概念モデリングに基づく O/R マッピング手法に関する研究

A Study on an O/R Mapping Method based on Conceptual Modeling

村上 直 ♥

Tadashi MURAKAMI

1. はじめに

オブジェクト関係マッピング（ORM）フレームワークは，データベースの利用が不可欠なアプリケーション（DB アプリケーション）の開発において，永続化対象となるメモリ上のデータをオブジェクト指向言語で扱うための（a）永続化クラス，これを関係データベース（RDB）上で永続化するための（b）関係スキーマ，および，永続化クラスから RDB へクエリし応答を受けとる（c）RDB アクセスコードの三点について，効率的な設計や開発に貢献する．ORM フレームワークは，永続化クラスとこれを扱う関係テーブルの対応関係に着目することで，二つのグループに大別できる．(1) 一対一を基本とした単純な対応関係（以下，単純な対応関係）を簡易に実現できるフレームワーク [6, 5, 7] ，(2) 単純ではない対応関係（以下，複雑な対応関係）を実現できるフレームワーク [2, 1].

DB アプリケーションは，小規模なプロトタイプの構築と機能拡張を繰り返す開発方式がとられることが多い．このとき（1）のフレームワークでは，開発が進み永続化クラスと関係テーブルの対応関係が複雑になった場合の対応が難しい．一方で（2）のフレームワークでは，単純な対応関係を簡易に記述できない上に，機能拡張の繰り返しに伴ってこの対応関係を度々修正する必要が生じる．このように ORM フレームワークでは，(1) と (2) のサポートの両立と，単純な対応関係から複雑な対応関係に移行する開発の効率的なサポートが望ましいが，既存の ORM フレームワークではこれは容易ではない．

このような問題の克服のために，本研究では ORM フレームワーク DBPowder を提案する．DBPowder では，単純な対応関係の簡易な実現が要求される開発初期には EER モデル [9] による概念モデリングにより開発を進め，複雑な対応関係の実現が要求される際には EER モデル上の実体の走査経路と利用方法を指定した有向グラフ ObjectView を追加する．提案手法の特徴は，EER モデルと ObjectView の連携により，(1) と (2) のサポートを両立し，単純な対応関係から複雑な対応関係に移行する開発を効率的にサポートできる点にある．また本手法は，概念モデル上の実体や関連を追加的あるいは修正的に指定して ObjectView を構成して，複雑な永続化クラスを生成する手法といえる．

♥ 正会員　高エネルギー加速器研究機構 計算科学センター（現職），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修了（北川研究室）　tadashi.murakami@kek.jp

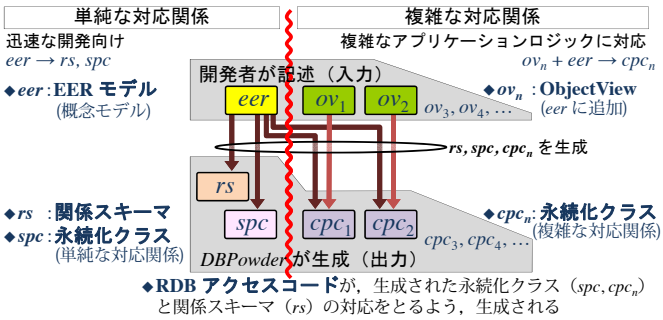

図 1　DBPowder フレームワークによる ORM の概要

本研究では，複数の ORM フレームワークを用いて四点の比較評価を実施した．(ア) ORM の記述力，(イ) 開発で必要とされる記述量，(ウ) 単純な対応関係のみで構成される ORM に対して複雑な対応関係を追加する際に，必要とされる編集回数や単語数，(エ) OO7 ベンチマークによる性能評価．比較評価の結果，本研究で示した提案の妥当性が示された．

