Vol.2011-DBS-153 No.31
2011/11/3

ページ要素をデータベースに直接結び付けるフレームワークとWebサイト設計手法

新　居　雅　行[†1]

筆者が開発したWebアプリケーション用フレームワークINTER-Mediatorは主として小規模な開発を考慮したものである．一般的なフレームワークではサーバ側でHTML文字列処理によるページ生成をするが，INTER-Mediatorはクライアント側でDOMノードに対する処理を行う．ページ内の要素の属性にテーブル名やカラム名の情報を付与することで，自動的にデータベースとページ要素が連動し，要素の属性や値が設定される．複数のレコードに対しては連動してノード複製を行う．サーバ側あるいはクライアント側に処理を追加して，複雑な処理を組み込める．

The Framework Binding Page Elements To Database Directly, And How To Involve In Web Site Development

Masayuki Nii[†1]

In this paper, I introduce the framework "INTER-Mediator" for Web application development. It could be applied to small scale development. Although general web application frameworks are working as the HTML generator with string operations on server, INTER-Mediator is working on client side with operating DOM nodes. By supplying the information of tables and columns to elements of the HTML page, these elements automatically synchronize with database. Each element will be duplicated with multiplying alone with records. It can be added on any programs to both server and client side, so it could develop with complex demands.

†1 連絡先：nii@msyk.net

1. はじめに

大規模なWebアプリケーションの構築では，MVC[1)]やJavaServer Pagesの仕様[⋆1]にある"Model 2"をアーキテクチャとして採用したフレームワークが使用される．一方，エンドユーザを中心とした小規模なWebアプリケーション開発向けの視点を持った技術は少ない．エンドユーザによる利用を想定したフレームワーク[2)]や，中小規模の開発向けのフレームワーク[3)]についても開発されており，生産性向上が報告されてはいる．しかしながら，一般には小規模開発にも大規模開発の手法がそのまま使えるという意識があると考えられる．小規模な業務系サイトは，ドメイン分析を十分に行わず，HTMLで作られたデザインを前提に開発を進めることも多く，オブジェクト指向設計が適用できない傾向にもある．MVCやModel 2によるフレームワークの有用性は疑いの余地はないものの，予算が限られる状況ではその適用は難しい．小さなコミュニティでの小さな開発では極力手間をかけずに開発をしたいといったニーズがある．

小規模開発において利用されるツールには，デスクトップアプリケーションとしてFileMaker[⋆2]やMicrosoft Access[⋆3]といった製品があり，手軽にシステム構築をしたいというニーズを満たしている．これらの製品の特徴は，データベースとユーザインタフェース要素を直接的に結びつけていることが挙げられる．Web開発でもそうした製品は従来から存在しており，最近ではMicrosoftはWindowsサーバを中心としたシステム構築に利用できるコーディングが不要なWebアプリケーション開発環境Visual Studio LightSwitch 2011[⋆4]を2011年7月にリリースしている．ジャストシステムも同様なコンセプトのUnitBase[⋆5]を2011年9月にリリースした．これらはデータベースのカラムとバインドしたユーザインタフェース要素をレイアウトに配置すれば，即座にデータベースの内容を参照したり変更できる点が特徴であり，開発ツールも提供している．

Webアプリケーションでもデータベースとページ要素を連動できれば，サイト開発が容易に短時間でできる可能性がある．このことをオープンソースの製品を中心に実現するためにINTER-Mediator[⋆6]を開発しソースコードを含めて公開した．サーバでページ生成をする

⋆1 http://www.kirkdorffer.com/jspspecs/jsp092.html#model
⋆2 http://www.filemaker.co.jp/
⋆3 http://office.microsoft.com/ja-jp/access/
⋆4 http://www.microsoft.com/japan/visualstudio/lightswitch
⋆5 http://www.justsystems.com/jp/products/unitbase/
⋆6 http://msyk.net/im

のではなく，クライアント側でテンプレート処理を行う手法を採用した．クライアント側でのテンプレート処理を行う手法により効率改善などが報告されている[4]．ページ定義において独自のテンプレート記述を行う方法ではなく，HTML そのままでデータベースと連動ができる方法を開発した．class 属性等にテーブルやカラム情報を記述し，その結果を DOM ノードとして処理をし，複数のレコードがあればノードを複製する．加えて，サーバ側およびクライアント側にプログラムを記述できるようにして，単なるデータベースとのバインドでは処理しきれない用途にも対応できる．

ブラウザの JavaScript の互換性の問題はどうしても発生するが，HTML5 対応のブラウザでの動作確認をしながら開発を進め，十分なパフォーマンスと開発の容易さが実現されたフレームワークが開発できた．サーバ側モジュールは PHP で開発したものを利用する．データベースには PDO[⋆1] に対応しているものと，FileMaker Server を利用できる．Apache を Web サーバに採用しているか，PHP が稼働する IIS を搭載した Windows Server であれば利用できる．

