．ここで記述する概念モデルは，実体，非キー属性，実体間の関連のみの簡素な構成とする．

アプリケーションに適した PC の開発に資するために，DBPowder ではアプリケーションが必要とする実体と関連を有向グラフに見立てた，ObjectView という仕組みを導入する．プログラマは ObjectView を記述することで，EER モデルを再構成した PC の構築を容易に行うことが可能となり，PC のデータ構造を柔軟に記述できる．

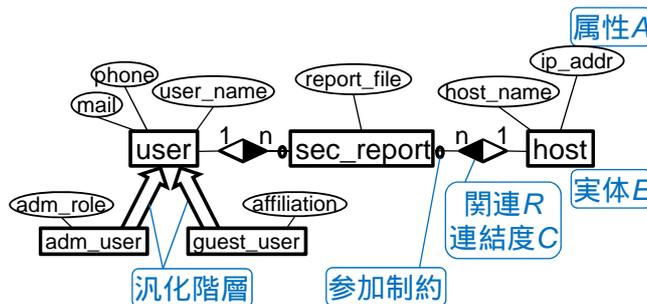


図 1 EER モデルの例

4.2 DBPowder における，EER モデルによる概念モデル記述

本節では，DBPowder が扱う EER モデルについて述べる．図 1 に，EER モデルを図示した例を示す．

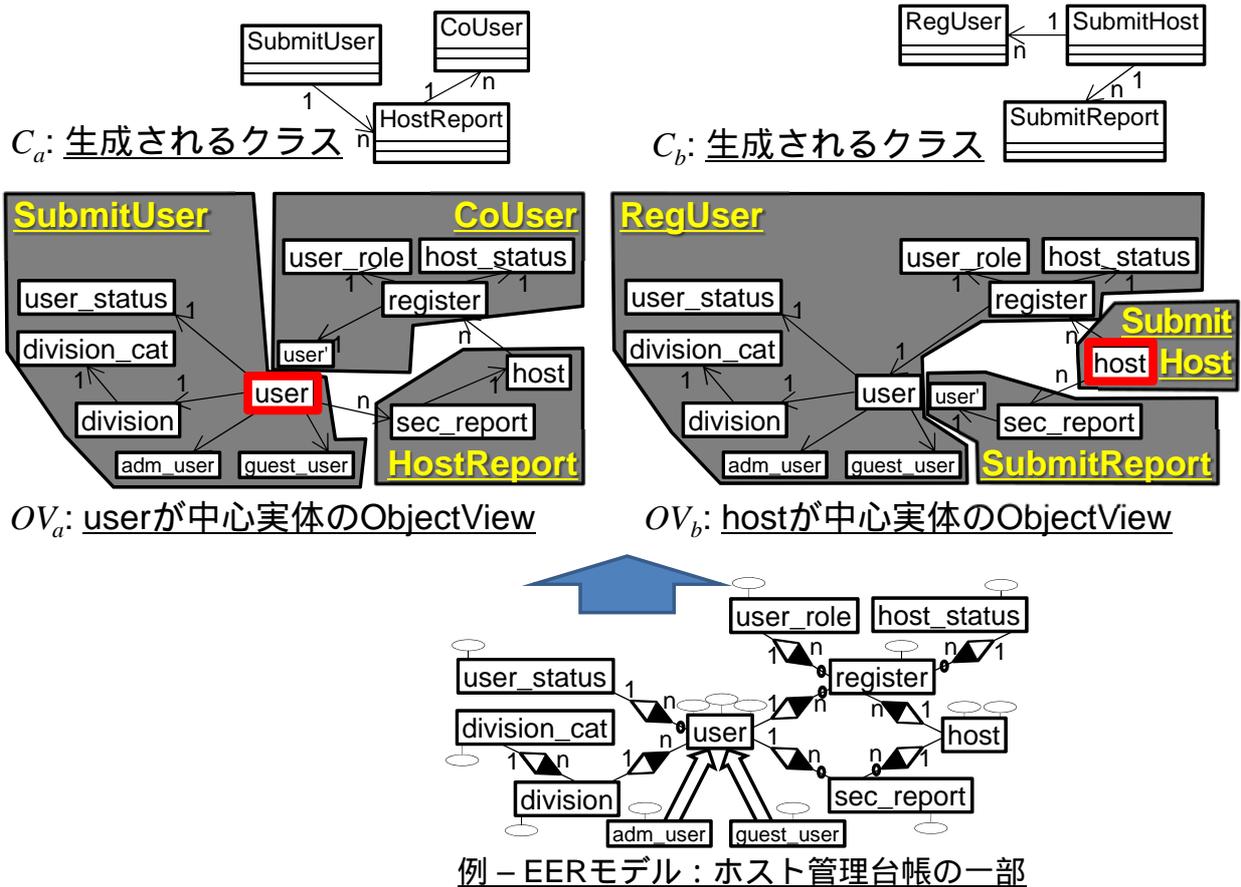
DBPowder で扱う EER モデルでは，属性 A ，実体 E ，関連 R ，連結度 C の四つを基本的な要素とする．定義は以下による．まず，データモデルが対象とするそれぞれのことがらを e とする． e は一つ以上の性質をもち，そのそれぞれを属性 A_1, A_2, \dots, A_n と呼ぶ．この属性について，集合 $\{A_1, A_2, \dots, A_n\}$ が同一である e を集めた集合が，実体 E である．ここで， e を実体の要素と呼ぶ．二つの実体 E_i, E_j について，要素間の連結度 C を定義できるとき，実体 E_i, E_j は関連 R_{ij} を持つとする．ここで連結度 C は，1:1, 1:n, n:m のいずれかとする．三つ以上の実体 E_1, E_2, E_3, \dots に関する関連は，関連を扱うための実体 E' を別途導入し， E' に対する各々の実体 E_1, E_2, E_3, \dots との関連を扱うことで記述する．

