ユーザー開発や保守が可能なWebアプリケーションフレームワークの開発

Web Application Framework Enabling to Develop and Maintain by Users

新居 雅行

電気通信大学大学院博士後期課程
情報システム学研究科
社会知能情報学専攻

博士（工学）の学位申請論文

2015 年 3 月

ユーザー開発や保守が可能なWebアプリケーションフレームワークの開発

Web Application Framework Enabling to Develop and Maintain by Users

主任指導教員 : 石川 冬樹 客員准教授
主査・指導教員 : 大須賀 昭彦 教授
指導教員 : 田中 健次 教授
審査員 : 田原 康之 准教授
審査員 : 古賀 久志 准教授

Abstract

INTER-Mediator is a framework for developing web applications and it can be applied to IT systems of small organizations that do not have large budgets. Common frameworks require imperative programming and consequently software engineers should be involved to develop any systems. On the other hand, Web pages based on INTER-Mediator can be synchronized with a database simply by using declarative descriptions. Moreover declarative descriptions realize repeating multiple records, associated records on one-to-many relationship, create and delete recode, pagination, authentication and authorization.

Although software engineers should build databases according to schema from scratch, users like end-users of business system can still be involved in the development especially in the maintenance phase if modifications can be done within declarative descriptions. The situation of database driven web system modification can be divided into 6 categories; “page element,” “request to database,” “response to single field,” “UI customization,” “response to database” and “schema change.” Last three ones are handled by software engineer, but first three ones are able to modify within declarative descriptions and users can handle them.

The web page template is described by the pure HTML without any special descriptions. Query results from database include multiple records and the question is how they should be merge into the template. The framework recognizes the elements for one record as “Repeater” and the parent of repeaters as “Enclosure” from the HTML template, and repeats the Repeater with corresponding to each record. The typical example is a table with TBODY for the Enclosure and TR for the Repeater.

If someone wants to develop with INTER-Mediator, he/she should learn the development method. The question is if user can learn it. To prove the learnability, subjects from web designers participated the experiment that includes study and examination sessions. One group couldn’t get it and abort the examination in short time. Other subjects took time about 2 hours and the scores scattered from 40 to 90%. So the later subjects took enough time to complete and the scores shows the result of learning. As the result, it concludes the INTER-Mediator has the learnability.

If users, designers, etc. can participate in the processes of the system development, the total cost can be reduced, and small- and medium-sized organizations will have more opportunities to introduce web-based business systems.

概要

データベース連動の Web アプリケーション向けのフレームワーク「INTER-Mediator」では，宣言的な記述で開発を進められる．手続き的なプログラミングに比べて，学習コストが低下し，その結果，業務システムのエンドユーザーなど，現場でシステムを利用する立場のユーザーが開発の一部や保守作業をできるようになる．初期開発と同程度の費用がかかるとされる業務システムの保守開発をユーザーが行うことでコスト削減が可能となり，システムの継続的進化をより低いコストで実現する．本論文では，フレームワークを利用した開発方法や実装手法を説明し，保守作業を宣言的な記述の範囲で行えることを示す．さらに，開発手法の学習がエンジニアリングの専門家でなくても可能なことを，被験者実験を通じて検証する．

目 次

図 目 次

表 目 次

アルゴリズム目次

ソースコード目次

第1章　はじめに

1.1　Webベースのシステム開発での問題点

データの共有や業務効率の改善のためのシステム開発は，組織の規模に関わらず行われている．データを効率良く記録し取り出せるようにするために，データベースが利用されており，さまざまなリレーショナルデータベース製品をベースにしたシステムが開発されている．加えて，多種多様なデバイスからシステムを利用できるようにするために，Web ブラウザで参照や利用ができる Web 技術の適用が望まれるようになった．本論文では，データベースを使用し Web をユーザーインタフェース手段として用いる業務システムやあるいはネットワークサービス等を開発する手法を扱う．通常，データベースの機能だけでは Web 経由でデータ保存や取り出しはできない．データベースへの入出力を可能にするためのユーザーインタフェースや処理ロジックの組み込みが必要になる．そのためにデータベースと Web ブラウザの間を埋める「Web アプリケーション」を構築する必要があり，そこでの処理はシステムの目的や用途によって異なるため，原則としてシステムごとに違うものを構築しなければならない．その Web アプリケーションを開発するための基盤となるのが「フレームワーク」である．

Web アプリケーション向けのフレームワークには，MVC アーキテクチャ[143] を採用し，機能の組み込みを Java，C#，PHP，Ruby などの手続き的プログラミング言語を用いて行われるタイプのものが広く利用されている．フレームワークはアプリケーション開発に必要なさまざまな機能を，クラスライブラリや，あるいは開発ツールとの連動機能として提供する．標準的な処理を容易に組み込めると同時に，手続き的なプログラミングによって多彩な機能を実現できる．これらのフレームワークを使うことで，開発者は高い自由度を得ることになり，要求に応じたきめ細かな機能の組み込みができることもあって，システム開発や Web 上でのサービス提供などに広く利用されている．

広く利用されているフレームワークでは，生成する Web ページのひな形となるテンプレートを HTML やあるいは独自の記述ルールで用意するのが一般的である．そして，データベースに対してクエリーを実行したり，得られた結果を体裁良く画面に表示したり，入力した値をチェックするなどの機能は，フレームワークの機能を使うために手続き的なプログラミング言語で記述される．従って，開発はもちろん開発後の保守においても，手続き的プログラミングの知識が必要になり，設計から実装や保守を含めて，システムに関連する作業の多くをシステム開発業者に発注することになる．軽微な変更や機能追加であっても発注をしなければならないという状況になる．そこでの問題点として，予算が限られ必要な改変ができなくなることが挙げられる．そうなるとシステムを利用する効果が低下して陳腐化し，使われないシステムになってしまう可能性もある．外注で進める場合，発注を受けた側は慎重に進めたいので価格設定が高めになる一方，利用者側としては小さな変更なのに時間も費用も想定以上にかかってしまうと写りがちである．さらに費用をかけても顧客が思い描く結果がなかなか得られないと利用者は不満を抱く [148]．ビジネス環境は刻一

刻と変化するので，利用者はシステムを変化に即した改良をしたいのだが，思い描くようには達成できない現状がある．

1.2　エンドユーザー開発を促進する宣言的な手法

情報システムの利用者は「エンドユーザー」と呼ばれ，エンドユーザーのニーズに合わせたシステムを構築することは，企業の IT 化における基本的な目標である．業務システム開発は，システム構築を誰が行うのかという点から，エンドユーザーが外部の専門業者に依頼する「受託開発」と，エンドユーザー自身が行う「内製」に大きく分かれる．内製の場合，社内に専門の部門や担当者がいる場合と，社内の通常業務の傍らシステム構築を行う場合がある．本論文で対象としたユーザーは，業務システムの場合は，主として最後に挙げた自身の業務を持ちつつ，システム化を進めようといったエンドユーザーである．

開発したシステムが業務用ではなく，自社の顧客向けにオンラインサービスとして提供される場合は，自社の顧客をエンドユーザーと呼ぶのが一般的である．この場合に本論文が対象としているユーザーは，サービスを提供するためにシステムを構築するスタッフである．顧客などの外部に対するシステム場合は，顧客向けサービスを提供するスタッフを本論文ではエンドユーザーと呼ぶことにする．すなわち，本論文での「エンドユーザー」は，実業務に関連するシステム開発に関わることが可能なユーザーを総称するものとする．

Web システム開発においては，開発側スタッフにはエンジニアだけでなく，Web デザイナも参画することがある．また，ユーザー企業には IT コンサルタントをはじめとして IT 化に関わる人たちが，実務担当者の周辺に存在する．これらエンドユーザーに接点があるような人たちについてもエンドユーザーによるシステム開発に関われる可能性がある．

自身の業務を持つエンドユーザーやその周辺の人たちが，業務システムの開発作業を業務と並行してできるようにすることを目指して，Web アプリケーション用フレームワークの「INTER-Mediator[56, 55]」を開発した．対象とするような人たちがソフトウェアエンジニア並みの知識を得てそれを保持するという状況は考え難い．エンジニアリングの知識がなくてもシステム開発の部分的な作業に関われるようにするために，宣言的な記述を主体に開発作業をできるようにすることを考えた．具体的には，HTML で記述された Web ページのひな形に，属性を追加したり，あるいは項目リストのような形式で，データベースを利用するための諸情報を記述するといったものである．この手法で開発を進めることができれば，手続き的プログラミング特有の知識は不要となる．加えて 1 つのページを作るための記述量が少なく，エンドユーザーが意図した結果を短時間で実現できることを目指した．

手続き的なプログラミングの習熟に対する困難さを示す事例として，プログラミングの学習過程に関する 6 つの障壁がある [62]．その中で最も大きな障害である「デザインの障壁」，すなわちアルゴリズムを正しく構築しなければならないという障壁は手続き的なプログラミングでは必ず発生する．例えば,「一覧」を作るということは，手続き的なプログラミングでは，繰り返されるレコードをループで処理してそれぞれ表示するように記述する必要がある．これに対し，同じことを宣言的に記述できるということは，フレームワークが繰り返し処理のアルゴリズムを正しく再現する作業を肩代わりしていることになる．宣言的な記述で機能を呼び出すことで，開発者は繰り返しの記述をプログラミング言語で書く必要がなく,「デザイン

の障壁」は低くなり，学習効率は高くなる．

1.3 エンドユーザーによる保守で実現する費用の軽減

システム開発を何もないところから始めるとすれば，最初の設計や実装においては，データベースのスキーマ設計のような専門家でないとできないような作業がある．本フレームワークを使うとしても，エンジニアリングの知識がないと初期開発は十分に進められないので，一定の機能を組み込むところまでは外注あるいは社内にいる専門家に依頼して進める必要がある．一方で，開発の途中からの作成物の改定作業や保守の作業（本論文ではこれらを併せて「保守」と位置付ける）であれば，宣言的な記述の変更で，ページの要素やあるいはデータの取り出し時の条件の変更などができるので，エンドユーザーでも取りかかれる作業である．従来はすべての開発作業を専門会社に外注したり，あるいは社内の専門部隊に依頼していたが，本フレームワークを使うことで，保守に関わる作業をエンドユーザー自身で行えるようになる．

エンドユーザーによって保守ができることでの 1 つのメリットは，外注の経費を下げられることができる点である．企業のシステム開発のコストの比率は，開発 24%，保守 31%，運用 45%といった調査結果がある [139]．受託開発の場合，概ね開発と保守の費用を合計した金額がシステム開発費である．保守費用を完全に 0 にはできない可能性はあるものの，外注費用の半分が保守費用とすれば，開発における外注先へ支払う費用は半分になる．もちろん，保守を内製することでの人件費の増加が懸念され，作業配分への配慮も必要になる．しかしながら，外注で発生するような依頼内容の指示や説明の作業を軽減できることなどから，軽減される作業も発生する．すべてを外注する状況ではなくなり経費構造は変化し，運用次第で経費の節減には大きく貢献できる．