手軽に開発が可能な状況では，いきあたりばったりな開発がなされるケースも多く，結果的にアプリケーションがニーズを満たさないものになってしまうこともある．本論文ではオブジェクト図やクラス図を使った分析結果と対照する手法も提案する．

本論文では 2 章で簡単な検索ページを作成した結果を示し，3 章でページ生成の動作について解説を行う．そして 4 章では，INTER-Mediator を利用したページ作成をアプリケーションやデータ構造についての分析結果を元に進める手法を説明する．

2. サンプルで見るフレームワークの動作

2.1 郵便番号検索の実例

実際に INTER-Mediator を使用して開発したページの動作を紹介する．**図 1** は郵便番号から地名を検索する機能を持つ．MySQL にあらかじめデータベースを作っておき，日本郵便が配布しているデータを取り込んでおいてある．ページを表示した後，検索条件をテキストフィールドに入れてボタンをクリックすれば，その番号に前方一致する郵便番号のレコードをデータベースから取り出し住所を含めて一覧表示する．10 レコードずつ表示され，そのための前後のページに移動するボタンも自動生成されるようにした．

⋆1 PHP Data Objects：PHP のデータベースアクセスのためのライブラリ．

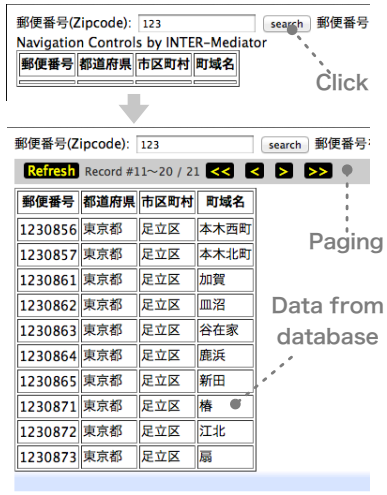

図 1 INTER-Mediator で作成した郵便番号検索ページ
Fig. 1 The Postal Code Search Page by INTER-Mediator.

```
1  CREATE table postalcode (
2    id  INT AUTO_INCREMENT PRIMARY
         KEY,
3    f3  VARCHAR(20),    #郵便番号
4    f7  VARCHAR(20),    #都道府県
5    f8  VARCHAR(20),    #市区町村
6    f9  VARCHAR(20));   #町域名
```

図 2 郵便番号データベースのスキーマ定義
Fig. 2 The Schema of “postalcode” Table.

2.2 スキーマと定義ファイル

スキーマは，**図 2** に示す通り，1 つのテーブルだけがある．サイトを作成するために 2 つのファイル「定義ファイル」と「ページファイル」を作成する．定義ファイルにはデータベース情報を PHP の配列の形式で記述する必要がある．このファイルは，サーバ側にあって Web サーバから呼び出されて実行できる必要がある．検索ページ向けに**図 3** のように作成した．5〜8 行目にあるように，postalcode というテーブルがあって，f3 カラムで並べ替えて，10 レコードずつ表示するという状況をここで定義している．4 行目の IM_Entry は INTER-Mediator に定義された関数である．

2.3 ページファイル

HTML で作成されるページファイルはレイアウトを決める．このファイルは原則としてどこにあってもかまわないが，通常は定義ファイルと同じところに配備して，Web ブラウザから参照できるようにする．郵便番号検索のページファイルは**図 4** のように作成した．ページファイルと定義ファイルが連動するように，ページファイルのヘッダセクションにある script 要素で，定義ファイルを読み込むようにしておく（4 行目）．

ページに 1 つのテーブルがあり，tbody 要素で 1 行分だけの表の内容が記述されている

```
<?php
require_once ('../INTER-Mediator/INTER-Mediator.php');  //INTER-Mediator の読み込み

IM_Entry( array( array(
    'name' => 'postalcode', //テーブル名
    'records' =>  10,     //1画面あたりのレコード数
    'paging' => true,   //ページ送り機能を利用
    'sort' => array( array( 'field'=>'f3', 'direction'=>'ASC' ),),),),  //並べ替え条件
  null, //その他の指定はなし
  array(
    'db-class' => 'PDO',  //データベース処理クラス
    'dsn' => 'mysql:unix_socket=/tmp/unix.sock;dbname=test_db' ), //PDOでの接続設定
  false);   //デバッグモードの指定
?>
```

図 3 定義ファイル include_MySQL.php
Fig. 3 The Definition File Named "include_MySQL.php"

(21〜33 行目)．表には 4 つのセルがあり，それぞれ class 属性に **IM[...]** の記述がある(27〜30 行目)．これがあれば，例えば最初の td 要素では，postalcode テーブルの f3 カラムの値がこの要素のテキストノードとして設定される．結果として表のセルに，データベースにある郵便番号のデータが表示される．複数レコードによって繰り返されるメカニズムは 3 章で説明する．