次に，候補キーと主キーを定める．候補キーは， E 中で e を一意に識別できる属性の集合である．主キーは，候補キーのうち，モデルで決定したあるひとつのキーである．なお DBPowder では，主キーの記述を省略可能とする．省略した場合は，DBPowder が主キーを割り当てる

DBPowder で扱う EER モデルでは，汎化階層と参加制約を扱うことができる．ある実体 A の属性をすべて継承した実体 B について， B における要素の存在が同時に A における要素の存在を意味するとき， A と B は汎化階層の関係にあるという．このとき， B の主キーは A によって定義される．次に，実体 E, E_r があり，それらが関連をもつとする． E_r の全ての要素が E のある要素と関連をもつ必要がある場合， E_r は E に対して全面的であるという．もつ必要がない場合，部分的という．全面的/部分的を決める制約を，参加制約という．

4.3 ObjectView: EER モデル上のオブジェクト表現

4.2 節で述べた EER モデルの実体 E と関連 R のうち，アプリケーションが必要とする要素を抽出した連結グラフを ObjectView と定義する．以下では，ObjectView を形式



例 - EERモデル：ホスト管理台帳の一部
図 2 ObjectView と生成されるクラス

的に定義する．まず， $G = (E_v, R_v)$ について考える．ここで E_v は節点， R_v は枝である． G に以下の制約を定める．

- G は必ず一つの始点を持つ．この始点に対応する実体を中心実体と呼ぶ
 - 枝 R_v は方向を持つ．つまり G は有向グラフである．
- このように定めた有向グラフ G について， E_v と R_v にアプリケーションが必要とする性質を付与したものを，ObjectView と呼ぶ．なお，EER モデル上の E や R のそれぞれについて，対応する E_v や R_v の個数は任意とする．また，枝 R_v に対応する関連が属性を持つ場合は，その属性が所属する実体を，連結度が多の側と定める．

ObjectView で指定した関連について，関連に対応する枝の終点に対応する連結度が 1 の場合は，その枝に連結する二つの節点をグループ化できる．このルールは再帰的に適用することができる．

図 2 に例を示す．下段は，[22] で示したアプリケーションから，ホスト管理台帳部分の一部を抜粋した EER モデルである．これについて，たとえば host を中心実体とする ObjectView と user を中心実体とする ObjectView を定めることができる．この例をそれぞれ，中段の OV_a と OV_b に示し，各々が生成する PC を，それぞれ上段の C_a と C_b に示す．

