第 XXVII 部

XML による RDB の抽象化

第 27 部
XML による RDB の抽象化

第 1 章　はじめに

XIRD WG は、XML 技術を介して RDB にアクセスするための技術に関する議論、設計および実装を行っている。特に、RDB で実現可能な範囲でフレキシビリティを最大限にし、かつ、XML をインタフェースとする汎用的なミドルウェアの構築を目指す。

XIRD はミドルウェアとして構築されるものであり、RDB を置き換えるものではない。XIRD の目的は各種 RDBMS 間の差違を吸収し、より簡易に使用することである。また、ネットワークとの親和性を高めることで複数 RDB 間連携なども視野に入れている。

WG の成果はさまざまなアプリケーションで利用可能なものとする。その適用例として、Auto-ID WG や TTS WG と協調していく。

WG 活動の初年度である今年度はまず、議論を重ね、WG の目的の明確化と参加者の意思統一を行った。その後、ミドルウェアの設計を行い問題点を洗い出した。これらの議論を踏まえて、一次試作を実装した。

次章では今年度のこれらの成果をまとめた論文を記す。なお、この論文は平成 16 年 9 月に行われた情報処理学会 FIT2004 第 3 回情報科学技術フォーラムにおける論文を元にしたものである。

第 2 章　迅速なアプリケーション開発のための DB 非依存ミドルウェア構築

概要

アプリケーションのプロトタイピングはいかに迅速に行うかが重要である。多くのアプリケーションは外部に DB を持つが、DB の選定や準備、設計に時間がかかり、また、DB 実装の差違などにより、迅速なプロトタイピングが妨げられることがある。本研究では、XIRD と呼ぶ DB に非依存なミドルウェアを構築した。XIRD は、XML および XML-RPC を利用することで、DB 実装に依存せずにアプリケーションを容易に開発できる。また、多くのアプリケーション実装言語を容易に利用できる。さらに、XIRD を利用したサンプルアプリケーションを実装し、本研究の有用性を示した。

2.1 背景

Web 用アプリケーションなど、現在のアプリケーション開発では、データ交換形式や外部保存形式などに XML を利用することが増えている。

XML は階層構造にしたがってデータ構造とデータ自身を同時に記述することができる。また、データの意味を人間が理解可能な形でタグ名として表現することができる。このことも XML が広く利用されている理由の 1 つである。

データを保持するために RDBMS（Relational Data Base Management System）が多くの場面で使われている。RDB によりデータの結合や抽出を容易に行うことができ、また実装も多く実行速度にも優れている。

RDBMS を使用する場合、データ構造をあらかじめ規定しておく必要がある。さらに、データ構造を綿密に設計しておくことで、速度や保守性が向上するため、構造設計にコストをかけることが多い。しかし、プロトタイピングなど開発初期段階では、データ構造が変化することが多く、これには多大なコストがかかってしまう。

一方、XML はタグを利用することでデータの意味や属性、データ構造を XML 自身が表現できるため、データ構造を変更した場合でもアプリケーション全体に対する影響は少なくなる。

以上の理由により、データ構造の変更が頻繁におきるプロトタイピングや開発初期段階では、データ保持形式として RDB より XML を選択したほうが好ましい。

本研究では XIRD（common XML Interface to Relational Database）と呼ばれるソフトウェアを構

築した。XIRD は RDB へのインタフェースを XML を利用して記述することができる。XIRD を使用することで、アプリケーションは RDB 内のデータ構造を意識することなく、XML だけで RDB を使用することができる。これにより、柔軟にデータ構造が変更でき、アプリケーションが迅速に開発できる。

2.2 関連研究

2.2.1 Perl DBI

Perl DBI[247] は関数、変数、規約などを定義することで、RDB 実装間の差違を吸収するための機構である。これは、API（Application Programming Interface）として多くの言語に対して提供されている。

Perl DBI を利用することで異なる RDB 間の情報のやりとりを簡単に行うこともできる。つまり、ある RDB 実装 A と別の RDB 実装 B を単一ソフトウェアから同様にアクセスすることで、A から B へとデータをコピーすることも容易に可能である。

しかし、Perl DBI はあくまで DB 実装間の差違を吸収するためのものであり、RDB の枠組を越えるものではない。

2.2.2 O/R マッピング（Object/Relation Mapping）

Java などのオブジェクト指向型言語から RDB をオブジェクトとして扱う場合、オブジェクトとリレーションという異なるモデルを統合する必要がある。O/R マッピング（Object/Relation Mapping）とは、そのためにこれら 2 つをマッピングするものである。