JavaScript の関数がヘッダセクションの script 要素で指定されている（5 行目）．関数内ではテキストフィールドにある文字列を取得して検索条件をオブジェクトとして指定している（7〜9 行目）．そして，INTERMediator.construct(true);（10 行目）によってページを解析し，**IM[...]** の記述のある要素にデータベースのデータが見えるようになる．この例ではボタンをクリック後にページ構築をしているが，body 要素の onload 属性に INTER-Mediator.construct(true); を記述すれば，ページの読み込みと同時にページの構築を行うこともできる．また，id 属性が IM_NAVIGATOR の DIV 要素（20 行目）が自動的にページ送りのコントロールをページ上に生成する．

3. INTER-Mediator の動作

3.1 リンクノード

前の例にあるように，HTML ページ内の要素の class 属性に **IM[...]** と記述して，テーブル名やカラム名を指定すると，その要素にデータベースのデータが設定される．class 属性を利用する理由は，スタイルシートに定義されていないキーワードはページ生成で無視されるからである．スタイル等の名前と区別がつきやすいように，**IM[...]** といった記述にした．

```
<html>
<head>
    :   <!-- 省略 -->
  <script type="text/javascript" src="include_MySQL.php"></script>
  <script type="text/javascript">
    function doSearchFromZipcode()  {
      var param = {field:'f3', operator:'LIKE', value:document.getElementById('zipcode').
          value+'%'};
      INTERMediator.additionalCondition["postalcode"] = param;
      INTERMediator.startFrom = 0;
      INTERMediator.construct(true);
    }
  </script>
</head>
<body>
  <div>郵便番号(Zipcode):
    <input type="text" id="zipcode" value="123"
      onkeydown="if(event.keyCode==13){doSearchFromZipcode()}" />
    <button onclick="doSearchFromZipcode()">search</button>郵便番号を前方一致で検索します.
  </div>
  <div id="IM_NAVIGATOR">Navigation Controls by INTER-Mediator</div>
  <table border="1">
    <thead><tr>
      <th>郵便番号</th><th>都道府県</th><th>市区町村</th><th>町域名</th>
    </tr></thead>
    <tbody>
      <tr>
        <td><div class="IM[postalcode@f3]"></div></td>
        <td><div class="IM[postalcode@f7]"></div></td>
        <td><div class="IM[postalcode@f8]"></div></td>
        <td><div class="IM[postalcode@f9]"></div></td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

図 4 HTML で記述されるページファイル
Fig. 4 The Page File Described by HTML

このテーブル名やカラム名の指定のある要素を「リンクノード」と呼ぶ．IM の後の [ ] 内は，**テーブル名@カラム名@設定先**と記述できる．**設定先**を省略すると，その属性の子ノードとしてテキストノードを作成しデータベースの内容をテキストとして設定する．ユーザによる入力や選択を行う input，select，textarea 要素については，**設定先**の指定をしない場合にはユーザに見えるテキストとして表示される．**テーブル名**と**カラム名**は省略できない．