次に，図 2 の OV_a の例について詳細を見る．特に，Ob-

jectView において実体がグループ化されていることの意味と，実体 user が二つ存在することの意味について考える．

OV_a は中心実体を user とし，利用者のセキュリティレポート提出に関する永続性を記述した ObjectView である．EER モデルでは九つのクラスと二つのサブクラスが存在するが，節点のグループ化を再帰的に適用した結果，ObjectView が生成するクラス C_a では，クラス数は三つとなっている．たとえば SubmitUser グループ内の実体 division, division_cat, user_status について，利用者の所属組織や利用者の状態を管理するために user と別の実体であることが必要だが，セキュリティレポート提出という局面に限定するならば利用者の属性の一部と考えることができる．また実体 user について，ObjectView では SubmitUser と CoUser の 2 箇所から参照されている．SubmitUser はレポートを提出した利用者，CoUser はレポート対象ホストの共同管理者であるため，SubmitUser と CoUser は同一人物ではない．従って，実体 user に対して user' という別名をつけて ObjectView を構成している． C_a はこのような状況に適したクラス構成といえる．つまり，ObjectView が実体を効果的にグループ化しており，また user を二つ存在させることが効果的に作用しているといえる．なお ObjectView では，節点のグループ化が不適当な場合は，グループ化を適用しないことも可能である．