本稿の構成は以下の通りである．2 節で DBPowder を提案し，3 節で本研究の評価と応用事例を示し，4 節でまとめを述べる．

2. DBPowder フレームワーク：概念モデリングに基づく O/R マッピング手法

DBPowder フレームワークによる ORM の概要を図 1 に示す．開発者は初め，EER モデル (*eer*) [9] のみを記述する．DBPowder は *eer* を用いて関係スキーマ (*rs*)，永続化クラス (*spc*)，および RDB アクセスコードを生成する．これにより，単純な対応関係による ORM が成立する．開発が進み複雑な対応関係をもつ ORM が必要になった場合には，開発者は *eer* に加えて ObjectView(ov_n) を記述する，DBPowder は *eer* と ov_n を用いて複雑な対応関係をもつ永続化クラス (cpc_n) を生成する．

本研究で扱うオブジェクトモデルは，ODMG3.0 [4] のサブセットを基本とし，Java を参考にした拡張を含む．図 2-1 の (a)–(g) に例を示す．本研究で扱う関係モデルは一般的に定義されるモデルを想定しており，図 2-1 の (h)–(k) に例を示す．本研究で扱う EER モデル [9] は，ER モデルを拡張してオブジェクトモデルを扱えるようにしたものであり，図 2-1 の (p)–(s) に例を示す．

2.1 節で EER モデルを用いた単純な対応関係による ORM について述べ，2.2 節で ObjectView を加えた複雑な対応関係による ORM を述べ，2.3 節で *eer* と ov_n を記述するための言語 DBPowder-mdl を示す．

2.1 EER モデルによる単純な対応関係を簡易に実現できる ORM 手法

迅速な開発が求められる開発フェーズでは，開発者は *eer* のみを記述する．DBPowder は *eer* を解釈し，単純な対応関係をもつ *rs* と *spc* を生成する．この手順を ORM 手順 1 – 4 に示す．