設定先には他に，属性名,「style. スタイル要素名」,「innerHTML」を記述でき，それら指定した設定先にデータベースから得られたデータを設定する．最後に # を付けることで，既存のテキストに追加できる．一連の記述を｜（縦棒）で区切って複数指定できる．

3.2 エンクロージャとリピータ

construct 関数が呼び出されることにより，ページ内を解析し，要素の階層を考慮しながら,「エンクロージャ」と「リピータ」という 2 種類の要素（リピータは要素群もある）を認識する．フレームワーク内での処理を**図 5** に示した．まず，上位のノードから走査を行い，リンクノードを発見する．そして，そこから階層をさかのぼり，最初に見つかるエンクロージャとリピータのノードを探す（図 5 の 1）．例えば，前の郵便番号検索の例だと，div 要素がリンクノードとして存在するが，そこから階層を遡った tbody 要素をエンクロージャ，そしてそれに含まれている 1 つの TR 要素をリピータとする．

そして，エンクロージャとリピータの認識ができれば，一度リピータを削除して別途保存しておく（図 5 の 2）．リピータ以下のリンクノードの情報を集めてデータベースにアクセスする（図 5 の 3）．前の例では，リンクノードにあるテーブル情報は postalcode だけなので，このテーブルにアクセスを試みる．そして，いくつかのレコードが得られるが，レコードの数だけリピータを複製する．つまり，残してあるリピータを複製してエンクロージャの子ノードとし，そこに含まれるリンクノードにデータベースのカラムの値を指定に応じて設定する（図 5 の 4）．削除したリピータを複製してカラムの値を設定する処理を，レコードの回数だけ繰り返す（図 5 の 6）．こうしてノードの複製により，レコードがページの上で「展開」されることになる．前の例では結果として，tbody のエンクロージャ内にレコードの数だけ TR の要素ができて，データベースの内容が表になる．もし，リピータの中にさらにエンクロージャとリピータのノードがある場合，その内部のエンクロージャより下位のノードは上位のノードの展開処理のときには特になにもしない．ノードを複製してカラムの値を設定した後，リピータより下位のノードの展開を行う（図 5 の 5）．エンクロージャとリピータに設定できる要素は，**表 1** にまとめた．

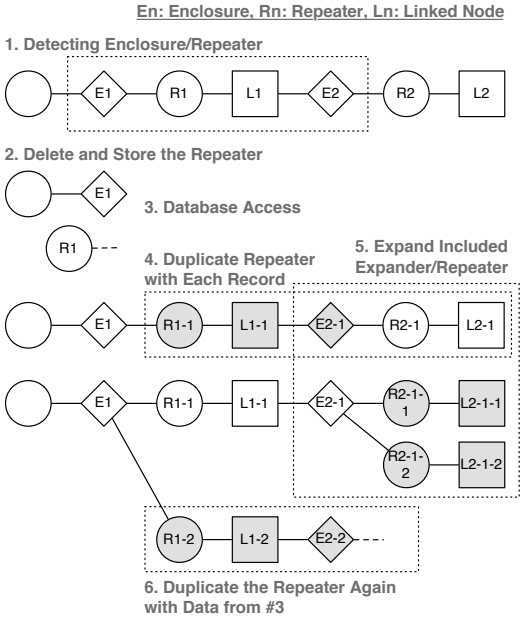

図 5 ノード複製の手順
Fig. 5 The Process of Node Duplication.

3.3 上位階層のリピータとのリレーション

上位のエンクロージャ／リピータと下位のエンクロージャ／リピータとの間では，1 対多のリレーション展開もできる．逆にリレーションとは関係ない展開もできる．上位のリピータのレコードと，下位のリピータのレコード間での関連付けは，リレーションシップで利用するカラムを利用すれば良い．指定がなければ関係なく展開される．具体的には定義ファイルに下位のリピータの外部キーのカラム名と，そのカラムに対応する上位のリピータのカラムを指定する（具体例は 4 章で解説する）．

図 6 に 1 対多の関係のあるようなデータベースの内容を展開した結果を示した．表全体が 1 レコードとすると，表の下半分はその 1 レコードに対応した複数のレコードがあるような形式のものだ．下半分ではリピータの繰り返しによって複数の行として展開されている．つまり全体のテーブルの最後の行（tr 要素）の中に，さらに table を定義している．つまり，下半分の 2 行に渡っている部分は，全体から見れば，リピータの中にあるエンクロージャとなる．下半分の繰り返しの中の左のドロップダウンリストは 1 行のリピータの中に，さらにエンクロージャ／リピータが定義されていて，選択肢はデータベースのテーブルから取得している．

こうして，エンクロージャ／リピータの繰り返しを階層的に記述することで，データのリ

表 1 認識可能なエンクロージャとリピータ
Table 1 Acceptable Enclosures and Repeaters

Type of Component	Enclosure [class]	Repeater [class]	Linked Node	
			Repeater	Enclosed
Table	tbody	tr	-	OK
Generic	div [_im_enclosure]	div [_im_repeater]	OK	OK
Number List	ol	li	OK	OK
Item List	ul	li	OK	OK
Popup Menu, List	select	option	OK	-

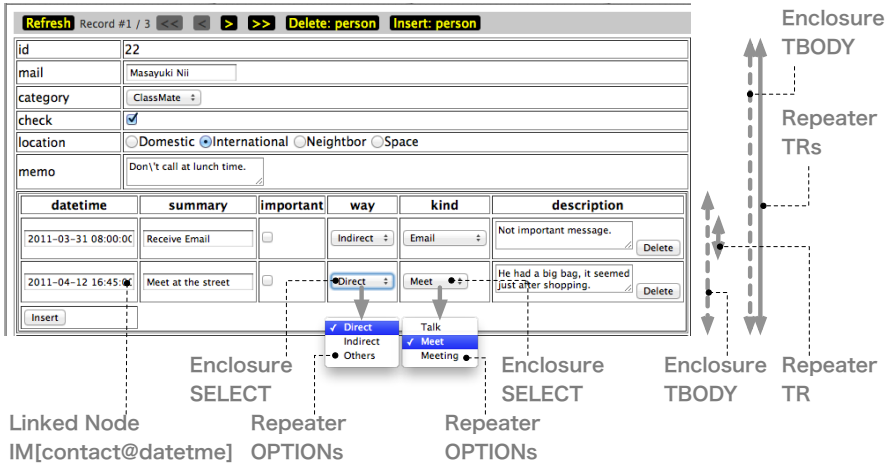

図 6 エンクロージャとリピータの例
Fig. 6 The Example of Enclosures and Repeaters.