国内に 380 万あまり存在する中小企業や小規模事業者は，IT 化にかけるコストが少なく，システム化されていない業務も少なくはない．中小企業や企業の部門内のような予算規模が小さな組織で開発したシステムの利用価値を保持するには，コストをかけずにシステム開発できる仕組みが必要である．本フレームワークを使用すれば，エンドユーザー自身が保守作業を実施できるようにすることで，コストの低下を狙うことができる．コストの点だけでなく，エンドユーザー自身がシステムに取り組むことで，実態に即したシステム保守を進められる可能性もある [61]．

1.4 宣言的な記述による保守の実現と学習可能性の検証

エンドユーザー主体の保守作業ができるような仕組みを実現するために，Web アプリケーションに必要とされる機能を宣言的な記述で実現できるように本フレームワークを開発した．例えば，HTML の特定の要素に属性として，フィールド名を記入すると，あるレコードの指定したフィールドの値が表示される．この仕組みにより，HTML で記述したページのテンプレートに，属性を追加することで，データベースと Web ページの論理的な結合，すなわち「バインド」が実現する．さらに，テキストフィールドなどのフォーム向けの要素では，バインドによりフィールドのデータの表示だけでなく，ユーザーによって修正された結果を元のフィールドに描き戻す仕組みまでを，属性への追記によって実現する．複数のレコードに対するバインドが必要なときには，例えば表要素の場合はテンプレートとして 1 行分だけを記述しておくことで，レ

コードの数分の行を複製し各行の各レコードを割り当ててバインドをする．読み出しと更新だけで済むようなページ作成は，データベースが用意されていれば手続き的なプログラミングを必要としない．

本論文では，開発した本フレームワークが，宣言的な記述でデータベース連動の Web アプリケーションを開発および保守することができ，その結果，エンドユーザーが開発の一部や保守作業への参画を実現していることを実証する．エンドユーザーによる保守が現実的なものであれば，本フレームワークは業務システムの継続的進化を低コストで実現可能な開発環境を提供するものであると言える．

宣言的な記述を主体に開発や保守ができることで，一般的なフレームワークであれば手続き的なプログラムの修正が必要な箇所でも，本フレームワークで作ったシステムであれば宣言的な記述で修正できる事例を示すことができる．本論文では，保守で実際に行われる作業を,「ページ要素」「データベース要求」「単一フィールド」「ユーザーインタフェースのカスタマイズ」「データベース応答」「スキーマ変更」と 6 つに分類した．それらの分類の中での典型的な保守作業を想定して，それが宣言的な記述でできる点を示す．データベースのスキーマ変更のように専門的な知識が必要な作業は無理としても，最初の 3 つの分類では宣言的な記述で改変できる作業が一般のフレームワークよりも多く，これらの分類に属する保守作業の多くはエンドユーザーでも取り組めることが示された．本件に関する議論は第 4 章に記述した．

一方，宣言的な記述であっても，それが学習をすることによって記述できるようになることが確認されない場合には，手続き的なプログラミングの知識がなくても取りかかれるとはいえ，現実にはもっと別の前提知識が必要かもしれない．本フレームワークが学習可能である点を確認するために，Web デザイナーを中心とした人たちに対して，宣言的な開発手法に関する学習を行い，その後に試験に臨むといった実験を行った．一部の被験者は学習ができなかったものの，半数以上の被験者は半分以上の問題に対しての正解を得ており，学習によってフレームワークに特有の記述を行う知識を得て，問題に対する回答を導き出すことができた．この点から，HTML を記述する能力があるような被験者群に対しては学習が可能であることが示された．加えて，データベースアプリケーションを利用するなど，データベースに関係した作業を過去に行った被験者ほど高得点を得る傾向があった．本件に関する議論は第 6 章に記述した．

1.5　本論文の構成

以下，第 2 章に現在の中小企業を取り巻くシステム開発の問題点と，エンドユーザー開発による解決方法について考察する．第 3 章には本フレームワークの機能や基本的な開発方法を説明する．第 4 章で保守作業に対する適合性を検討する．第 5 章では本フレームワークの動作上での特徴を解説し，どのように実装を進めたのかを説明する．第 6 章では本フレームワークの学習可能性に対する被験者実験の結果を説明する．第 7 章ではフレームワークがどのような業務に向くのかを検討し，実際の利用者の評価や運用実績を紹介する．

第2章　エンドユーザー開発と求められる要件

最初に開発における保守費用の割合が高いことを示す．そして，中小企業での情報化の状況から，システム開発へのニーズがあることを説明する．開発を進めるには，受託開発を始めとした外注だけでなく，エンドユーザー開発による内製もある．現場の業務担当者がシステム開発にかかわるといったエンドユーザー開発に対する状況を分析した上で，INTER-Mediator がエンドユーザー開発に対して保守作業を可能にし，加えて学習しやすい点が貢献することを説明する．

2.1　保守の比率が高いシステム開発費用

企業を対象にシステム開発費用のコスト構造を調査したのが，日経システム運用ナレッジの「企業情報システムの運用管理に関する実態調査 [139]」（調査時期は 2013 年 1〜2 月）である．調査結果によると，システム開発のコストの比率は，開発 24%，保守 31%，運用 45%となった．さらに，売上高が 100 億円未満の企業では，開発 21%，保守 34%，運用 45%と保守の割合が高くなった（図 2.1）．売上規模の小さな企業ほど，保守費用の割合が増えると分析されている．また，社団法人日本情報システム・ユーザー協会による「ソフトウェアメトリックス調査 2011[141]」の保守調査報告によると，初期開発にかけたコストに対して，保守と追加の開発のための費用を 5 年に渡り集計すると 1.23 倍，つまり，初期コストと同額を超える保守開発コストが開発後に発生している．

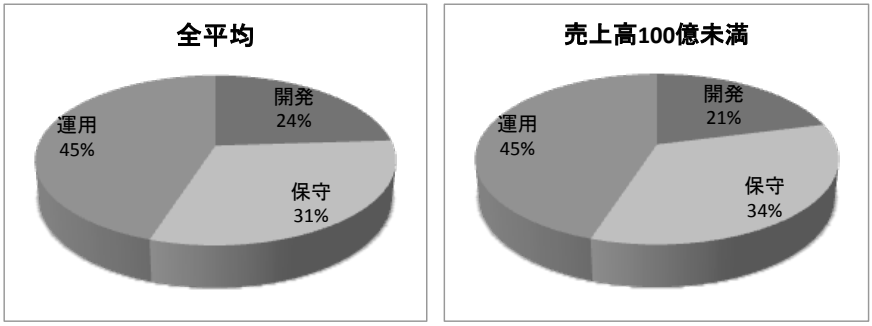

図 2.1: システム開発費用の内訳 [139]

いずれも，初期開発とその後の保守開発が同程度の費用がかかることを示している．初期開発にかかるコストだけでなく，保守開発や運用に費用がかかる点は従来から指摘されており，受託開発においても，初期開発だけでなく，保守に関しても引き続き受託する契約を結ぶことは多い．本論文では，開発した Web アプリケーションフレームワークの INTER-Mediator を利用した開発によって，保守開発をエンドユーザーが行うことでコスト構造を変化させ，経費の節減を目指すことを提案する．

2.2　中小企業の情報化の現状

経済産業省中小企業庁の調査発表 [121] によると，2012 年 2 月現在で，日本にある 386 万の事業者数に対して，その 99.7%の 385 万者が中小企業および小規模事業者である．内訳は大企業が 1 万者，中小企業が 51 万者，小規模事業者が 334 万者である．中小企業の売上規模は大企業に比べて小さいため，システム化は大企業ほど進んでいないと見られているのが一般的であるが，以下の調査結果から，実際にシステム化の余地があり従業員も必要性を感じていることが伺える．

2.2.1　中小企業ではシステム化されていない業務がある

2012 年に実施された「中小企業等の IT 活用に関する実態調査 [126]」は，従業員 300 人未満の 1887 件の事業所を対象にしたもので，売上が 50 億未満の事業所はそのうち 58%であった．IT 化の現状に対する調査結果では，パソコン，インターネット，メールへの対応はほぼ 100%であったのに対して，拠点間の指示や報告は 49%，掲示板等のグループウエアは 46%，e コマースは 21%，アフターサービスは 13%という結果になった．基幹業務の IT 化状況では，財務会計が 88%，顧客管理が 65%と比較的高く，続いて人事総務が 51%，物流・在庫が 50%，生産・製造が 39%，研究・開発が 21%となっている．パソコンなどのように購入するだけで利用できるものは高い利用率になるが，その他の用途は 50%に満たないものが多く，基幹システムも会計や顧客は比較的普及していると言えるものの，他の用途では普及率は概ね半分以下となる．つまり，中小企業ではパソコン等の機材は導入されているものの，システム開発による業務の効率化や，運用が必要なシステムの利用面では改善の余地があると言える．

日経 BP コンサルティングによる 2012 年に公開された，従業員 300 人未満の国内の企業の 327 人対する「中堅中小企業の IT 利活用調査 [138]」では，改善の必要性が最も高い課題として「システム化・自動化の遅れ」が挙げられ，改善の必要性に関しては「事業継続計画，リスク管理，セキュリティー」「人材確保や体制整備」「コスト，省電力，省力化」「IT 資産の一元管理」「拠点間連携」といった項目が順に並んだ．つまり，中小企業の従業員の問題意識として，システム化の遅れが最も高く，セキュリティーに関することも比較的高いという調査結果が出ている．

2.2.2　中小企業のシステム化が遅れている理由

中小企業のシステム化が遅れている理由は，予算が少ないことと，システム開発を行う要員が少ないことである．経済産業省が実施した平成 25 年度の「情報処理実態調査 [120]」（調査は平成 24 年=2012 年の状況）から，その様子が伺える．

すべての業種に対して従業員数で調査対象を分類し，事業収入に対する情報処理関連費用の比率を示したのが図 2.2 である．また，情報処理関連費用の中のソフトウェア関連費用の比率を示したものが図 2.3 である．いずれも，従業員数の分類ごとの 1 社あたりの平均値である．ソフトウェア関連費用には，ライセンス費用などが含まれるが，開発にかける費用もここに含まれると考えれば，従業員数の少ない組織ほど，開発にかける予算規模は少なくなる傾向にあると言える．その結果，IT 化が進まないか，あるいは進める

としても社内のスタッフの手に依ることになる．

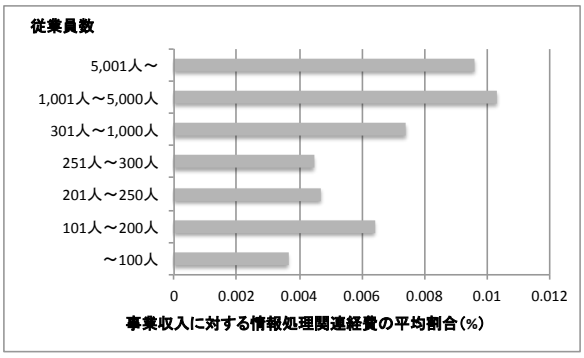

図 2.2: 情報処理費用の比率 [120]