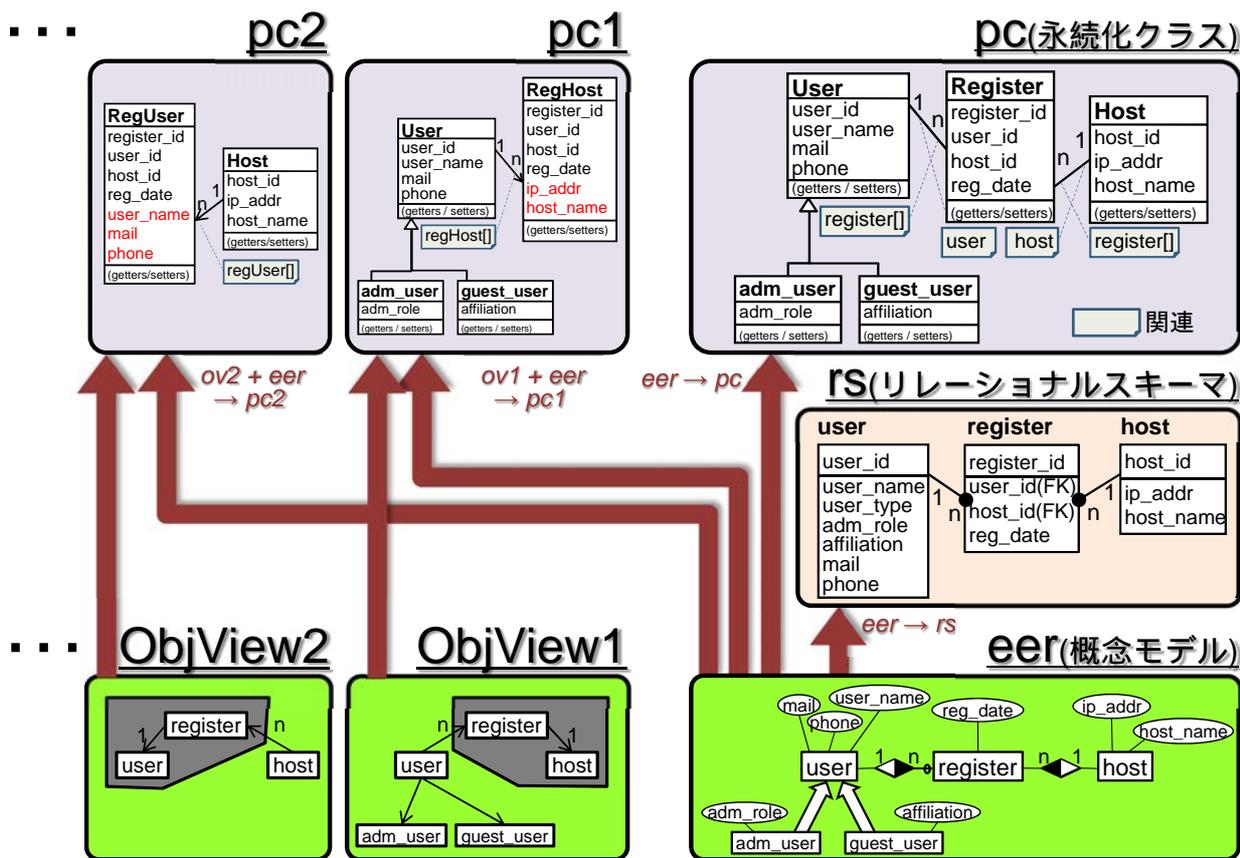


図 3 DBPowder の O/R マッピング

4.4 EER モデルと ObjectView を用いた O/R マッピング (ORM) の流れ

本節では、4.2 節と 4.3 節で述べた EER モデルと ObjectView を用いた O/R マッピングについて述べる。図 3 にその流れを示す。

はじめにプログラマは、EER モデルによるデータスキーマを記述する（記述したスキーマを *eер* とする）。次に DBPowder は、RS と PC を以下に示す手順により生成する。なお ObjectView は、RS の生成過程では使用しない。

- (1) DBPowder は、*eер* で主キーを省略した実体について主キー属性を付与する
- (2) DBPowder は、[19] に示される手法に基づいて *eер* から RS を生成する（生成結果を *rs* とする）
 - a) *eер* の実体及び属性に対応して、*rs* のリレーション及び属性を生成する
 - b) *eер* の関連については、連結度に応じて片方の主キーをもう片方に加え、加えたキーを外部キーとする
 - c) *eер* の汎化階層については後述する
- (3) DBPowder は、*eер* から PC を生成する（生成結果を *pc* とする）

- a) クラスと属性については、(2) に示した *rs* のリレーション及び属性の生成方法と同様にして、生成する
 - b) 関連については、(2) に示した *rs* の外部キーと同様の方法により片方向関連を生成し、同時に逆方向の関連を生成することにより、双方向関連を生成する
 - c) 関連の多重度は、*eер* で記述したものを採用する
- (4) DBPowder は、*eер* と ObjectView からアプリケーションに対応した PC を生成する（生成結果を *pc_n* とする）。手順は以下の通りである。
- a) 4.3 節で述べた方法により、クラス、関連および多重度を導出する
 - b) 属性については、ObjectView においてグループ化の対象となった節点に対応する *pc* 上のクラスを考え、このそれぞれのクラスが保持する属性をすべて、ObjectView 上のグループに対応するクラスがもつ属性とする

ObjectView 上のクラス、属性、関連、および多重度は、*pc* 上に対応する要素が存在する。この対応する要素を利用して、ObjectView と *rs* の対応づけを定義できる。

くる。PC とリレーションの単純な対応関係を前提とする ORM フレームワークでは、この局面に対応することができない。DBPowder では、ObjectView を設計すれば複雑な PC を構築することができる。図 3 では ObjView1 や ObjView2 がそれに該当する。図 3 のような単純な例であれば、もとの *pc* のみでアプリケーションロジックを扱ってもメリットはそれほど大きくないが、図 2 のような複雑なアプリケーションロジックを扱うようになると、アプリケーション利用の一局面に焦点を当てたクラス生成が威力を発揮する。

ObjectView は、設計済の概念モデルの資産を最大限活用できる。アプリケーションロジックに対応した有向グラフを記述することで、それぞれのロジックに適したクラスを容易に生成できる。複雑なクラス定義とリレーションの間の対応関係を記述できるフレームワークでは、各々のロジック全てについて、複雑なクラス定義を都度記述する必要がある。ObjectView は、この開発コストも大幅に削減できる。

6. まとめと今後の課題

本研究では、EER モデルと有向グラフを用いることで柔軟な O/R マッピングを実現するフレームワーク DBPowder を提案した。DBPowder では、EER モデルでクラスとリレーションの単純な対応関係を扱うことで初期開発における設計や開発のコスト低減を可能とし、有向グラフの併用により利用アプリケーションに固有の性質を捉えることで複雑なアプリケーションロジックに対応することを可能にした。提案者は [16] [22] にて、DBPowder フレームワークを実現するための言語 (DBPowder-mdl) とプロトタイプ実装、および実稼働例を提示している。本報告での考察を踏まえて DBPowder-mdl へのフィードバックを行い、再評価を実施する予定である。