レーション結果をもとに，関連する複数のレコードを理論上は無制限に展開できる．実装の上では再帰呼び出しで実現している．ただし，循環参照的になると無限ループになるため，繰り返しは適当な数より多くなるとそこで展開を自動的に中止するようにしている．

ページ展開をしたとき，リンクノードに設定した値やそのリピータを展開するときに利用したレコードのキーの値を記録しておく．テキストフィールドでは変更時に呼び出されるメソッドで，記録していた情報をもとにデータベースへの書き込みの処理を行える．ページ展開時の値も保持してあるので，その値と変化がないことを確認して書き込むことができ，楽観的ロックの仕組みも持っている．INPUT などのフォーム要素以外の要素では，独自にデータベースアクセスのための関数等を呼び出して，データの書き込みを組み込める．

3.4 クライアントとサーバでの拡張

INTER-Mediator では単なる 2 階層的な仕組みだけでなく，サーバサイドおよびクライアントサイドで，処理を追加する仕組みを作った．その概要を示したのが**図 7** である．INTER-Mediator 本体と別に「データベースアダプタ」として，データベース処理の部分を別オブジェクトして作成できるようにしている．これらはサーバ上に配置しているが，実際にはクライアントにダウンロードした上で稼働する．なお，執筆時点でのバージョンでは本体と分離させる理由はないが，将来的に HTML5 のローカルデータベース対応やクラウドのデータベース対応を考えていて，データベースアダプタの切り替えでの対応を予定している．

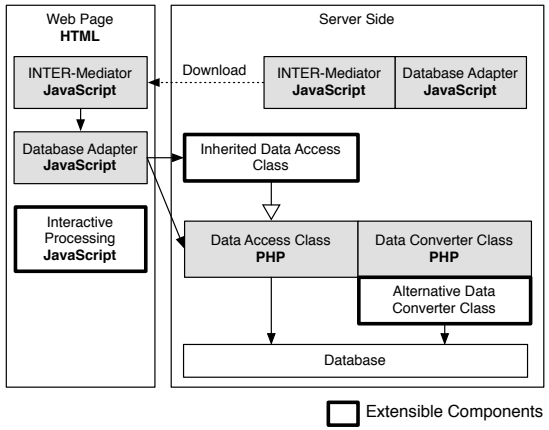

図 7 INTER-Mediator での処理機能の拡張
Fig. 7 Extensible Components of INTER-Mediator.

サーバ上でデータベース処理をするのは「データアクセスクラス」である．このクラスでは，読み出しや新規作成など CRUD（Create, Read/Retrieve, Update, Delete）に対応した 4 のメソッドを持っている．例えば読み出しメソッドの場合は検索条件などを与えて，データベースから取り出した結果を配列で戻すメソッドがある．取り出し内容を大きく処理したい場合は，データアクセスクラスを継承したクラスを定義する．データベースとの基本的な読み書き処理は元のクラスが行うため，継承したクラスでは開発者が行いたい加工の処理だけを記述すれば良い．INTER-Mediator の Web サイトではこの仕組みを利用して，ブラウザの言語に応じてどの言語の文書を取り出すかを判定するプログラムを持った独自のデータアクセスクラスを利用している．

さらに「データコンバータクラス」は 1 つのカラムの値を変更するために用意される．読み出しと書き込みの 2 種類のメソッドを決められた名前で作る．たとえば，日付の表記の変更や，数値に通貨単位やカンマを付けるといった処理をクラスとして定義する．そして，定義ファイルにおいて，特定のカラムでデータコンバータクラスを利用する指定ができる．このクラスも独自に作って指定できる．

クライアント側では自由に JavaScript のプログラムを作成できる．INTER-Mediator との連動処理によりプログラム作成をサポートしている．伝票を作成するようなときには，例

えば単価と個数から金額を計算するようなニーズが発生する．次の章ではそれをビューを構成して行う方法を紹介しているが，ページ展開後に値を変更した場合の再計算ができる必要がある．そのためには，JavaScript の関数を組み込む事になるが，INTER-Mediator によって展開されたノードは id 属性を意図的に書き換えている．そのため，展開前の id 属性値を使ってノードを参照して計算するといったことができない．そこで，class 属性等に記述したリンクノードのための情報を利用して，他のノードへの参照を見つける関数を用意している．また，展開直後にイベント処理的に独自に定義した関数を呼び出すこともできる．こうした仕組みを使えば，Web ブラウザ上での展開やユーザによる操作の機会を得てプログラムを実行できるので，さまざまな機能を組み込むことはもちろん，展開されたデータベースの処理も組み込むことができる．

このように，サーバ側ではパイプ処理的な機能拡張を追加するとともに，クライアント側では自由に JavaScript のプログラムの追加が可能である．FileMaker や Access のようなデスクトップアプリケーションで利用できるスクリプトは，クライアント側の JavaScript に対応するものと言えるが，サーバ側のカスタマイズはほとんどできない．INTER-Mediator はその意味でこれらのデスクトップアプリケーションよりも汎用性が高いとも言える．

4. サイト開発の手法