これにより、あたかもオブジェクトを扱っているかのように RDB を扱うことができる。O/R マッピングツールには Hibernate[126] などがある。

異なるモデル間をマッピングによって接続するアイデアは XIRD と同等である。しかし、XIRD は XML を用いることで言語に依存しないマッピングを実現する。

2.3 設計

2.3.1 設計方針

XML は階層構造を元にデータ構造を記述する。しかし、一般的に RDB は列と行からなる二次元の表で構成される。そのため、単純に XML から RDB への変換は難しく、NeoCore XMS[224] などの XML DB のように XML の階層構造をそのまま DB に保持する研究が盛んに行われている。

一方 XIRD はあくまで迅速なアプリケーション開発に主眼を置き、そのための手法として XML を選択した。

XIRD では、データ構造に制限を持たせることで XML から RDB への変換を容易にした。そのため、XIRD は基本的な機能しか持たない。また、XIRD はあくまで迅速なアプリケーション開発のためのミドルウェアであり、プロトタイピングが主目的である。したがって、大規模あるいは高信頼性が求められるアプリケーションは想定しない。

また、一般的な Web アプリケーションでは、RDB のすべての機能を使うわけではなく、データの参照、変更、追加、削除といった基本的な機能だけを使うことが多い。したがって、RDB が持つ機能すべてを実現せずとも、プロトタイピングには多くの場合十分と考える。

提供する機能

これまで述べてきた設計方針にしたがい、XIRD が提供する機能を以下に示す。

- 情報の提供／追加／削除／更新：XIRD はこれら DB が持つ基本的な機能を備える。
- ロールバック：とくに複数人が同時にアクセスする Web アプリケーションなどを想定した場合、情報の追加、削除、更新など RDB の情報を変更する機能は他者が行う変更と衝突を招く可能性がある。これを回避するための最低限の機能として、ハッシュ関数を用いたロールバック機能を提供する。

2.3.2 全体概要

図 2.1 に XIRD の全体概要図を示す。

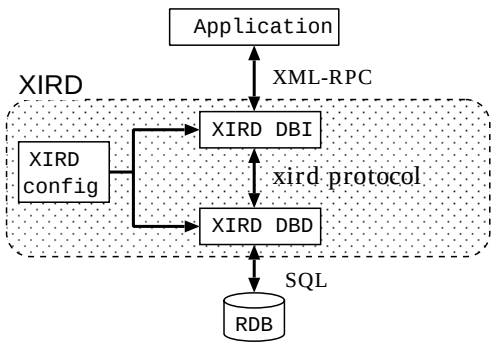

図 2.1.　XIRD 全体概要図

XIRD は、大きく XIRD DBI（XIRD DB Independent、以下 DBI）および XIRD DBD（XIRD DB Dependent、以下 DBD）モジュールの 2 つに分けられる。

本項ではこれら XIRD の構成要素について述べる。

XIRD DBI

DBI モジュールは提供するアプリケーションごとに 1 つあり、アプリケーションからの要求を待ち受ける。アプリケーションは要求を XML 形式で DBI に送信する。

要求を受け取った DBI は後述する XIRD 設定ファイルにしたがって XML を解析する。解析した結果を DBI は DBD に対して送信する。

これに対して DBD は RDB に対する処理を行い、DBI に返信する。DBI は XIRD 設定ファイルにしたがい XML へと変換し、アプリケーションに返信する。

XIRD DBD

DBD モジュールは、さまざまな RDB における実装の差違を吸収するためのモジュールである。

DBD は DBI からの要求を受け、RDB に対して要求を発する。また、要求に対する返答も DBD が受けた後に DBI に戻される。

RDB に対して直接アクセスするため、DBD は RDB の実装ごとに異なる。

XIRD Config（XIRD 設定ファイル）

XIRD は、各アプリケーションごとに設定ファイルを作成する。

この設定ファイルは、XML のタグ名で与えられるデータと RDB でのデータとの関連付け（マッピング）および RDB の設定が記載される。RDB の設定には DB のホスト名やユーザ名、パスワードなどが含まれる。

関連付けでは、XML のタグ名と RDB でのカラム名は一対一で対応しなければならない。また、XML は階層構造をとるため、異なる枝に同一名のタグが存在することが可能だが RDB は同一テーブル内では不可能である。そのため、XML のタグ名も RDB でのカラム名と同様一意でなければならない。