ORM 手順 1 DBPowder は，*eer* 上の主キーが省略された実体に，代理キーを追加する．

ORM 手順 2 DBPowder は，Teorey らの手法 [9] に基づき，*eer* から *rs* を生成する．

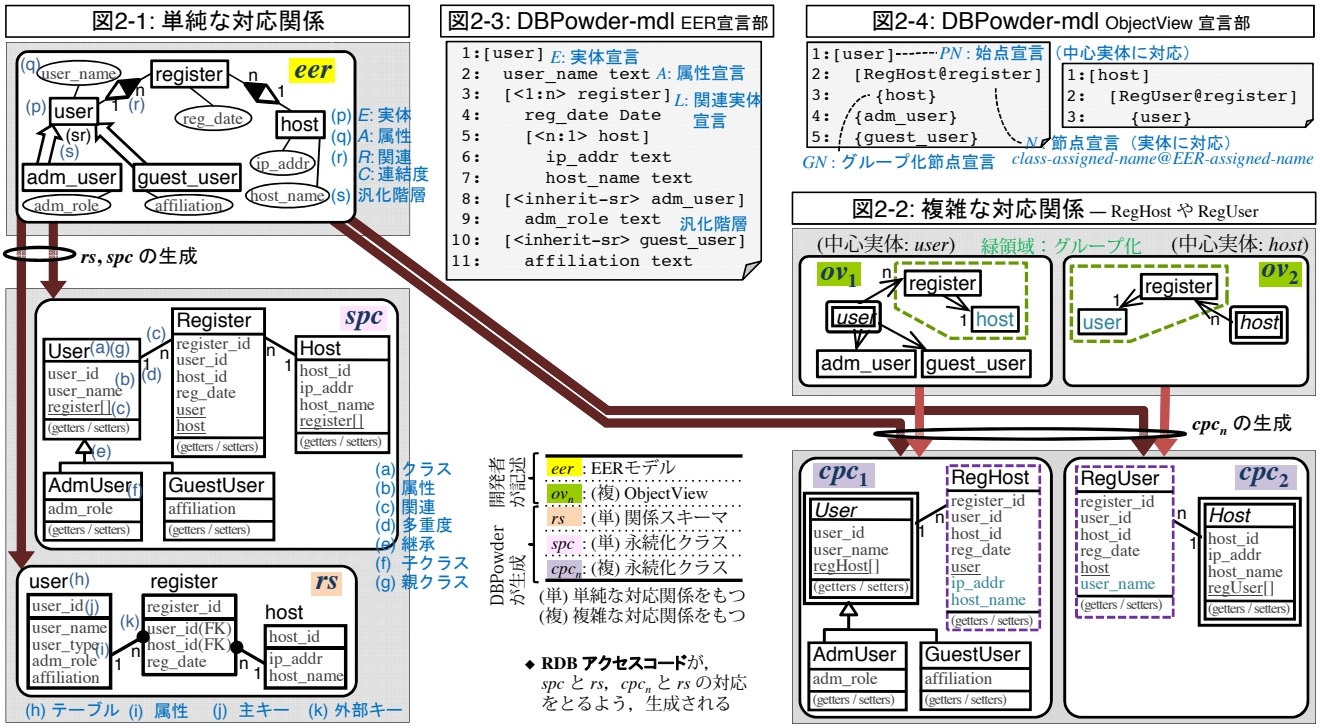

図 2　単純な対応関係（左側 図 2-1），複雑な対応関係（右側 図 2-2），DBPowder-mdl EER 宣言部（中央 図 2-3），DBPowder-mdl ObjectView 宣言部（右側 図 2-4）　[8]

ORM 手順 3 DBPowder は，ORM 手順 2 と類似の方法により，*eer* から *spc* を生成する．

ORM 手順 4 DBPowder は，単一テーブル継承（SR），クラステーブル継承（CR），具象テーブル継承（CCR）として知られる方法 [5] により，*eer* の汎化階層を *rs* で扱う．

単純な対応関係における ORM 手順 1 – 4 の実施例を 図 2-1 に示す．開発者は *eer* のみを記述する（図上部）．この *eer* に基づき，DBPowder は *rs* と *spc* を生成する（図下部と中部）．ORM 手順 1 – 4 に基づいて生成された *rs* と *spc* の対応関係を，DBPowder では単純な対応関係と定義づけている．

2.2 ObjectView による複雑な対応関係を実現できる ORM 手法

複雑な対応関係を実現するために，DBPowder では *ObjectView* を導入する．ObjectView は，EER モデルに対する有向グラフベースのオブジェクト記述形式である．アプリケーションロジックが必要とする実体と関連について，走査経路と利用方法を指定することで，開発者は複雑な対応関係もつクラス cpc_n を簡潔に記述できる．ObjectView がサポートする *rs* と cpc_n の複雑な対応関係は，以下に示す三種類である．(a) あるテーブルを基準にして多対 1 や 1 対 1 で結合される複数のテーブルを，一つの永続化クラスに対応づける対応関係，(b) テーブルの属性の一部を，対応する永続化クラス内で部品化し，別の永続化クラスとして対応づける対応関係，(c) 複数のテーブルに分散する各々の属性の集合を，一種類のインタフェースとして対応づける対応関係．

ObjectView は，以下に示す *ov* 記述 1, 2, 3 から構成される有向グラフ G によって定義される．開発者が記述する ObjectView の内容に従って，DBPowder は cpc_n を生成する．

***ov* 記述 1** アプリケーションロジックが必要とする実体と関連の，走査経路の指定による，有向グラフ G

はじめに *ov* 記述 1 で，開発者は対象のアプリケーションロジックが扱う実体 E と関連 R を指定する．このうち，アプリケーションロジックが中心に扱う実体を中心実体とよぶ．開発者は，実体 E を節点，関連 R を枝，中心実体を始点と見立てて，必要な要素を指定した有向グラフ G を記述する．G 内のそれぞれの枝は終点側にもとの R に対応した連結度をもち，G 内のそれぞれの節点はもとの実体 E 内の属性 A を引き継ぐ．このようにして，ObjectView は EER モデルにおける連結度や属性を引き継ぐ．

***ov* 記述 2** G を単純化するための，G の節点のグループ化指定

ov 記述 2 では，始点から辿って連結度が 1 の枝を用いて節点をグループ化することで，*ov* 記述 1 で指定した ObjectView の構造を単純化する．