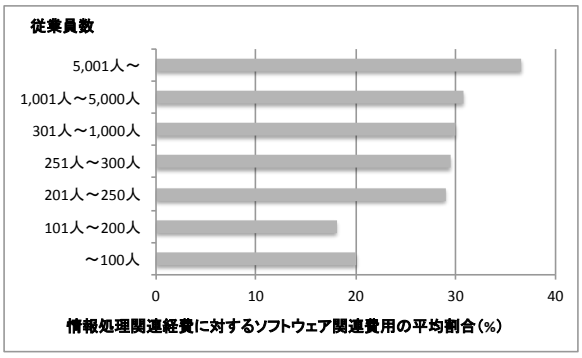

図 2.3: ソフトウェア関連費用の比率 [120]

同じく平成 25 年度の情報処理実態調査から，事業収入で組織を分類し，情報処理要員数の平均値を示したものが図 2.4 である．自社で雇用している内部要員と，派遣等で常駐している外部要員を色分けした．規模の小さな企業ほど要員が少ない傾向にあると同時に，規模が大きな企業ほど外部要員の比率が増える傾向にある [1]．規模の小さな組織ほど，社内の要員によるシステム開発や運用を余儀なくされている状況であるとも言える．

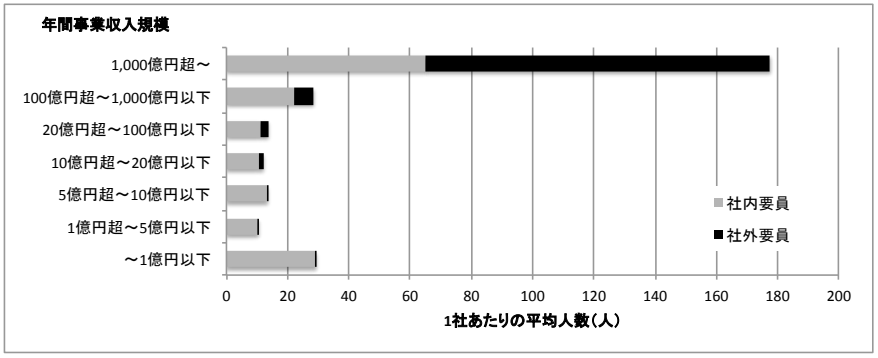

図 2.4: 事業収入ごとの情報処理要員の平均値 [120]

受託開発が大企業を中心に行われていることを示す調査として，2008 年に財団法人名古屋市工業技術振興協会が実施した愛知県内の IT ベンダーを中心とした 132 社に対する「中小企業向け情報システムの現状調査 [133]」がある．この調査は，受託開発を受ける側の企業に対して行われた．調査によると，販売先企業の売上高が 5 億以上の件数が 78%に達しており，IT ベンダーの業務の中心は中小企業ではなく，中堅以上の規模の企業である．また，中小企業への営業を行っていない IT ベンダーは 39%あった．

この調査結果は，受託によるシステム発注が可能な中小企業は少なく，多くの受託開発案件は中堅企業以上の規模の会社から発生していることを示している．調査対象の企業が挙げた中小企業の IT 導入に対するいちばんの障害は資金不足であり，次いで経営者の知識不足や人材不足であった．顧客企業内部の人材育成と併せて IT 導入を行うことが，調査対象の企業から得られた成功する要因として挙げられている．

[1] 図 2.4 の事業収入が 1 億円未満のグラフが高い傾向にあるのは偏りと思われる．この分類の 3 分の 2 の企業が「その他の非製造業」に含まれており，情報処理要員の比率が極端に高く，小規模なソフトウェア開発業や Web サイトデザインの会社がまとまってしまったのではないかと考えられる．

2.3　エンドユーザーによるシステム開発

企業や組織は，IT 化により，業務効率を高めることを求めている．加えて，システム投資によって市場における競争力を高めるという考え方もあり [68]，システム化の有効性は高い．しかしながら，予算と人材の問題がある中小企業ではシステム導入が困難であるという実情がある．

2.3.1　受託開発と内製

ユーザーが利用する業務システムは，ユーザー側が開発会社に発注し，開発会社が契約に基づいて開発を進めて納品する受託開発が 1 つの一般的な手法である．開発部門を持たない企業であれば，システム開発専門業者が進める方が時間や費用の点で効率良く開発できること主要な理由だ．その一方で，出来上がったシステムに対する顧客が不満を持つことも多い．顧客が感じるのは，思っていたものが作られなかった [123] ということと，時間も費用もかかる [130] といった点である．こうした問題点を解消するために，ソフトウェア工学的アプローチでの開発プロセスの改良や，クラウドあるいは新たなビジネスモデルをベースにした開発などが考案され，実施されている．しかしながら，ぎりぎりの予算での開発では期待した効果が得られないこともあり，受託開発は予算が潤沢な大企業が行うシスムテム化の手法であると言える．

一方，業務システム開発を受託モデルではなく，エンドユーザー自身が行う形態がある．こうした動きは「内製」と呼ばれ，成功事例も紹介されており [140]，大手のコンサルティング会社も対応に乗り出している [129] など，開発形態としても認知されている．エンドユーザー企業内に開発あるいはシステム管理部門があるような場合から，現場のスタッフによる開発を行う場合まで，さまざまな状況があり得る．システム部門がある場合には，専門知識を持ったスタッフの雇用が一般的なので，発注先が社内になる違いはあるものの，委託をする点では外注をするのと同じである．エンドユーザー自身の内製としては，他に現場のスタッフが業務の片手間でシステム構築をすることもある．

受託開発を行う会社の業務内容については，いくつかの業界団体などが集約しており，その活動は比較的公開されている．一方，エンドユーザー開発の現場は，個別の企業内で完結していることもあり，個別の開発事例が公開されることはあっても，全体的な市場規模などの全体像はつかみにくい．特に，現場ユーザーレベルでの開発については規模は明確ではない．米国では職業プログラマは 2012 年に 300 万人に満たないのに対して，表計算ソフトやデータベースを業務で利用し，式の設定やクエリー作成を行っているのは 5500 万人に及ぶという予測もされていた [90]．小規模ながらも自身の業務を遂行するためにシステム化を行っているエンドユーザーは相当な数になる．IT 技術者の比率を日本とアメリカに比べると，それぞれ 25%と 72%であることなどから，日本ではアメリカに比べてユーザー企業に所属するエンジニアの比率が低く，その結果，日本では内製される比率が少ないという見方もある [148]．

2.3.2　エンドユーザーによるシステム開発の傾向

企業などの組織内での仕事は千差万別であるが，ほとんどの業務はなんらかの情報共有が行われる．システム化の初期段階では入力と簡単な処理を Excel を中心とした表計算ソフトで行われる．当初は自身の

ための記録として作ったり，あるいは印刷する資料作成用として作られるが，インターネットに接続していることが当然の現在では，メールやさまざまなサービスを使ってファイルの交換が行われる．文書ファイルをメールでやりとりすると，バージョン管理や更新管理が的確に行われなくなるなど，ファイルの置き場所の移動が簡単になった分，最新データにたどり着くことが困難にもなる．そうした問題を解決するには，データベースを利用して，ネットワーク経由の情報共有を図ることになる．Microsoft Access[21] や FileMaker[34] のような製品を使うことも 1 つの方法ではあり，ユーザー自身で開発に乗り出す場合もある．アプリケーション製品は一般にはユーザーメリットを最大限に引き出せるような機能を揃えるが，開発会社が描くビジネスモデルに影響される．例えば，Microsoft Access で作ったデータベースは Mac やスマートフォンで利用できないということにもなる．

しかしながら，FileMaker や Microsoft Access を使うにしても，さまざまなオープンソースの素材を使うとしても，初期開発を完遂するにはエンジニアリングの知識が必要である．筆者も，FileMaker の開発で，エンドユーザーが作ったデータベースを引き継いで完成させることを数多く行ってきたが，ほとんどの場合，スキーマの設計に不備があり，一定の範囲までの機能の組み込みでとどまり，想定したニーズを満たすまでの組み込みができなかったことが容易に見て取れるものであった．しかしながら，適切なスキーマを定義した状態にすれば，エンドユーザーがレイアウトを独自に改良するといったことも見られた．中にはユーザー自身で完成できたシステムもあるとは思われるが，スキーマの設計や，仕様上どうしても手続き的なプログラムを記述しないとできないようなことは，エンジニアが関わって進める必要が出てくる．どこまでをエンジニアが行い，どこからをエンドユーザーが行えるのかは，作成するシステムや組織の形態，エンドユーザーの資質等，その現場特有のさまざまな事情が絡むことになる．こうした状況でシステム化を進めるには，仕事の分担や管理を行う必要がある．それを，受託された側で行う方法や，エンドユーザー側の管理者，あるいは外部のコンサルタントを利用するといった手段が考えられる．

2.3.3 Web システムを業務システムに利用するモチベーション

ビジネス利用の情報機器となると，一時期は Microsoft Windows[22] が事実上「ほぼすべて」と言っていい状態だったが，ここ 5 年の経過を見ても，スマートフォンでの Android[40]，iOS[4] を搭載したデバイスの台頭，さらには Windows ではないデスクトップパソコン向けの OS として，OS X[5] への再評価や Google ChromeOS[42][81] 搭載ノートパソコンの発売など，システム稼働環境は多様化している．Windows で社内のデバイスを揃えるということも 1 つの手段だが，今後出てくるかもしれないいろいろなデバイスへの対応をスムーズにする方法としては，オープンスタンダードを基調としている Web を利用するのが 1 つの方法である．

情報共有のためのサーバーがシステム構築には通常は必要になるが，プロバイダサービスの低価格化やクラウドの進展により，Web ベースのインフラは個人でも簡単に手を出せるくらいの低価格になっている．オープンスタンダード，あるいは評価の高いオープンソースソフトウェアでは，日々進化し，例えば近年に注目されるようになったテスト駆動開発や継続的インテグレーションといった新しい手法も使えるようになる．一方，例えば，FileMaker では単体テストやあるいはレポジトリを利用した開発物の管理という仕組みと統合することはできず，FileMaker 自身にそうした機能が組み込まれるのを待つことになる．Web という

オープンスタンダードをベースにしたオープンソースソフトウェアを利用した開発が，特定の会社の製品に比べると，継続性や発展性，そして柔軟性を持つということもあり，業務システムを Web 技術を利用して構築したいというニーズが発生する．

Web ベースの開発に限ると，サイトのデザインを行う Web デザイナーが比較的初期の段階から関わり，モックアップを作ったり，要求との調整を行うような場合もある．Web デザイナーは，ソフトウェアエンジニアほどのシステム開発の知識があるとは限らないものの，ユーザーの目に触れる部分を作ることもあり，エンドユーザーの立場に近いポジションでもある．また，業務改善やあるいは PC のセットアップなど，IT コンサルタントや特定の業務を外注しているような場合もあり，コンピュータを取り巻く業務のサポートをしている人たちも，顧客であるエンドユーザーの視点を持ち，エンドユーザーの業務と接点がある．本論文で中心的に捉えるエンドユーザーは，システム開発を専業にしている人ではなく，現場のさまざなま業務をこなす人たちである．加えて，Web デザイナーや協力関係にある人も交えて，エンドユーザーとして捉えることにする．