DBI および DBD はこの設定ファイルに基づき、アプリケーションと RDB との仲立ちを行う。

2.3.3 XIRD で使用される XML

情報の取得

情報の取得要求に含まれる XML の例を図 2.2 に示す。

図 2.2 には、<xird:query> タグがあり、その 1 段下には <nutrition> タグがある。この要素名がテーブル名となる。また、この要素の内容にはテキスト内容を含む <id> 要素がある。<xird:query> 内容のその他の要素にはテキスト内容を含む要素は存在しない。

XML を受け取った DBI は、<xird:query> タグ内のテキスト内容を含む要素（この例では <id> 要素）を条件式に、その他の要素を要求するカラムとして解釈する。

したがってこの要求例は SQL 文で表すと以下のようになる。

SELECT energy, protein, company, product
FROM nutrition WHERE id = 123456;

なお、テキスト内容が含まれる要素が複数ある場合、条件式はそれらを *and* でつないだものとして解釈される。

この要求に対する DBI からの返答に含まれる XML を図 2.3 に示す。<xird:row> 要素の中に含まれる形で返答が記載されていることがわかる。条件に適合する要素が複数ある場合、<xird:row> は複数存在する。

また、<xird:row> 要素は serial_no 属性を含む。この属性にはカラムを一意に示すための primary key が記載される。

XIRD 内でなんらかのエラーが生じた場合、DBI

```
<?xml version=''1.0''?>
<xird>
  <xird:query>
    <nutrition>
      <id>123456</id>
      <protein />
      <company />
      <product />
    </nutrition>
  </xird:query>
</xird>
```

図 2.2. 情報の取得要求の例

は内容にエラーが記述された <xird:error> 要素を付与して返信する。

情報の更新／追加／削除

返答に含まれる serial_no を指定することで、情報の更新／追加／削除ができる。

図 2.4 に情報の更新および追加の例を示す。

更新は <xird:row> 要素の serial_no 属性を指定し、更新する情報を <xird:row> 要素の内容に記述することで行われる。図の例では RDB 内の product というテーブルの情報を <product> 要素の内容に更新するよう指示している。

追加は <xird:row> 属性に serial_no が含まれない場合に行われる。追加は、前述の設定ファイルにおける accept_null 属性が false であり、その要素が含まれていない時に DBI はエラーを返す。

削除は <xird:row> が空要素である場合に行われる。

```
<?xml version='1.0'?>
<xird>
  <xird:result>
    <xird:row serial_no='3'>
      <id>123456</id>
      <protein>0.0</protein>
      <company>東チョコ S</company>
      <product>ライスチョコ</product>
    </xird:row>
    <xird:row serial_no='8'>
      <id>test</id>
      <protein>0.0</protein>
      <company>ダミーデータ</company>
      <product>ダミーデータ</product>
    </xird:row>
  </xird:result>
  <xird:hash>4e1bfd0ce75dccd59895d473ba626bee
  </xird:hash>
</xird>
```

図 2.3.　要求への返答の例

```
<?xml version=''1.0'' ?>
<xird>
  <xird:query>
    <xird:row serial_no='3'> <!-- 更新 -->
      <product>製品名変更</product>
    </xird:row>
    <xird:row>   <!-- 追加 -->
      <product>高タンパク質食品</product>
      <energy>581</energy>
      <protein>9999.0</protein>
    </xird:row>
    <xird:row serial_no='8'> <!-- 削除 -->
    </xird:row>
  </xird:query>
  <xird:hash>4e1bfd0ce75dccd59895d473ba626bee
  </xird:hash>
</xird>
```

図 2.4.　情報の更新／追加の例

2.4 実装

XIRD は Ruby-1.8 を、XML パーザには REXML [257] を利用して実装した。

2.4.1 XIRD DBI

アプリケーションは DBI に対して XML-RPC[330] を利用して要求を送る。また、DBI、DBD 間の通信は XIRD 独自のプロトコルにしたがって行われる。

2.4.2 XIRD DBD

DBD は、DBD モジュールそのものと使用する DB に依存する部分に分離している。DBD モジュールは主に DBI との通信を担当する。DBI から受けた要求をその DBD モジュールが扱う DB に対する要求へと変換する。この変換には、DBD は Ruby/DBI[268] を使用した。Ruby DBI は Perl/DBI の Ruby 実装である。

DBD モジュールは設定ファイルから DB に関する設定情報を読み込む。DBD はこの設定にしたがって、適切な DB 依存部分を呼び出す。

2.4.3 XIRD Config

図 2.5 に XIRD 設定ファイルの一部を示す。

<mapping> 内容に XML 要素名と DB のカラム名との変換マップを記述する。たとえば、<id> <map datatype="varchar(13)" accept_null="false" /> </id> という記述は、XML 要素名および DB のカラム名が "id" であり、DB におけるデータ保持形式が "varchar(13)" であることを示している。