***ov* 記述 3** G の部品化（コンポジション）と多態性（ポリモーフィズム）を実現する，構造化リテラルとインタフェースの指定

ov 記述 3 では，*ov* 記述 2 でグループ化した ObjectView に対して，構造化リテラルとインタフェースを用いて部品化と多態性を実現する．

ORM 手順 5 DBPowder は，*ov* 記述 1, 2, 3 に示す ObjectView の記述に従い，cpc_n を生成する．

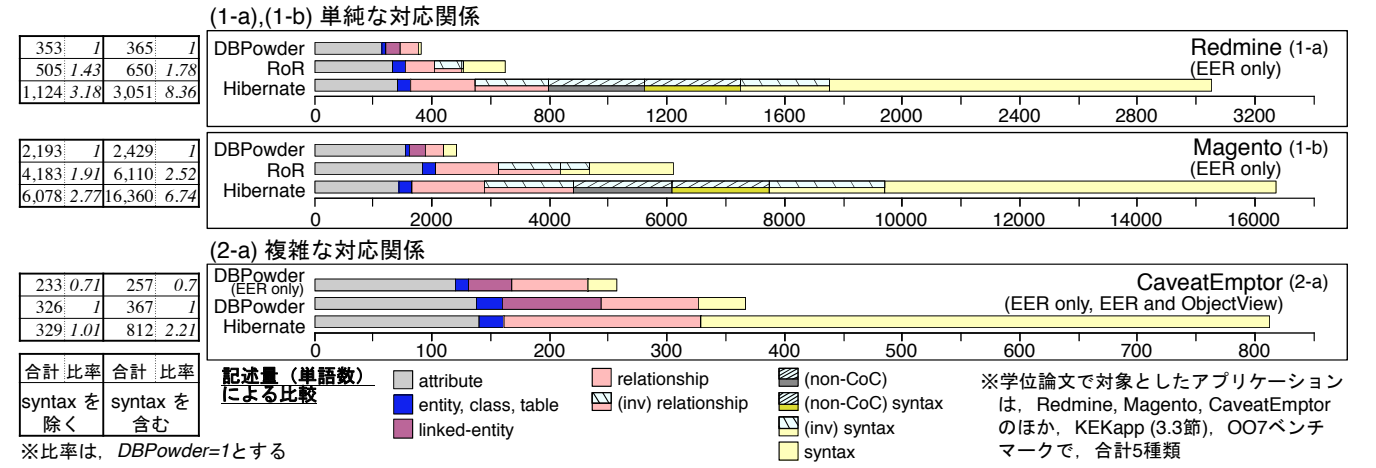

図 3　開発に要する負担の評価：単語数を用いた記述量の比較（一部抜粋）

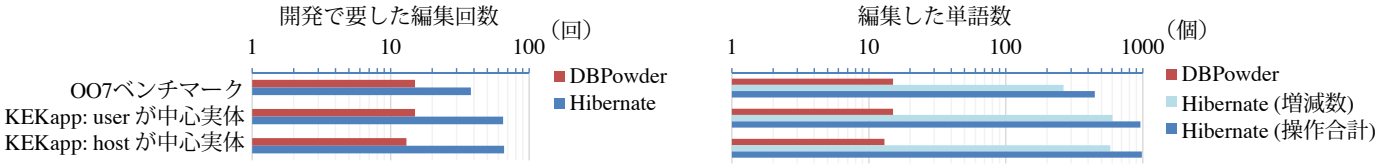

図 4　開発に要する負担の評価：複雑な対応関係に移行する開発で要した，編集の回数および編集した単語数

複雑な対応関係における ORM 手順 5 の実施例を図 2-2 に示す．この例の前提として，図 2-1 で開発者が *eer* を記述済であり，*eer* に基づき *rs* と *spc* を生成済である．図 2-2 では開発者は，ObjectView ov_1, ov_2 を記述する（図上部）．この ov_1, ov_2 に基づき DBPowder は永続化クラス cpc_1, cpc_2 を生成する（図下部）．

ov_1 は管理者 `user` を中心としたアプリケーションロジックを扱っており，*ov* 記述 1 に示すグラフ構造を記述したのち，*ov* 記述 2 により節点 `register` と `host` をグループ化している．この ov_1 に基づき DBPowder は永続化クラス cpc_1 を生成する，ここでクラス `RegHost` に対応する *rs* 上のテーブルは `register` と `host` の二つであり，複雑な対応関係が成立している．ov_2 や cpc_2 については，ホスト `host` を中心として，ov_1 や cpc_1 と類似の関係が成立している．