2.3.4　積極的な内製を進める医療業界

エンドユーザー開発が顕在化している業界として，医療業界がある．「日本ユーザーメード医療 IT 研究会（J-SUMMITS）[142]」は，現場の医療従事者による開発を行っている人たちが集まり，開発事例などの研究を進めている．医療現場では電子カルテを始めとしてシステム化の必要性があるものや，あるいは有効性が高い業務が数多くある．一方，業務形態やニーズは病院ごとに異なり，カスタマイズの必要性も高い．J-SUMMITS では，開発結果をカスタマイズしやすい FileMaker 製品を使う医療関係者が多く所属しており，紹介されている事例も FileMaker を使ったシステムが多い．

筆者も，FileMaker を使用した案件で，医療系のエンドユーザーとの開発を経験している．そのうち 2 件はエンドユーザーによって作られたものを発展させる業務であった．ある顧客は，FileMaker のパッケージ製品をカスタマイズした電子カルテを導入後，保守を院内のスタッフが行うことにした．筆者はスタッフへのサポートや，あるいはスタッフの手に負えない部分の開発を請け負う業務を行っている．パッケージ製品の開発会社との保守契約はある模様だが，開発会社とのやりとりだけでは時間もかかり費用もかかるので，内部での保守体制を整えたと説明された．

医療業界でこうしたエンドユーザー開発が進展する理由としては，業務で発生する資料が紙ベースのもので運用されてきており，一定レベルまでの情報化は紙を置き換えることで実現することがある．特に，FileMaker では帳票開発を効率的に行える点から，医療業界で FileMaker の利用が促進されている．また，医療業界は景気に左右されないことや，あるいは福祉分野のように重点的に補助金が投入されるといった事情もあり，他の業界に比べて小さな組織でも予算規模が大きい傾向があり，IT 投資には積極的である．また，医師が組織のトップになることが一般的でもあり，組織全体での取り組みをしやすい環境にある．

2.4 エンドユーザー開発の利点と問題点

現場のユーザーが実際に積極的に開発や保守をおこなったり，あるいは開発プロジェクトで主導的な立場をとるような場合を「エンドユーザー開発」として，分析を行う．エンドユーザー開発の利点と，その利点に対応する問題点などの別の側面を，表 2.1 にまとめた．本項目では，この表に基づいて，エンドユーザー開発の利点と問題点を議論する．

表 2.1: エンドユーザー開発の特徴

	特徴	別の側面
1	予算を抑えられる	人件費に転化される，兼任スタッフが多忙になる
2	保守の即時対応	（同上）
3	現場担当者が取り組める	技術の習得が必要，少ない学習コストで使えるツールが必要
4	現場の知識を活用	システムの抽象化が不十分になる，仕様書不在になりがち
5	要件の明確化	担当者の知識範囲外が考慮されない，将来を見越した設計にならない
6	的確なテストができる	システム的な限界点をテストできない
7	自由にシステム運用ができる	組織から見れば非効率，セキュリティ方針やコンプライアンスに対する違反の可能性

エンドユーザー開発のモチベーションの 1 つとして，予算がないからという消極的な理由も含めて，コスト削減効果への期待がある（表 2.1 の 1）．しかしながら，仮に外注をやめて内製にすることで予算が削減できるとしても，一方で，内製のための人件費負担は増加する．雇用すれば直接人件費に充当されるが，現場スタッフの仕事を増やすことで，既存の業務の効率化を落とす可能性もある．外注から内製への変更は，結果として組織運営や人事に関する問題の解決が，まず必要であると言える．

予算がかけられないとしたら，それは時間がかけられないということにも直結する．こうしたニーズから，短時間であまり労力をかけずに開発を進める動きも見られる [107]．また，日本では 2013 年に経営の変化に迅速に対応するシステム開発を目指す「超高速開発コミュニティ」[131][54] がユーザー企業やベンダーなどを交えて発足している．開発にかける時間が短縮できるのであれば，変化するビジネス現場での要求に対して迅速に対応できるといった利点にもつながる．

仮に内製のスタッフが整えば，保守作業は外注するよりもスピーティーに進めることが期待できる（表 2.1 の 2）．外部への発注となると，見積や契約といった事前の作業に時間がかかる場合もあるが，内製であればそうしたプロセスの多くは一般には省略でき，情報共有のための時間がかからなくなる点はメリットになる．

現場の担当者が取り組むことでのさまざまなメリットが発生する（表 2.1 の 3）．ただし，取り組むことができるようにするには，一定以上の能力を持つスタッフが必要になる．スタッフの教育やあるいは雇用の問題が発生する．ここで望まれているのは，可能な限り学習が容易である点である．

エンドユーザー開発が多大なメリットをもたらすのは，現場の知識をソフトウェア開発に折り込める点である（表 2.1 の 4 および 5）[61]．受託開発の問題点の 1 つは，顧客のニーズを満足させない点があり，要件定義漏れや優先順位の判断の相違などさまざまな理由が挙げられる．ユーザー自身が要件を検討すれば，受託開発における聞き取り調査などのプロセスを経由しなくても要件が抽出できる．しかしながら，抽象化が十分に行われず，目先の機能を中心に要件をまとめると，将来的な変更を見越した開発ができなくなる可

能性もある．また，開発を担当したスタッフの知識の範囲外のことを考慮しなければいけないようなシステムの場合で，その点が考慮されていない場合にはシステムの利用価値が減少する可能性もある．他の部署との折衝やあるいはデータ交換などが必要になると，お互いの要件の擦り合わせなど，受託開発での上流工程に近いことをしなければならなくなる．このような部署間をまたがるような開発は，特定部署のスタッフではなく，横断的に対応ができるシステム部門のような部署がリードしないと開発が進まない可能性もある．

平成 25 年度の情報処理実態調査では，各企業での IT 利活用の実態を示す「ステージ」という指標に関する調査が行われている．「IT の浸透度」「IT マネジメント体制の確立」「IT 投資評価の仕組みと実践」など，6 項目の IT に対する評価基準に対して，ステージ 1〜4 の 4 段階（多いほど高いレベル）で自己評価した結果が公開されている．例えば，「IT 投資評価の仕組みと実践」はステージ 1 は効果を評価していない，ステージ 2 は投資前に予測，ステージ 3 は投資前後の比較と PCDA サイクルの確立，ステージ 4 はポートフォリオとなっている．調査対象を従業員数で分類して，すべての項目に対してそれぞれのどのステージであるかを評価した結果の割合を図 2.5 に示した．IT 活用のレベルは，従業員数が多い企業ほど，より高い傾向にあると言える．従業員数が多い大きな企業ほど，管理を厳重に行っているという点と，外注比率が高まりプロの開発者による高いレベルのゴールを実現していることの現れであると考えられる．このことは，小さな組織での内製では IT 活用レベルが高くならない可能性があることを示唆する．

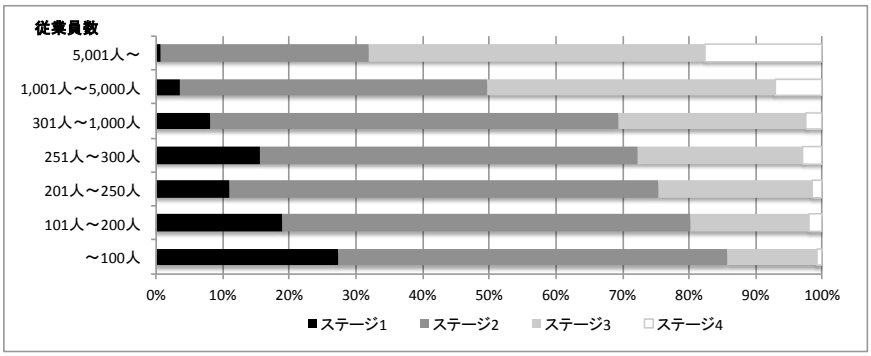

図 2.5: IT 化実現のステージ [120]

開発したシステムのテストについては，エンドユーザーであれば，実際に発生しそうな状況を想定して，テストにかかることができる（表 2.1 の 6）．受託開発では，テストデータやあるいは業務を想定して行うことに比べると，効率は良い．しかしながら，システムの限界のテストができない点や，自身は想定していなくても他の利用者では発生するような状況をカバーしていないなど，テスト範囲が不十分になることも考えられる．

規模の大きな会社でも，単一の組織では予算の問題で，部署内の内製をする場合もありうるが，そのような場合は，システム運用の問題が発生する（表 2.1 の 7）．運用機材のコストも安くなるものの，全社でそうした運用を行うのは効率が悪くなり，セキュリティの確保や，ビジネス上のコンプライアンスを守っていない状況が発生しやすくなる．気軽にサーバーを立てられるとしても，運用上許されない場合はむしろ一般的になっていると言える．

エンドユーザー開発には利点もあるが，同時にそのことは問題点でもある．利点を活かして問題点が噴出しないようにするには，エンドユーザー側の組織や管理の点での配慮と，学習が容易で汎用性のある使い

勝手の良いツールの存在が重要と考えられる．組織運用上の考慮を十分にした上で，技術的なサポートによる解決が必要である．

開発をしようとしているエンドユーザー組織は，開発そのもののマネジメントやプロジェクト進行を進め，必要な人材や予算の投下，従業員の業務バランスなどを考慮した管理体制が必要となる．予算が限られているのですべてを内製するという方法もあるが，外部のリソースを的確に利用することで，いくつかの問題は解決させることができる．例えば，コンサルタントに依頼して，実装やテストプランを検討することで，プロの視点や大局的な視点が入り，大きな問題の発生を防ぐこともできる．また，分析などを含む上流工程についても，より的確に指摘を受けることができる．

2.5 エンドユーザー開発に対する提案

予算規模の少ない抽象企業や組織が INTER-Mediator を使用してシステム開発に利用することを提案する．宣言的な記述での開発が可能な INTER-Mediator は，第 4 章に示すように，保守の作業を主として宣言的な記述で行うことができる．第 6 章での実験結果で示すように，学習可能性は十分にあることから，エンドユーザー向けツールに要求される学習のしやすさを備えていると言える．初期開発と同程度の費用がかかる保守をエンドユーザーが行うことで，直接経費のコストダウンが可能であり，予算が潤沢でなくても，開発したシステムをビジネスの変化に追随できるような継続的な進化を実現する．

保守作業を中心にエンドユーザーが開発に参画することでの経費節減効果を，経済産業省が実施した平成 25 年度の「情報処理実態調査 [120]」の結果から推測する．ソフトウェア投資金額の回答をもとに，全業種に対して 1 社あたりの平均金額を求めた．従業員数が 100 人以下の企業では平均して年間 140 万円，100〜300 人規模の会社では 420 万円のソフトウェア投資が行われている [2]．ソフトウェア投資にはライセンス料等も含まれていることを考えて約半分が受託開発費用だとし，さらにその半分が保守にかかる費用だとすれば，従業員が 100 人以下の企業で年間 35 万円，100〜300 人の企業で 105 万円の経費節減効果があると試算できる．

エンドユーザーは良いツールを常に探している．ここでの「良い」の評価基準は，学習が容易である点がまず挙げられる．すべてのことを学習しないでも使い始めることができることや，少ない作業で開発ができる点が求められている．また，エンドユーザーの中には手続き的プログラミングに対して拒否反応を起こす場合もあり,「手続き的プログラミングをしない」という点を魅力に感じるユーザーもいる．