Web ページを含むサイトやアプリケーションを開発するとき，UML やアジャイル等の効率的な開発手法を取り入れるのが一般化している．INTER-Mediator を使った開発においても，UML におけるクラス図やオブジェクト図を元にした開発が可能であり，モデリングの結果をページ開発に割り当てることができる．**図 8** はその一例を示したものである．

4.1 オブジェクト図とクラス図による要求の記述

ここではオブジェクト図とクラス図を利用し，それぞれの操作については考慮しないものとする．属性だけに注目した手法はメタデータオブジェクトモデル（MDO）として提唱されており[5]，MVC や O-R マッピングとは異なるモデルとしている．

ここでは，データベースにある売上等の情報から伝票形式の表示を Web ブラウザで行う仕組みを作る事にする．まず，システム利用者にとっての要求がある．伝票形式での帳票であっても，どんな種類のデータをどのように配置するのかなど，仕様としてまとめることになる（図 8 の 1）．ここで，スケッチとして描いた Web ページ上のデータの関連性をオブジェクト図として記述してみる．同一の属性の値が一覧になっていれば，それを複数のレコードであると見なす．単純に並んでいるだけの属性は 1 つのレコードとして見ることにす

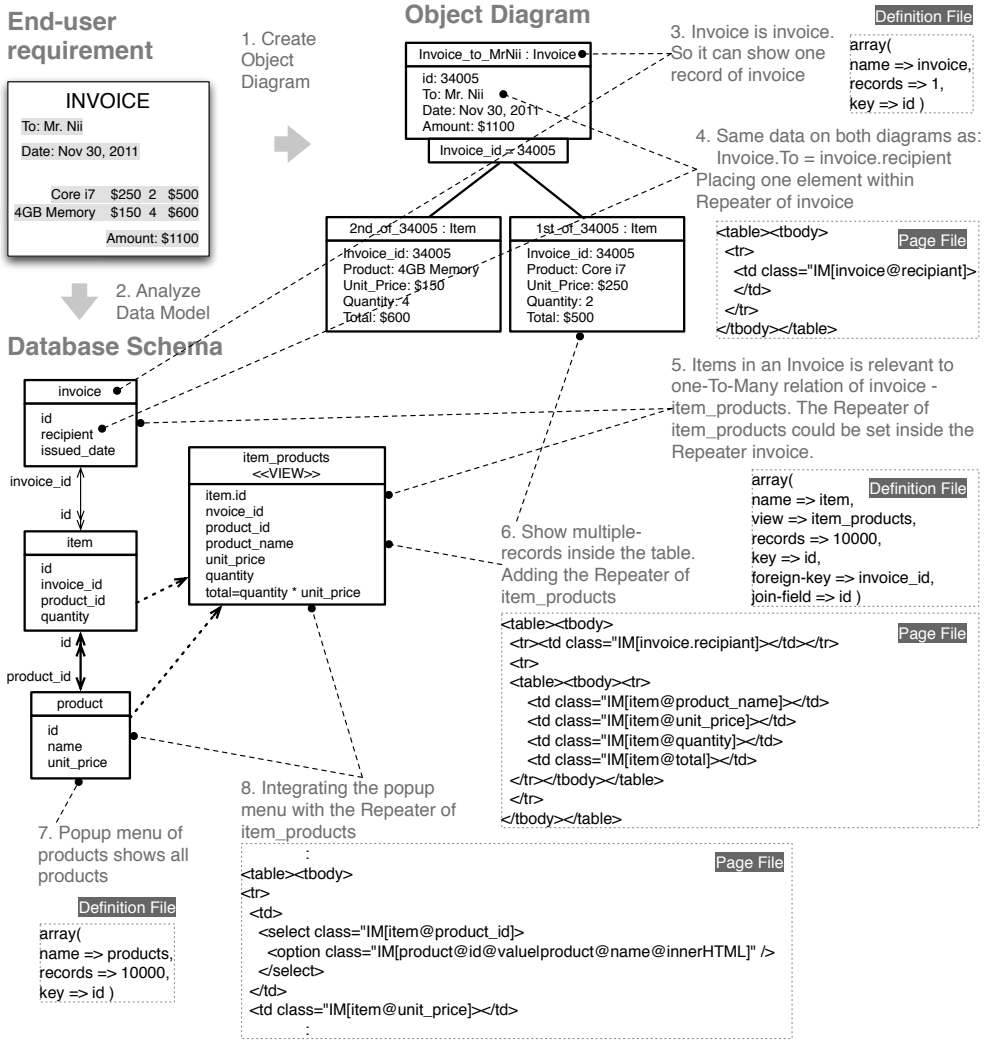

図 8 INTER-Mediator を利用したページ開発の流れ
Fig. 8 How to build a web page with INTER-Mediator

る．そして，伝票と明細のような 1 対多の関係があれば，それぞれ異なるオブジェクトとして図を作成する（図 8 の 2）．一方，実際にどのようにデータを記録すればいいかということすなわちデータモデルはクラス図で記述できる（図 8 の 3）．新たにシステムを作る場合だと，最初の要求を満たすようにデータのモデルを作る事ができる．一方，既存のシステムに新たにページを追加したいという場合には，すでに定義済みのデータモデルが存在することになる．

4.2 最上位のテーブルとカラムの対応

ユーザが得たいものをオブジェクト図にし，データモデルをクラス図で作成する．このようなシンプルな例では，オブジェクト図の Invoice とクラス図の invoice，そして Item と item がそれぞれ対応するのは明白である[⋆1]（図 8 の 3）．この事実から定義ファイルを作り始める．ページの宛名などの部分には invoice テーブルのデータが必要である（name ⇒ invoice）[⋆2]．そして，ページには 1 枚の伝票だけが表示されるので 1 レコードだけを取り出す（records ⇒ 1）．さらにこのテーブルのキーカラムは id である（key ⇒ id）．こうして，定義ファイルの一部の記述が作られる（図 8 の 3 の右の方）．

Invoice と invoice の対応する属性を調べてみる．すると Invoice クラスの To 属性と，invoice クラスの recipient 属性は同一のデータであると言える．従って，invoice クラスのカラムである recipient を取り出すリンクノードを定義すれば良い（図 8 の 4）．こうすれば，該当するカラムの値を表示する．図には示していないが，Invoice の Date と invoice の issued_date も同様に同じデータである．ただし，一般にはデータベースが出力する日付と伝票に表示したい日付のフォーマットは違う．従って，「<td class=”IM[invoice@issued_date]”></td>」という表記をページ上に行えば良いが，定義ファイルでこのカラムに対してデータコンバータクラスを適用し，値をデータベース形式と元号を使う形式との相互変換をする指定を入れればよい．

4.3 1 対多の関係を定義ファイルに記述する

次に，Invoice と Item の関係は，invoice と item との関係に対応することから，明細部分を検討する．ただし，item だけでは Item にある全ての属性をまかなえない．この場合の 1 つの対処法として，図にある item_products というビューを定義し，invoice と item_products の対応に注目する（図 8 の 5）．まず，クラスとして item_products が加わったと考えるべきなのだが，一般にはビューは読み込みしかでない処理系が多い．そこで，定義ファイルでは「name ⇒ item」「view ⇒ item_products」という記述を行う．こうすれば，読み出しは item_products から行い，書き込みは item テーブルに対して行う．