また、<map> 要素の accept_null 属性は、情報の追加時に必須な情報であるかを示している。この属性が false である場合、情報の追加時には空要素が受け入れられないことを示す。

```
<?xml version=''1.0''?>
<config>
 <database>
   <xird_dbd>localhost</xird_dbd>
   <dbname> shirou </dbname>
   <username> shirou </username>
   <dbtype> Pg </dbtype>
   <primary_key>xird_serial</primary_key>
 </database>
 <mapper>
  <mapping>
   <nutrition>
    <id>
     <map datatype=''varchar(13)''
                          accept_null=''false'' />
    </id>
    <energy>
     <map datatype=''real'' accept_null=''true'' />
    </energy>
          :
   </nutrition>
  </mapping>
 </mapper>
</config>
```

図 2.5. XIRD 設定ファイルの例（一部）

2.4.4 ロールバック機能

XIRD ではアプリケーションが情報を取得した際に、その情報を一方向ハッシュ関数にかけた値もアプリケーションに対して送信する。アプリケーションが変更する際にはこのハッシュ値も同時に XIRD に対して送信する。

XIRD はアプリケーションから受け取ったハッシュ値と、XIRD が DB から再度取得した情報をハッシュした値とを比べ、一致していれば情報を変更する。一致していなければ他者が変更を行ったものとして変更せずエラーを返す。

2.4.5 XIRD Compiler

XIRD を利用する際でもテーブル作成など初期には RDB 自体を扱う必要がある。そのため、XIRDCompiler を実装した。これは、XIRD 設定ファイルからテーブルの作成など、RDB の初期設定を行うスクリプトを生成するものである。

しかし、これらの設定に関する記述方法は DB 実装に極めて依存しているため、XIRDCompiler はあくまで雛型でしかなく、問題が起きれば手動で修正する必要がある。また、複雑なトランザクションなどは記述できない。

2.5 評価

カロリー DB

XIRD に対する評価の 1 つとして、カロリー DB と呼ばれるサンプルアプリケーションを作成した。

このサンプルアプリケーションは、飲食物など商品が含むカロリーなどの栄養素に関するデータベースである。保持されている栄養素情報は商品に添付されているバーコードをキーとして呼び出す。

図 2.6 はカロリー DB の情報取得画面の例である。図に示されているように登録されている商品のさまざまな栄養素を表示できる。

これらの栄養素はユーザ自身が登録する。より多くのユーザが登録することにより、さまざまな商品に関する情報が蓄積される。このような情報はダイエットや健康のために役立ち、さらには介護・医療分野においても有意義であると考える。

情報取得だけでなく栄養素情報の追加も XIRD を経由して行われている。図で示したようなカロリー DB に対する Web インタフェースは PHP で実装した。また、PHP から XIRD を扱うための共通ライブラリも作成した。

図 2.6. カロリー DB 情報取得画面

XIRD を使用することで 3 時間程度でプロトタイピングが完了した。また、作成途中でビタミンなどその他の栄養素に関しても情報を追加したいとの要望があり追加を行った。また、i-mode 用の簡略化されたページも作成して欲しいとの要望もあった。これらの要望はごく短時間で実現できた。

派生クライアント

XIRD を使用することの利点の 1 つとして、Web アプリケーション以外にもさまざまなクライアントを容易に実装可能という点が挙げられる。カロリー DB に対するもう 1 つのサンプルクライアントとして、カロリーの値のみを取得し、値を csv ファイルに書き出すアプリケーションを作成した。この csv ファイルからは容易にグラフを生成することができる。

XML としては、カロリー DB のサブセットとしてカロリーのみを取得する XML を XIRD に送信している。XIRD を利用することで将来データ構造が変化した場合でもクライアント側はなにも変更する必要がない。

2.6 まとめと今後の課題

本研究では、アプリケーションが XML を使用して RDB を扱うためのミドルウェアである XIRD を開発した。XIRD により、以下の利点が得られた。

- アプリケーション開発中にデータ構造の変更を行っても RDB を変更する必要がない
- アプリケーションは DB 実装の差違を意識する必要がない
- データ構造に依存しないためソースコードの再利用が容易である

今後の課題として、現在の XIRD 実装では DBI および DBD に認証機構などのセキュリティ機構が未実装である点が挙げられる。今後はこれらの問題点を解決していきたい。