INTER-Mediator による開発方法を学習するための前提知識は，データベースについての概念的な理解と HTML によるページの記述である．加えて，サーバーで動く動的な Web アプリケーションについての概念的な理解が必要である．これらの知識があれば，宣言的な手法によるいくつかの記述ルールを理解することで，Web アプリケーションの開発や保守ができる．手続き的なプログラミング言語や，あるいはその上で稼働するフレームワークを学習する必要があるとすると，心理的な障壁があり，加えて実際に学習にかかる時間もかかる．INTER-Mediator においては，宣言的な手法として，HTML の範囲で記述できる点は，心理的な障壁を低くし，学習時間の短縮にも貢献する．INTER-Mediator でのシステム開発や保守に求めら

[2] ソフトウェア投資金額は全回答社のうち約半分しか回答していない．ここでは未回答の企業は投資が 0 であるとみなして平均値を求めた．

れている前提知識は，特殊な技術ではなく，一般的な知識であり，学習素材は簡単に手に入る．また，すでに学習ができている人たちも少なくはない．また，まったく知識がない人でも，例えば HTML の学習は INTER-Mediator による業務システムを作る以外の用途にも転用が効く知識であることから，学習するモチベーションも上がる．

現実の開発作業の上では，開発に携わるエンドユーザーは同じ結果が得られるのであれば，より作業負荷が軽い手法を好む．INTER-Mediator は相対的なコード記述量が，手続き的なプログラミングを主体としたフレームワークに比べて約半分になることを 3.5 節において示す．一般的な Web アプリケーションフレームワークよりも少ない記述でシステムの構築ができる点でも，高い開発効率を求めるエンドユーザー向けのツールとしての要件を満たす．

第3章　INTER-Mediatorの概要

本章では，INTER-Mediator のアーキテクチャや機能の概要を解説する．そして，資産の貸出履歴を管理する簡単なアプリケーションを通じて，INTER-Mediator を利用した Web アプリケーションの作成方法を説明する．そして，INTER-Mediator に近い仕組みの他のソフトウェアやサービスとの比較を行う．

3.1　フレームワークの目的とアーキテクチャ

INTER-Mediator はデータベースと連動した Web ページを宣言的な記述を行うだけで作成できる Web アプリケーションフレームワークである．2010 年初頭より MIT License として配布しており，GitHub 上にレポジトリを公開してオープンソースとして開発を進めている．中小企業や小さな組織が情報共有を中心とした業務システムの構築と保守を低コストで実現することを目指して開発を進めて来た．

3.1.1　フレームワークの利用者

INTER-Mediator は，実務担当者などのエンドユーザーによる保守の実現を目指している．ここで想定しているエンドユーザーは，手続き的プログラミングに関する知識はないものの，HTML や CSS についての知識はあり，ページ上に表示する部品（ウィジェット）の変更やその書式の改変を自力でできるようなユーザーを示す．Web 制作現場でのデザイナーも，対象者と想定している．

システムを新規開発する場合，ドメイン分析に基づくデータベースのスキーマ設計など，専門的な知識が必要な作業が必要になり，リリースするまでの開発は主に専門家が行う必要がある．その後，運用中に見つかった改訂などを含めた保守作業をエンドユーザーが行うことを本論文では想定している．

INTER-Mediator は筆者の 1 人が受注したいくつかのシステムでの稼働実績がある．2014 年 10 月の時点で，看護師の勤務予定を作成するシステム，オンデマンド印刷を受け付けるサイト，海外出張の承認システムなどがある．本フレームワークの利用実績については，7.3 節で解説する．

3.1.2　アーキテクチャ

INTER-Mediator を利用して開発される Web アプリケーションのアーキテクチャを図 3.1 に示した．INTER-Mediator はサーバーサイドで稼働するソフトウェアと，クライアントサイドで稼働するものに分類される．図中,「拡張」コンポーネントの部分は手続き的なプログラミングにより独自に機能を追加できる箇所である．サーバーサイドで稼働するモジュールと，クライアント側にダウンロードして実行されるモジュールに分かれており，これらが通信を行ってデータベースから取り出したり，データベースへ書き込みを行う．

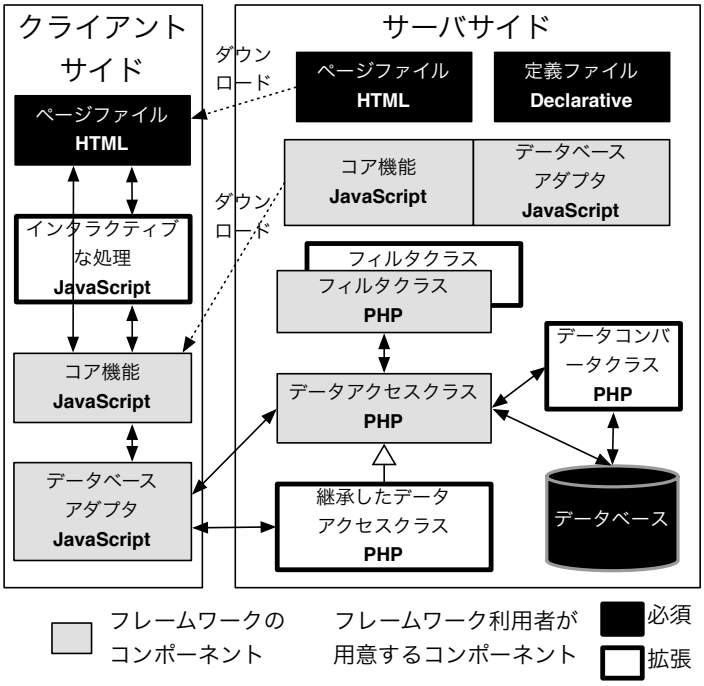

図 3.1: INTER-Mediator のアーキテクチャ

開発において必ず用意するものとして,「データベース」,「ページファイル」,「定義ファイル」がある．データベースは RDB 等を用意しスキーマを適用しておく.「ページファイル」とは，HTML で記述した Web ページのテンプレートである.「定義ファイル」には宣言的な記述でデータベースの利用方法などを記述する．ページファイルと定義ファイルの構成については，3.4 節で詳しく解説する．

Ver.4.5 現在で対応しているデータベースエンジンには，MySQL や PostgreSQL などの PDO（PHP Data Object[97]）に対応した RDB，あるいは FileMaker Server[33] を利用できる．PDO に対応していれば原則として利用できるが，テストケースを定義しているのは，MySQL，PostgreSQL，SQLite の 3 つのオープンソースデータベースである．Web サーバーには PHP Ver.5.2 以上が稼働する環境が必要である．クライアントとなる Web ブラウザについては，HTML5 対応ブラウザでの利用を原則としている．Internet Explorer については，Ver.8 以降で利用できる．

3.2　機能概要

INTER-Mediator の機能概要を示すために，Web アプリケーションで組み込まれる機能について，表示，編集，インタラクティブ，その他の 4 つに分類した結果を表 3.1 に示す.「表示」は主にデータベースの内容をページ上に展開する機能である.「編集」はページ上で入力や変更したデータをデータベース側に反映させる機能である.「インタラクティブ」はフレームワークに自動的に追加されるユーザーインターフェースをまとめた.「その他」には認証などを記載した．それぞれの分類に関して INTER-Mediator で宣言的な記述だけで利用できる機能と，プログラムを記述することで実現する機能を表 3.1 に示した．

JavaScript 向けには多数のライブラリがあり，それらが混在することで同一名関数がそれぞれのライブラ

表 3.1: INTER-Mediator の機能概要（Ver.4.6 現在）

機能分類	宣言的な記述で実現	手続き的プログラム記述が必要
表示 (データベース アクセス)	• テーブルやビューの指定 • 検索条件の指定 • ソート条件の指定 • リレーションシップに基づく検索	• サーバーサイドでデータベースへの要求や応答を変更
(バインディング)	• データベースのフィールドに関して – データを要素内に表示 – データを変更して要素内に表示 – データをフォーム要素に設定 – データを要素の属性やスタイル属性に設定 – 標準で用意された JavaScript コンポーネントとの連携 • 複数のレコードを繰り返して表示 • リレーションによって得られたレコードの表示 • フィールドから求めた計算結果の表示やスタイルへの適用 • 独立したキー/バリュー形式のデータ保存機能に対するバインディング	• ノード生成時での割り込み処理 • 標準で用意されていない変換処理 • 標準で用意されていない JavaScript コンポーネントの利用
編集	• ユーザーによる編集結果をフィールドに書き戻す • 編集結果を変換して書き戻す • 楽観的ロックに基づくデータベース更新のマルチユーザー対応 • 値の検証 • 同一ページ内での同一フィールドの変更結果を同期 • 変更したフィールドとリレーションが設定された他のフィールドの更新 • 値の変更，レコード作成，レコード削除を他のクライアントに通知して更新	• サーバーサイドでデータベースへの要求や応答を変更 • 標準で用意されていない変換処理 • フォーム要素以外のクリック等に対する処理
インタラクティブ	• レコード作成ボタン • レコード削除ボタン • 一定数のレコードごとの表示と表示範囲移動のコントロール • 新規レコード作成専用ページ • ファイルのアップロード • 一覧と詳細を行き来するユーザーインタフェース • 検索条件設定，並べ替え基準となるフィールド指定のためのユーザーインタフェース	• 入力結果やユーザー操作に応じた複雑な処理 • サーバーサイドでデータベースへの要求や応答を変更
その他	• 画像などのメディアファイルへの代理アクセス • 認証・認可およびユーザーインターフェース • パスワードの変更 • 画像などのメディアアクセスへの認証の適用 • データベース処理後にメール送信	• PDF 生成などのプログラムを経由したファイルの作成

リに存在するような不具合も発生する．INTER-Mediator では識別子に他のライブラリで使っていないようなものを使用し，他のライブラリとの混在を想定した設計とした．INTER-Mediator では各種ライブラリとのインターフェースを取る仕組みも備えている．ライブラリをラッピングするオブジェクトを定義し，決められた名前のセッタやゲッタ，初期化メソッドを実装することで，データベースと連携しての利用もできる．Moodle 等でも利用されている HTML エディタの TinyMCE[98] や，ソースコードエディタの CodeMirror[49] を使用した例をサンプルとして製品に含めている．

エンドユーザー向けの Web 開発ツールを構築するために Web アプリケーションが持つべき機能が調査されている [86][88]．その調査ではエンドユーザーのニーズに合うような 61 のサイトを評価した結果から，表 3.2 の 1 列目に示された機能を挙げている．それぞれの機能について INTER-Mediator での対応を 3 列目に記述した．ほぼすべての機能項目を INTER-Mediator はサポートし，多くは宣言的な記述で実現している．なお,「Menu」については HTML での構築あるいはライブラリの利用を想定しており，INTER-Mediator では直接のサポートはない.「Data persistence」についてはセッションのサポート等が考えられるが，1 ページに機能をまとめて埋め込む作成方法が一般的であることや，必要ならデータベースを利用できるので，現状ではサポートをしていない．