2.3　記述言語 DBPowder-mdl

DBPowder は記述言語 *DBPowder-mdl* をもつ．図 2-3, 2-4 に例を示す．DBPowder-mdl の記述形式には EER 宣言部（図 2-3）と ObjectView 宣言部（図 2-4）の二種類がある．

EER 宣言部の基本要素は実体 E，属性 A，関連実体 L である．DBPowder-mdl の記述形式は，関連実体 L を導入した上で基本要素を階層構造で記述することにより，EER モデルの記述の際に同じ名称を指し示す回数を減らしており，簡潔な記述を可能としている．なお実体のデータ表現が階層構造では不足する箇所では，E や L の同一要素を複数回記述することで，グラフ構造を記述できる．また汎化階層もサポートする．

ObjectView 宣言部では，中心実体に対応する始点 PN，節点 N，およびグループ化される節点 GN を用いて，ObjectView の有向グラフを記述する．

3.　評価と応用事例

3.1　記述力および開発に要する負担の評価

DBPowder の単純な対応関係や複雑な対応関係への適性を評価するため，記述力の評価，記述量の比較による開発に要する負担の評価（図 3），および複雑な対応関係に移行する開発で要した編集の回数および編集した単語数の比較（図 4）を行った．比較対象は DBPowder, Hibernate, Ruby on Rails (RoR) とした．

記述力の評価では，Hibernate がサポートする永続化クラスと関係スキーマの対応関係について，DBPowder-mdl が以下の三つを除いてサポートできることを確認した．1) 関連や継承関係をもたない複数のクラスを一つのテーブルに対応づける，2) SQL 文を永続化クラスや属性に対応づける，3) コレクションを構成するクラスから一属性を選んで配列にして，コレクションを保持するクラスの属性として対応づける．

ORM 実施のための開発に要する負担について，単語数を記述量と見なして比較した結果の一部抜粋を図 3 に示す．学位論文では，5 種類のアプリケーションについて単純な対応関係や複雑な対応関係の複数のパターンを対象とした．各々のアプリケーション上で ORM を実現するために必要なスクリプトを DBPowder，RoR，Hibernate で記述し，記述に要した単語数を比較した．XML のタグ名や属性名のような文法に属する内容は syntax として数えることとし，グラフ上では右部に配置した．

比較した結果，DBPowder では必要な記述量を大幅に削減できることから，単純な対応関係や複雑な対応関係を簡易に実現できることを示せた．RoR ではテーブル名などの命名規則への制約により開発に要する負担を軽減するが，DBPowder は概念モデルの効率的な記述により開発に要する負担を軽減する．このアプローチの違いが，RoR と DBPowder の削減効果の差異の要因と考え

られる．なお Hibernate では，syntax で膨大な記述量を要した．

二種類のアプリケーションについて，単純な対応関係から複雑な対応関係に移行する場合を想定し，DBPowder と Hibernate について開発に要する負担を比較したものを 図 4 に示す．2 種類とも，DBPowder では移行対象の実体や関連を ObjectView で指し示して記述するのみだが，Hibernate では移行対象のクラス定義の全体をコピー＆ペーストしたのちに，複数回の編集が発生した．結果として DBPowder では開発に要する負担を大幅に軽減できており，単純な対応関係から複雑な対応関係に移行する開発を効率的にサポートできることが示された．

3.2 DBPowder ORM システムと性能評価

本研究では，2 節 で示した提案手法を実現するシステム，DBPowder ORM システムを提案した．このプロトタイプシステムを構築し，OO7 ベンチマーク [3] を用いた性能評価を実施した．性能評価では，Hibernate で実装された OO7 のオープンソースプログラムと，これを DBPowder に移植したプログラムで，処理時間の比較を実施した．その結果，多くの場合について，DBPowder のほうが処理時間が短く，またほぼ全てについて，DBPowder の性能が著しく劣ることはないことを示せた．