Invoice と Item の 1 対多の関係は，Item クラスの Invoice_id 属性の値から参照できるものである．クラス図で言えば，item テーブルの invoice_id に対応する invoice テーブルの id の値を入れることでリンクが成り立つ．このクラス間の関連付けを定義ファイルに追加する．つまり，item（item_products）の invoice_id が外部キーとなる（foreign-key ⇒ invoice_id）．その外部キーと対応するのが invoice テーブルの id カラムである（join-field ⇒ id）．なお，records は単に非常に大きな数値にしているに過ぎない．ここで，invoice テーブルとリレーションするという情報が定義ファイルの「name ⇒ item」を含むキー付けされた配列に必要ではないかと思われるかもしれないが，それは不要である．理由は引き続いて説明する．

4.4 1 対多の関係をリピータに含まれるエンクロージャで表現

1 対多のリレーションは，リピータの内部にエンクロージャ／リピータのセットを作る．図 8 の 4 で伝票の宛先などを table 要素を使って配置しているが，その最後の行にさらに table 要素を内部に記述する．そうすれば，tr 要素の中に table 要素が入ることになる．外側のテーブルのリピータ内にさらにエンクロージャ／リピータのセットができ，内側の table 要素内に明細が展開される（図 8 の 6）．このとき，定義ファイルにあるキー付けされた配列のうち，「name ⇒ item」を含む定義を使用するが，外側のリピータは「name ⇒ invoice」によって展開されているわけであるから，外部キーと関連付けるテーブルは invoice であることは確定している．従って，「name ⇒ item」のキー付けされた配列の定義には「invoice テーブルを示す情報」は不要となる．明細では，product_name，unit_price，quantity，total のそれぞれのカラムがテーブルのセルに入り，明細の数だけ行数が増やされる．

ビューの item_products から得られるデータは，name 属性の「item」をテーブル名として，リンクノードに記述する．「item@quantity」という記述はまだしも「item@unit_price」という記述はやや違和感があるかもしれないが，item という名前で一連のデータを代表する名前と解釈するべきだろう．また，仮に参照先が「item@unit_price」の要素がテキストフィールドだった場合，データを修正してデータベース更新すると「item テーブルの unit_price カラム」へのアクセスになってしまう．この例では td 要素を利用しているので更新はできず動作上の支障はない．

⋆1 オブジェクト図のクラス名は大文字，クラス図のクラス名は小文字で始めることにする．また，クラス図のクラス名はデータベースのテーブル名と同一とする．
⋆2 本文および図のこの記述は，PHP のキー付けされた配列を意味する．演算子⇒を挟んで左側がキー右側が値を意味する．正しくは文字列なので，クォートで囲む必要があるが見づらくなるのでクォートの表示は省略している．

ここではビューを使ってオブジェクト図とクラス図の対応を取っているが，ビューを使わない方法も可能である．ただし，Invoice に対して，多対 1 となる invoice と product のリレーション結果を適用させることになる．クラス図では多対 1 となるが，実際のデータでは 1 対 1 の対応となる．この場合，invoice と item_products のときと同様に，item と product の関係をエンクロージャ／リピータで展開すれば良い．定義ファイルに「name ⇒ product」のキー付けされた配列が加わることになる．また，エンクロージャとリピータをセルの中に作るには div 要素を利用する．

4.5 繰り返されるリピータ内部でポップアップメニューを配置

もしここで，この伝票は入力のためのユーザインタフェースであるとして，商品名はポップアップメニューで選択したいという要望が出たとする．選択肢は product テーブルのデータをとりあえずは全て利用するものとする．従って，product テーブルから取り出すという指定を含むキー付けされた配列を，定義ファイルにさらに追加する（図 8 の 7）．ポップアップメニューつまり select 要素はその行の選択した商品に関連する値を持つので，select 要素自体はリンクノードである必要がある．しかしながら，select と option の要素がエンクロージャとリピータとして稼働するので，明細のテーブルのセルに，select 要素を記述する（図 8 の 8）．option 要素は 1 つだけで良く，product テーブルのレコード数だけ option 要素が複製される．option 要素の値に product テーブルの name カラムの値を指定し，VALUE 属性に id カラムの値を指定する．1 つのノードに複数のリンク設定を記述した．これで，ポップアップメニューには商品名が並ぶが，選択した結果は option 要素の id 属性の値であり，その値が item テーブルの product_id 属性に設定される．