3.3　サンプルプログラムの概要

フレームワークを使った Web アプリケーション作成を解説するために，サンプルとして社内の共有物などについて貸し出し管理や一覧作成を行う「資産管理」アプリケーションの例を示す．本サンプルは INTER-Mediator のソースに含めて配布している．スキーマには，資産（asset），貸出履歴（rent），社員名簿（staff）といった 3 つのエンティティだけがあり，それらはテーブルとしてデータベース上に定義した．図 3.2 は「資産一覧」が表示されている．データベースの資産テーブルにあるレコードが表示されている．複数のレコードが繰り返され，それぞれのレコードのフィールドが表示されている．一定数のレコードごとに表示するインタラクティブ要素がページ最上部に表示されている．リストの右側には「削除」ボタンがあり，対象レコードの削除が行える．このテーブルに新しいレコードを作成するには，画面上部の「レコード追加: asset」表示された部分をクリックする．

図 3.2 のリストの左側には「詳細」ボタンが見えている．このボタンをクリックすると，該当するレコードに関して個別の資産に関する貸出履歴を表示するさらに詳しい「資産詳細」が図 3.3 のように表示される．テキストフィールドなどのフォーム要素や，貸出履歴として関連レコードの繰り返し表示が行われている．ここで貸出履歴の追加やメモの記述が行える．テキストフィールドの内容は変更でき，変更するとデータベース側のフィールドのデータも更新する．1 つの「資産」に対して通常は「貸出履歴」は 0〜複数個あり，こうした関係はリレーションシップによって規定できる．このページには該当する資産に対する貸出履歴のみが一覧表示される.「一覧表示」ボタンをクリックすると，図 3.2 の画面に戻る．

これら 2 つの画面は，アプリケーションでよく利用される典型的な形式で，マスター/ディテール形式などと呼ばれている．前者を「一覧ページ」，後者を「詳細ページ」と呼び，データのあらましを一覧表示で見せて，そこでユーザーが選択した 1 つの項目に対するさまざまなデータを詳細ページで表示する．2 つのレイアウトがあるような動作をしているが，実際には 1 つの HTML ファイルに 2 つのレイアウト要素を定義

表 3.2: Web アプリケーションが持つべき機能との対応 [86]

列挙された機能	概要	INTER-Mediator での対応 （★は手続的プログラムが必要）
Static text, Checkbox, Radiobutton, Combobox, Listbox, Textfield, Link	機能記載の通り	HTML で実現可能，コンボボックス以外はバインド可能．引用元のリストではポップアップメニューの記載がないが，これについても対応
Dynamic Output	変化する値の扱い	バインディングによりデータベースと連動
Image	アプリケーションが管理する画像	サーバー上のファイルを利用可能
Button	なんらかの処理を呼び出す	表示は HTML で可能，★動作は JavaScript で記述するが，一部は宣言的な記述で可能
Menu	階層的あるいはフラットな機能呼び出し	（特にサポートはない）
Scrollable cotext	大量のデータを部分ごとに表示	ブラウザあるいは HTML で対応
Visual properties	スタイルをデータに依存して変更	属性や CSS 項目へのバインドや計算結果の適用
Decorative elements	ラインやボックスなどの要素	HTML で対応
Screen	一時的に見せる画面	HTML で対応
Data persistence	別ページで利用するなどのデータ保持	セッションのサポートはなく，データベースを経由するか★ URL パラメータ等を利用
Data manipulation	計算や文字列処理など	計算フィールドで式や関数を記述可能
Variable	フォームの値など	★モデルへのアクセスで利用可能
Conditions	状況に応じた条件付きの動作	計算プロパティやバインディングで可能
Input Validation	入力データの検証	指定により可能
Authentication	認証	設定とユーザーデータベースにより可能
Authorization	認可	設定とユーザーデータベースにより可能
Table data output & Repeating section	繰り返し表示	エンクロージャー/リピーターの仕組みで可能
Table sort	特定の列で並べ替え	コンテキストの指定や，★動的な指定
Table summary	集計などの処理	計算プロパティ，★サーバーサイドの拡張可能
Table nesting	テーブル内にテーブルを展開	
Database	データの永続化	データベースエンジンを利用
Record	レコードとしての扱い	モデルで対応している
Recordset	レコードの集合	モデルで対応している
Master-Detail Relation	一覧と詳細のページ	コンテキストでの指定により可能
Recordset Browsing	大量のデータを部分ごとに表示	ページネーションをサポート
Recordset Filter & Search	クエリーでの検索条件	コンテキストの指定や，★動的な指定
Add record	テーブルへのレコード追加	コンテキストの指定によりボタンを生成
Delete record	テーブルからレコードを削除	コンテキストの指定によりボタンを生成
Messaging	電子メールによる通知	データベース処理時にメールを送信可能
File upload	ファイルをアップロード	標準添付の JavaScript コンポーネントで対応可能

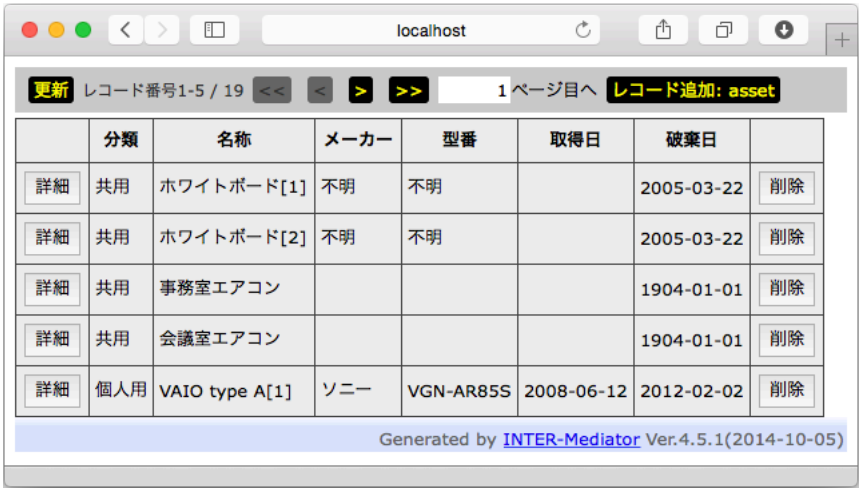

図 3.2: 資産管理アプリケーションの「資産一覧」

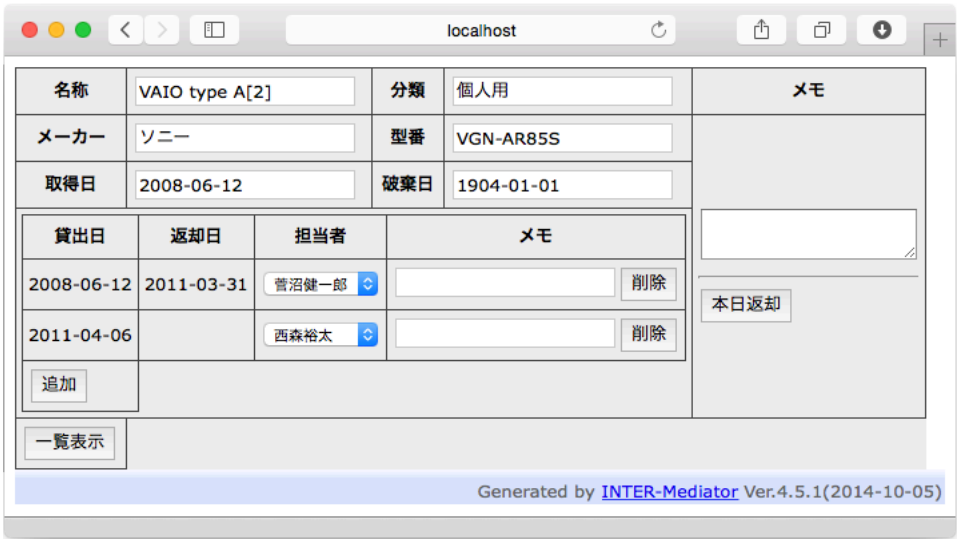

図 3.3: 資産管理アプリケーションの「資産詳細」

し，この場合はそれぞれを表示/非表示を制御して，切り替わって見えるようにしている．INTER-Mediator ではこうした形式のユーザーインタフェースも宣言的な記述のみで開発できる．

INTER-Mediator では，表 3.1 の「プログラム記述が必要」と記載された機能を実現するような場合には，手続き的なプログラミングが必要となる．図 3.3 のページでは，「本日返却」ボタンを押したときに今日の日付を貸出履歴の最後のレコードに設定する機能に関しては数行ほどの JavaScript のプログラムを記述した．また，図 3.2 では，フレームワークの呼び出しには 1 ステートメントのみの JavaScript プログラムを記述した．これら以外には手続き的なプログラムの記述は行っていない．

3.4　データベース連動ページ作成

INTER-Mediator での Web アプリケーション作成に必要な「定義ファイル」と「ページファイル」に記述すべき内容を，資産管理アプリケーションでの記述例を用いて説明する．「定義ファイル」は PHP で記述す

るが，配列を，決められたキーとアプリケーションの動作に合わせた値で記述するだけで，クラス定義や制御構造を交えた手続き的なプログラミングを行う必要はない.「ページファイル」は HTML で既定された範囲のタグと属性のみで記述する．一般的なフレームワークではテンプレートのための拡張タグを用いることがあり，テンプレート単体で仕上がり具合をブラウザで確認できない場合があるといった問題点がある．これら 2 種類のファイルでは手続き的なプログラミングは必要なく，宣言的な記述のみである．

3.4.1 定義ファイルの作成

リスト 3.1 は資産管理アプリケーションで利用する定義ファイルである（ファイル名は「asset_context.php」）．この定義ファイルでは，フレームワークの読み込みの記述と，フレームワークが用意する IM_Entry 関数の呼び出しが必要である．最初の行はフレームワークのプログラムを利用するための読み込みで，ファイルのディレクトリ配置に依存するが，フレームワークのフォルダを参照できればよい．そして IM_Entry 関数の引数にフレームワークが必要とする情報を配列の形式で指定する．呼び出す関数は IM_Entry 関数と常に決まっているので，実質的に開発案件ごとの違いがあるのは，配列で指定する部分だけである．