3.3 応用事例

DBPowder の初期プロトタイプシステムと，これを用いたウェブアプリケーション（KEKapp）を 2006 年に構築し，以来改良を重ねている．KEKapp は，2006 年より高エネルギー加速器研究機構でのセキュリティ管理業務で利用されている [10]．さらに，KEKapp を 2011 年に連携組織 J-PARC に展開した．利用者数は 2013 年 12 月現在，およそ 140 名である．

KEKapp では追加開発を幾度も行い利用者に提供しており，セキュリティ管理や，部署ごと年度ごとに実施するセキュリティ点検における，業務上の要望に応えている．ここで，図 3，図 4 などの評価を実施しており，その結果として DBPowder の効果が高いことが示された．

4. まとめ

本論文では，概念モデリングに基づく O/R マッピング（ORM）手法に関する研究として，DBPowder フレームワークを提案した．本研究の貢献は大きく三点を挙げられる．一点目の貢献は，永続化クラスと関係テーブル間の，単純な対応関係の簡易な実現と複雑な対応関係の実現についてサポートを両立できる，実用的な ORM 手法を示したことである．この利点は，EER モデルと ObjectView を併用により連携することで，対象のデータモデルを柔軟に記述できたことで得られた．この方式は同時に，二点目の貢献である，単純な対応関係から複雑な対応関係に移行する開発の効率的なサポートをもたらした．三点目の貢献は，DBPowder のプロトタイプシステムを構築し，実運用環境に適用することでその有用性を示したことである．

提案手法 DBPowder の評価として，記述力の評価と開発に要する負担の評価を実施した．記述力の評価では，DBPowder-mdl の言語定義を，Hibernate の文書型定義（DTD）や Ruby on Rails の機能と比較することにより，DBPowder のサポート範囲を確認した．開発に要する負担の評価では，単語数を用いた記述量の比較と，複雑な対応関係に移行する開発で要した編集の回数および編集した単語数の比較を実施した．その結果，永続化クラスと関係テーブル間の，単純な対応関係の簡易な実現と複雑な対応関係の実現についてサポートを両立できることと単純な対応関係から複雑な対応関係に移行する開発の効率的なサポートが可能であることを示せた．また OO7 ベンチマークを用いた性能評価を実施し，DBPowder の性能が Hibernate に比べて多くの場合で処理時間が短くなることを示せた．これらの結果から本研究の妥当性を示すことができた．

今後の課題としては，データ更新処理の効率化，大規模データへの対応，リバースエンジニアリング，EER モデルと ObjectView の連携方式の深化が挙げられる．

【文献】

[1] A. Adya, J. A. Blakeley, S. Melnik, and S. Muralidhar. Anatomy of the ADO.NET entity framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 877–888. ACM, 2007.

[2] C. Bauer and G. King. *Java Persistence with Hibernate*. Manning Publications Co., Greenwich, CT, USA, 2006.

[3] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The OO7 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 12–21, New York, NY, USA, 1993. ACM.

[4] R. G. G. Cattell and D. K. Barry. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[5] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, MA, USA, November 2002.

[6] D. Heinemeier and et al. Ruby on Rails. http://www.rubyonrails.org/. (accessed in Dec. 2013).

[7] C. Kleissner. Enterprise objects framework, a second generation object-relational enabler. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 455–459. ACM Press, 1995.

[8] T. Murakami, T. Amagasa, and H. Kitagawa. DBPowder: A flexible object-relational mapping framework based on a conceptual model. pages 589–598, 2013.

[9] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[10] 村上 直. DBPowder-mdl: EoD と記述力を兼備した 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)*, 3(3):46–67, Sep. 2010.

村上 直 Tadashi MURAKAMI

1999 年京都大学工学部情報学科卒，2001 年東京大学大学院理学系研究科情報科学専攻修了．SE 職を経て現在，高エネルギー加速器研究機構 (KEK) 計算科学センター助教．2014 年筑波大学大学院システム情報工学研究科博士後期課程修了．博士（工学）．データ工学の研究に従事．日本データベース学会，情報処理学会，電子情報通信学会，ACM 各会員．