参考文献

- [1] Adya, A., Blakeley, J. A., Melnik, S. and Muralidhar, S.: Anatomy of the ADO.NET entity framework, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, ACM, pp. 877-888 (2007).
- [2] Blakeley, J. A., Campbell, D., Muralidhar, S. and Nori, A.: The ADO.NET entity framework: making the conceptual level real, *SIGMOD Record*, Vol. 35, No. 4, pp. 32-39 (2006).
- [3] Brambilla, M., Comai, S., Fraternali, P. and Matera, M.: *Designing Web Applications with Webml and Webratio* (2007).
- [4] Cabibbo, L.: Objects Meet Relations: On the Transparent Management of Persistent Objects, *CAiSE, Lecture Notes in Computer Science*, Vol. 3084, Springer, pp. 429-445 (2004).
- [5] Cabibbo, L. and Carosi, A.: Managing Inheritance Hierarchies in Object/Relational Mapping Tools, *CAiSE, Lecture Notes in Computer Science*, Vol. 3520, Springer, pp. 135-150 (2005).
- [6] Cattell, R. G. G. and Barry, D. K.: *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann (2000).
- [7] Ceri, S., Brambilla, M. and Fraternali, P.: The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications, *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, Lecture Notes in Computer Science, Vol. 5600, Springer, pp. 273-292 (2009).
- [8] Chen, P. P.-S.: The entity-relationship model toward a unified view of data, *ACM Trans. Database Syst.*, Vol. 1, No. 1, pp. 9-36 (1976).
- [9] Edd Dumbill: Ruby on Rails: An Interview with David Heinemeier Hansson, <http://www.oreillynet.com/pub/a/network/2005/08/30/ruby%2Drails%2Ddavid%2Dheinemeier%2Dhansson.html> (2005).
- [10] Fowler, M.: *Patterns of Enterprise Application Architecture*, Addison-Wesley, Boston, MA, USA (2002).
- [11] Heinemeier, D. and et al: Ruby on Rails, <http://www.rubyonrails.org/> (2010).
- [12] JBoss Inc.: Hibernate, <http://www.hibernate.org/> (2007).
- [13] Kleissner, C.: Enterprise Objects Framework, A Second Generation Object-Relational Enabler, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, ACM Press, pp. 455-459 (1995).
- [14] Martin, R. C.: *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall PTR, Upper Saddle River, NJ, USA (2003).
- [15] Melnik, S., Adya, A. and Bernstein, P. A.: Compiling mappings to bridge applications and databases, *ACM Trans. Database Syst.*, Vol. 33, No. 4 (2008).
- [16] Murakami, T.: DBPowder-mdl: Mapping Description Language between Applications and Databases, *7th IEEE/ACIS International Conference on Computer and Information Science, IEEE/ACIS ICIS 2008, 14-16 May 2008, Portland, Oregon, USA*, IEEE Computer Society, pp. 127-132 (2008).
- [17] Nicholas Chen: Convention over Configuration, <http://softwareengineering.vazexqi.com/files/pattern.html> (2006).
- [18] Royce, W.: Managing the development of large software systems, *proceedings of IEEE WESCON*, Vol. 26, No. 8, Los Angeles (1970).
- [19] Teorey, T. J., Yang, D. and Fry, J. P.: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Surveys*, Vol. 18, No. 2, pp. 197-222 (1986).
- [20] Thiran, P., Hainaut, J.-L. and Houben, G.-J.: Database Wrappers Development: Towards Automatic Generation, *9th European Conference on Software Maintenance and Reengineering (CSMR 2005), 21-23 March 2005, Manchester, UK, Proceedings*, IEEE Computer Society, pp. 207-216 (2005).
- [21] Thiran, P., Hainaut, J.-L., Houben, G.-J. and Benslimane, D.: Wrapper-based evolution of legacy information systems, *ACM Transactions on Software Engineering Methodology*, Vol. 15, No. 4, pp. 329-359 (2006).
- [22] 村上 直: DBPowder-mdl: EoD と記述力を兼ねた O/R マッピング言語, *情報処理学会論文誌データベース (TOD)*, Vol. 3, No. 3, pp. 46-67 (2010).