以上の方針のもとで，定義ファイルとページファイルを作成する．ページファイルの body 要素の onload 属性に「INTERMediator.conctruct(true);」と記述すれば，ページをロードした直後にページの展開が行われる．実際には，invoice テーブルへのアクセスで，何らかの制約が必要になるが，それらは定義ファイルや JavaScript のプログラムでの指定する．

5. ま と め

クライアント側で DOM ノード複製によるデータベース連動の Web ページを作成するフレームワークを開発した．ページ生成においては独自のテンプレート言語を利用するフレームワークが多いが，INTER-Mediator では HTML のままでテンプレートの表現が可能である．そして，データベースと Web ページ上の要素との直接的なやりとりを完全に自動化するとともに，追加の処理やイベント処理等が必要な場合にはそれぞれサーバおよびクライアント側にプログラムを追加することで対応ができる．データベースとページ要素が連動しているという点での容易さと，プログラムの追加による複雑な要求への両方に適合できるフレームワークである．

オブジェクト図とクラス図からページ作成ができる一方，ページ上での表現とデータベース側のモデルが大きく違う場合の対処方法については明確な指針が与えられていない．Web アプリケーションの開発で出てくるようなさまざまなパターンを抽出して，どのような対処をすることでサイト構築に結びつけるのかを詳細に洗い出し検討する必要があると考えられる．加えて，Web アプリケーションにおけるニーズを洗い出してフレームワーク上での対処を文書化する必要がある．オブジェクト図とクラス図は静的な状態のモデリングには向くものの，ユーザの操作を交えたモデル記述手法についても検討する余地はある．

参 考 文 献

1) Krasner, G.E. and Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80, *J. Object Oriented Program.*, Vol.1, pp. 26–49 (1988).
2) 中所武司：業務の知識を有するエンドユーザ主導のアプリケーション開発技法 : フレームワーク・ドメインモデル・サービス連携，電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学， Vol.107, No.331, pp.19–24 (2007-11-12).
3) 北野 優，本間 圭，高橋佳嗣，上崎達也，宮内一平，齋藤 敬，鈴木博勝，松田豊臣，富樫 敦：画面遷移設計を基盤とした小規模 Web システム開発用 Web API フレームワークと NST・褥瘡システム開発への応用，*IPSJ SIG Notes*, Vol.2008, No.54, pp.29–34 (2008-06-12).
4) Tatsubori, M. and Suzumura, T.: HTML templates that fly: a template engine approach to automated offloading from server to client, *Proceedings of the 18th international conference on World wide web*, WWW '09, New York, NY, USA, ACM, pp.951–960 (online), DOI:http://doi.acm.org/10.1145/1526709.1526837 (2009).
5) Xudong, L., Xiaofei, X., Dechen, Z. and Limin, Q.: An Adaptive Development Framework for Web-Based Enterprise Information System, *Proceedings of the 2008 International Symposium on Computer Science and Computational Technology - Volume 02*, Washington, DC, USA, IEEE Computer Society, pp. 82–86 (online), DOI:10.1109/ISCSCT.2008.127 (2008).