リスト 3.1: 定義ファイルの例（asset_context.php）

```
require_once('../INTER-Mediator.php');

IM_Entry( array(
   array(    //一覧領域用のコンテキスト定義（一覧ページ）
      'name'=>'asset',  //ページファイルからの参照名
      'key'=>'asset_id',    //キーフィールド名
      'repeat-control'=>'insert delete',  //挿入，削除ボタンを設置
      'records'=>5,     //5レコードずつ表示する
      'paging'=>true,   //一定数のレコードごとに表示
      'sort'=>array(        //並べ替えのフィールドと方向
         array('field'=>'purchase', 'direction'=>'ASC'), // purchaseフィールドを昇順で並べる
      ),
      'navi-control'=>'master-hide',       //このコンテキストが一覧ページのものであることを示している
               //この設定により，詳細ページにジャンプするボタンを配置する
      'default-values'=>array(       //レコード作成時の初期値．今日の日付が設定される
         array('field'=>'purchase', 'value'=> IM_TODAY),
      ),
   array(    //詳細領域表示用のコンテキスト定義（詳細表示）
      'name' => 'assetdetail',    //ページファイルからの参照名
      'view'=>'asset',  //データベース側のテーブル名（参照時）
      'table' => 'asset',   //データベース側のテーブル名（更新時）
      'records' => 1,       //詳細なので，1レコードのみ
      'key' => 'asset_id', //キーフィールド名
      'navi-control' => 'detail-bottom',   //詳細ページであること示す
        ),              //一覧に戻るボタンを下部に追加する
   array(    //1つの資産についての貸出履歴のコンテキスト
      'name' => 'rent', //ページファイルからの参照名で，データベース上でもrentテーブル
      'key' => 'rent_id',   //キーフィールド名
      'sort' => array(  //貸出日フィールドの昇順でソートする
         array('field' => 'rentdate', 'direction' => 'ASC'),
      ),
      'relation' => array( //リレーションシップについての定義．外部キー，親コンテキストの対応するキー，演算子を指定
         array('foreign-key' => 'asset_id', 'join-field'=> 'asset_id', 'operator' => '='),
      ),
      'repeat-control'=>'insert delete',   //追加ボタンと削除ボタンを配置する
      'default-values'=>array(    //レコード作成時の初期値．今日の日付が設定される
         array('field'=>'rentdate', 'value'=> IM_TODAY),
      )
   ),
   array(    //担当者のポップアップの選択肢を取り出すコンテキスト．
      'name' => 'staff',    //ページファイルからの参照名で，データベース上でもstaffテーブル
```

```
42                  //常に全てのレコードを表示するので，検索やリレーションシップの設定はない
43          ),
44      ),
45      NULL,
46      array( /*データベース接続情報，詳細は省略*/ )
47  );
```

「資産一覧」はリスト 3.1 の最初の方に「一覧領域用のコンテキスト定義（一覧ページ）」コメントが描かれた行以降で定義された配列を利用してデータベースから情報を取り出し，レイアウトに対する指定等も含まれる．それぞれの設定の概要はリスト内にコメントで示した．この中の name キーの値によって，データベース内の asset テーブルを利用することが記述されている．加えて sort キーに記述されている通り，データベースからデータを取り出すときに purchase フィールドの値で昇順での並べ替えを行うといったクエリーのための付加的な情報も記述している．配列のキーは INTER-Mediator で独自に定義したものであり，さまざまな動作を定義するために必要に応じて記述する．この 1 つの配列を「コンテキスト」と呼ぶ．この定義ファイルには 4 つのコンテキストがあり，以後はそれらを name キーに対する値で呼ぶことにする（例えば最初のコンテキストは「asset コンテキスト」）．

実装上は定義ファイルで定義された「コンテキスト定義」と，この定義を元にして構築されたモデルオブジェクトしての「コンテキストモデル」を区別する必要があるが，宣言的な記述の範囲内ではまとめて「コンテキスト」と示すことでも大きな問題はないと考える．第 5 章での議論の一部はこれらを分離して検討する必要があるが，その他の部分では，定義ファイルの記述や内部で実体化したものを総称して「コンテキスト」と呼ぶ．

3.4.2　Web アプリケーションで供給される定義ファイルエディタ

定義ファイルは最終的には PHP 言語で記述される必要があるが，利用者が利用する上では，キーと値で記述できれば必要な情報を含めることができる．例えば，XML で定義ファイルを記述したとしても，最終的には PHP の連想配列に変換すれば，定義ファイルとして利用できる．具体的に示すために本論文中では PHP 言語で記述したものを示している．ただし，PHP そのものをプログラマではない人が記述すると，カンマやカッコの過不足を補うような作業が発生したときに作業が困難になる．開発ツール上での文法チェック等を活用すれば不可能ではないものの，開発作業を難しくしてしまう．そこで，PHP 言語を習得しなくても定義ファイルの編集ができるように，図 3.4 のような Web アプリケーション形式の「定義ファイルエディタ」を開発し，フレームワークに含めて配布している．この点を含めて，PHP 言語で記述してはいるが，手続き的なプログラミングが必要な記述を必須としておらず，実質的には宣言的な記述と言える．

3.4.3　ページファイルの作成

資産管理アプリケーションのページファイルをリスト 3.2 に示す．前述の通り，2 つのレイアウトをこの 1 つのファイルで実現している．BODY タグ内に 2 つの TABLE タグ要素があり，前の方が一覧ページ用，後の方が詳細ページ用のレイアウトである．ページファイルでの記述内容について，図中の番号と対比させて解説する．また，リスト内の#以降の文字列はコメントである．

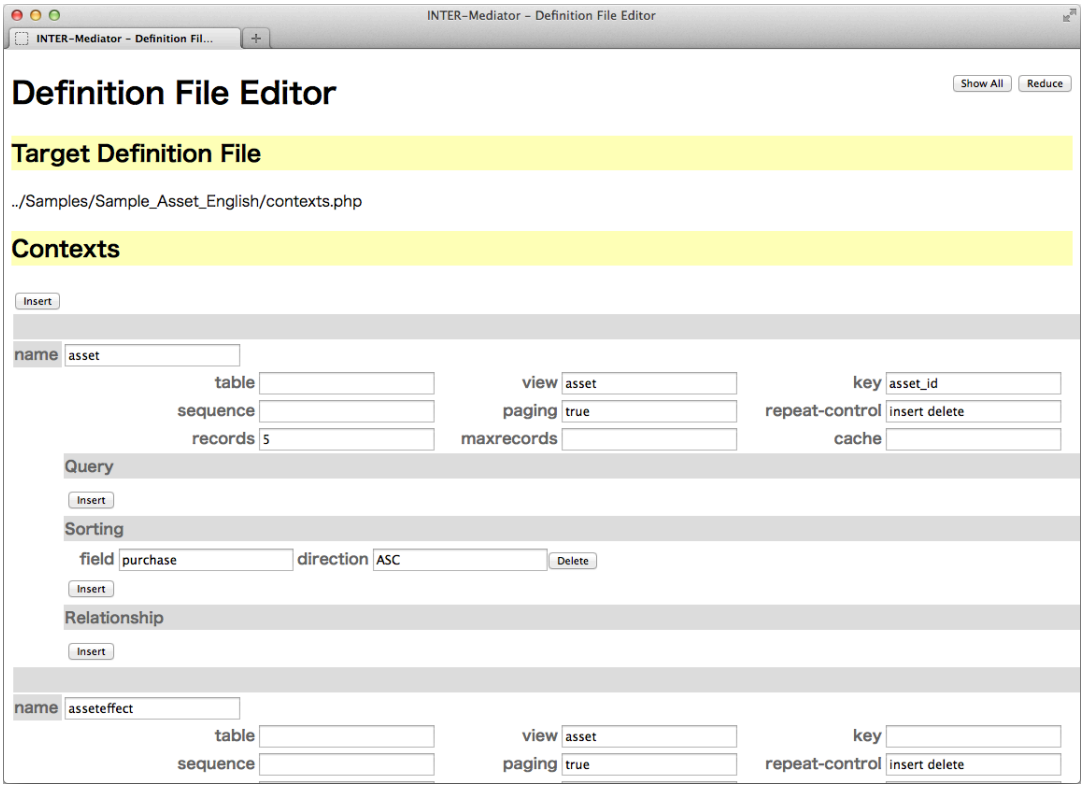

図 3.4: 定義ファイルエディタ（主要な項目のみを表示）

リスト 3.2: 「資産一覧」のページファイルの例

```
<html>
<head>
    ①<script src="asset_contexts.php"></script>
    <script>
        ⑧function setBackDate(id)  {
            var context = IMLibContextPool.getContextFromName("rent")[0];
            context.setDataAtLastRecord('backdate', generateToday());
        }
        function generateToday()    { .... } #今日の日付の文字列を返す関数（内容は省略）
    </script>
</head>
<body onload="INTERMediator.construct()">③
<div id="IM_NAVIGATOR"></div>④
<table>  #一覧ページのレイアウト領域．asset コンテキストを利用している
    <thead><tr><th></th><th>分類</th><th>名称</th><th>メーカー</th>
            <th>型番</th><th>取得日</th><th>破棄日</th><th></th><th></tr></thead>
    <tbody>
    <tr>
        <td></td>   #ここに自動的にディテール領域を表示するボタンが挿入される
        ②<td data-im="asset@category"></td><td data-im="asset@name"></td>
        <td data-im="asset@manifacture"></td><td data-im="asset@productinfo"></td>
        <td data-im="asset@purchase"></td><td data-im="asset@discard"></td>
        <td></td>   #ここに自動的にレコード削除ボタンが挿入される
    </tr>
    </tbody>
</table>

<table>  #詳細ページのレイアウト領域．最上位階層では，assetdetail コンテキストを利用する
    <tbody>
        <tr>  <th>名称</th><td><input type="text" data-im="assetdetail@name"/></td>
            <th>分類</th><td><input type="text" data-im="assetdetail@category"/></td>
            <th>メモ</th>
        </tr>
```

```
34      <tr>  <th>メーカー</th>
35        <td>②<input type="text" data-im="assetdetail@manifacture"/></td>
36        <th>型番</th><td><input type="text" data-im="assetdetail@productinfo"/></td>
37        <td rowspan="4">
38          <textarea data-im="assetdetail@memo"></textarea><hr>
39          ⑦<button onclick="setBackDate()">本日返却</button>
40        </td>
41      </tr>
42      <tr>  <th>取得日</th><td><input type="text" data-im="assetdetail@purchase"/></td>
43        <th>破棄日</th><td><input type="text" data-im="assetdetail@discard"/></td>
44      </tr>
45      <tr>  <td colspan="4">
46        <table><thead><tr><th>貸出日</th><th>返却日</th><th>担当者</th><th>メモ</th></tr></thead>
47          <tbody><tr>
48            ⑤<td data-im="rent@rentdate"></td><td data-im="rent@backdate"></td>
49              <td>  ⑥<select data-im="rent@staff_id">
50                    <option data-im="staff@staff_id@value staff@name"></option>
51                </select>
52              </td>
53              <td><input type="text" data-im="rent@memo"/></td>
54          </tr></tbody>
55        </table>
56      </td></tr>
57    </tbody>
58  </table>
59  </body>
60  </html>
```

以下，リスト 3.2 に記述した内容を解説するが，どういった仕組みで実現しているかについては，第 5 章において詳細に解説を行う．

①ヘッダ部の SCRIPT タグの要素で，リスト 3.1 の定義ファイルを読み込む部分があり，これによりページにフレームワークが組み込まれる．定義ファイルにアクセスすることで得られるのは JavaScript のプログラムであるが，固定的なプログラムだけでなく，一部はサーバー側でフレームワークによって動的にプログラムを生成している．これらを取り込んだ JavaScript のプログラムを利用して，テンプレート処理や更新処理を行っている．

リスト 3.2 の 20 行目にある②は一覧ページ内にあり，TD タグ内に「data-im="asset@category"」といった記述が見られる．ここでは，定義ファイル側でのコンテキストを参照する名前（name キーの値）と，データベース側のフィールド名を，@で区切って記述する．この指定のある要素を「リンクノード」と呼び，data-im 属性を「ターゲット指定」と呼ぶ．INTER-Mediator がテンプレート処理をするとき，この要素に対して，asset コンテキスト，つまり asset テーブルにある category フィールドの値を挿入する．この場合は，TD タグの間に TD タグ要素のテキスト子要素として，フィールドの値が追加され，その結果，フィールドの値がテーブルのセルに見えるようになる．ここでは TR タグ要素が 1 つだけの 1 行分の表であるが，レコードが複数あれば，レコードの数だけ TR タグ要素を複製してレコードが一覧されて見える．

一方，リスト 3.2 の 35 行目にある②は，INPUT タグ内に「assetdetail@manifacture」といったターゲット指定がある．assetdetail はリスト 3.1 の定義ファイルを見れば，表示，更新ともに asset テーブルを利用し，manifacture フィールドの値がテキストフィールドに表示される．一覧ページは asset，詳細ページは assetdatail と異なる名前をつけるようにした．asset コンテキストの records は 5 であり，5 レコードずつ表示される．一方，assetdatail コンテキストの records キーの値は 1 であり，1 レコードずつ表示されることを想定している．また，navi-control キーの値は asset コンテキストは「master-hide」，assetdatail コンテキ

ストは「detail-bottom」であり，前者が一覧ページ，後者が詳細ページとして機能するような指示がある．master-hide の場合は初期状態では詳細ページを非表示にする．detail-bottom により詳細ページの下部に一覧ページに戻るボタンを付与する．navi-control キーの値によって，マスター/ディテール形式のユーザーインタフェースが実現している．

③テンプレート処理を始めるには,「INTERMediator.construct()」という JavaScript の呼び出しを行う．このページでは，BODY 要素の onload 属性で記述したので，ページファイルの内容をブラウザが読み出した直後にテンプレート処理が行われることになる．スクリプト処理として完全に分離したい場合は，SCRIPT タグ内やあるいは別途拡張子が.js のファイルに「window.onload = function()INTERMediator.construct();」といった記述をしてもよい．

④に id 属性が「IM_NAVIGATOR」の要素がある．これにより，図 3.2 に見られるような，5 レコードずつ表示しつつ，前後のページへ移動するためのユーザーインターフェース（ページネーション）が，フレームワークによって自動的に挿入される．ページネーションは，リスト 3.1 の定義ファイルで paging キーの値が true に設定されているコンテキストにのみ作用する．

図 3.2 では，1 行ごとに「削除」ボタンがあり，画面上部には「レコード追加 asset」というリンクがある．これらは，リスト 3.1 の定義ファイルにある repeat-control キーによって指示でき，フレームワークによって自動的に挿入される．asset コンテキストでは「insert delete」という値になっており，挿入ボタン，削除ボタンのいずれも作成される．

⑤の部分は，TABLE タグ内の 1 つのセル内にさらに TABLE タグがあるといった構成になっている．このように，リピーター内にさらに別のエンクロージャーがあるような場合，内部のエンクロージャーを解析して，この場合は rent というコンテキストを使っているということを判別する．rent コンテキストには relation キーの値があるため，ここでは，上位の assetdatail コンテキストの現在のレコードと，下位の rent コンテキストにあるいくつかのレコードが関連することをが定義されている．ここで，rent コンテキストの定義に従って rent テーブルから値を取り出すが，外部キー値を上位のコンテキストから取り出すことにより，関連レコードを取り出すクエリーを自動的に生成する．

⑥の部分は，さらに rent コンテキストの中に，staff コンテキストを内包している．staff コンテキストは relation がないため，特に関連レコードは関係なく，この場合はすべてのレコードを取り出して，OPTION タグに設定することで，ポップアップメニューにはスタッフの氏名が並ぶようになっている．

「本日返却」のボタンは⑦の部分で記述されている．ここでは，ページヘッダ内の⑧の部分のプログラムを呼び出している．汎用的な機能は宣言的な記述で可能な限り対処できるように機能を用意しているが，フレームワークに存在しない機能はどうしても JavaScript を利用したプログラムが必要である．ここで，貸出履歴に対して「本日返却」ということは，貸し借りの情報の最後のレコードの返却日に今日の日付を入力することであり，言い換えれば，ここでの rent コンテキストで得られたレコード群の最後のレコードの backdate フィールドに今日の日付を入力すればよい．関数に記述されたプログラムは 2 行だけで，コンテキスト名からフレームワークが管理しているモデルを得て，そのモデルの最後のレコードの指定フィールドの値を更新するメソッドを利用している．

3.4.4　本論文での「宣言的」の意味

一般用語としての「プログラム」「プログラミング」「プログラミング言語」は，状況によって示す範囲が異なる場合がある．本論文では，関連研究の論文 [63] に従い，プログラミング言語は，手続き的なもの（Imperative）と宣言的なもの（Declarative）にまず大きく分類され，宣言的なものとして，SQL，HTML，CSS などがあるという立場である．一般には，HTML は「マークアップ言語」であり,「プログラミング言語」ではないという立場を取る．一方，プログラミングはコンピュータに対して指示を与えるという広義の観点からすれば，HTML や XML などのマークアップ言語も，レイアウトの指示やデータ構造を記述できる点ではプログラミングを行っていると主張はできる．このように，用語の混乱があるので，本論文では「手続き的なプログラミング」と「宣言的な記述」と表現することにした．

3.4.1 項に説明した定義ファイルは，記述言語自体は PHP ではあるが，指定するのは連想配列と決められた関数呼び出しのみである．PHP 言語による繰り返しや条件分岐は，もちろん記述は可能ではあるが，Web アプリケーション開発上は使用は想定していない．この定義ファイルは，XML や JSON，さらには独自フォーマットのテキストファイルで記述することも可能であり，キーと値のセットを階層的に記述できれば，そのファイルの内容から定義ファイルの内容は生成可能である．しかしながら，異なる表記にしても，開発作業や記述内容の理解を容易にすることにはつながらないとして，直接的な編集あるいは専用のエディタによる編集に対応することにとどめている．

3.5　開発効率の比較

INTER-Mediator を利用した開発の効率を，コードの生成量で検討する．3.3 節に示した「資産管理」アプリケーションと同等なものを，Web アプリケーションフレームワークの 1 つである CodeIgniter[17] で開発した．両者の開発で作られたファイルの行数を比較する．INTER-Mediator で開発したときに作成したデータベースと CSS のファイルを共通に使い，CodeIgniter で Web アプリケーションを開発した．それぞれのフレームワークで作成したファイルと行数を表 3.3 にまとめた．ファイルの行数は，統合開発ツールの PhpStorm[57] Ver.8.0.1 のソースコードフォーマット機能を利用してフォーマットした結果を数えた．

表 3.3: 同じ機能を持つ Web アプリケーションの開発結果の比較

INTER-Mediator による開発結果	CodeIgniter による開発結果
• ページファイル（HTML ファイル） – 1 ファイル，109 行 – （うち JavaScript は 14 行） • 定義ファイル（PHP ファイル） – 1 ファイル，72 行	• ビューとして機能する HTML 主体の PHP ファイル – 2 ファイル，合計 130 行 – （JavaScript は使用ぜず） • コントローラーおよびモデルの PHP ファイル – 2 ファイル，合計 216 行 • 修正した設定ファイル – 2 ファイル，合計 6 行分
合計＝ 2 ファイル，181 行	合計＝ 6 ファイル，352 行

CodeIgniter での開発物では，2 種類のページがあるので，ビューに対応するファイルを 2 つ用意した．各

種ボタンは同一のものを用意したが，いずれも，クリックすることでサーバー側のプログラムが呼び出されるタイプの動作にした．ページネーションは，CodeIgniter に含まれているクラスを利用した．その表示を整えるために数行の CSS を追加したが，その行数は含めていない．なお，モデルに関しては，SQL コマンドを発行するタイプのシンプルなものにした．

3.5.1 ソースコード行数の比較

表 3.3 より，総合計行数の単純な比較で言えば，CodeIgniter に比べて INTER-Mediator はほぼ半分の行数となった．CodeIgniter のビューに相当する HTML 主体の PHP ファイルは，INTER-Mediator のページファイルに相当する．同一構成のページにもかかわらず CodeIgniter の方が行数が多いのはファイル内で繰り返しやポップアップメニュー処理のための PHP のコードが含まれるからである．また，数行ではあるが，ファイルの数が少ない方がヘッダ部の行数は少なくなる．

INTER-Mediator では定義ファイルとして PHP のファイルを記述するが，すでに説明したように，設定を書き並べた宣言的な記述をするものである．CodeIgniter でのコントローラーやモデルを構成する PHP ファイルでは，オブジェクト指向に基づくプログラミングが必要である．INTER-Mediator の定義ファイルと，CodeIgniter のコントローラーおよびモデルを比較すると，3 倍の違いがあるが，記述の質は CodeIginter での開発の方がオブジェクト指向プログラミングの知識を要求されることなど，開発にかける労力はより高くなることが考えられる．

以上の結果より，INTER-Mediator による開発によって作成が必要なコードは，手続き的なプログラミングを行う一般的な Web アプリケーション用フレームワークよりも少ない傾向にあることが言える．

3.5.2 ソースコード内に含まれるフィールド名

それぞれのフレームワークで作ったアプリケーションにおいて，特定のフィールド名の出現回数を比較した．管理している資産の「メーカー」のデータは，asset テーブルの manifacture フィールドを利用して記録している．INTER-Mediator では，ページファイルにこの「manifacture」という記述が 2 箇所あるのに対して，CodeIgniter は 4 箇所になる．

特定のフィールドに対する読み出した結果の表示という点では，両者ともビューに 1 度フィールドを記述することで実現できる．CodeIgniter でのモデルではテーブルの全フィールドを読み込んでいるので，読み込みに関してはモデル内では検索条件以外は記述されていない．

一方，読み出しと書き込みを行うとすると，CodeInginter ではビューに記述するフィールド名以外に，更新時のやりとりのための手続き的なプログラムが別途必要になり，この例ではコントローラーからモデルへの伝達時と，モデル内での SQL コマンドの生成時の 2 箇所の記述を行っている．更新時にフィールド名が出てこないようにすることも可能ではあるが，通信結果からの取り出しとモデルの更新時に 2 度記述するのが素直な方法であると考えた．

一般的な Web アプリケーションフレームワークでは，HTML のフォームを利用する．その場合，テキストフィールドなどに設定した name 属性を元に，通信結果から入力したデータを取り出すので，更新時など

では name 属性値とフィールド名の突き合わせが必要になる．name 属性とフィールド名を同じにすることもあるが，次に説明するように必ずしもその方法が取れるとは限らない．このアプリケーションの場合は，1 対多の「多」のエンティティの修正結果をまとめて受け取り，それを解析して 1 レコードごとの SQL に展開する必要がある.「多」のエンティティ側のフォーム用オブジェクトでは，フィールド名をそのまま name 属性にすると重複が生じるため，例えば配列にしたり，主キー値を含めた name 属性にするなどの工夫が必要になる．

INTER-Mediator では，ページファイルの属性にフィールド名を 1 度書くだけで，読み出しと更新ができるように工夫をしている．この点からも開発効率は高いと言える．また，1 対多のリレーションシップがあってもそれぞれのコンテキストに従って 1 レコード分だけの記述を行えば良いので，フォームを使った手法のように複数のデータソースの混在を意識した作りをしなくても済む．

3.6　関連研究および関連製品

宣言的な記述を用意することでデータベースとの連動を行う仕組みを持つフレームワークとして，古い時代から存在して現在でも販売されている製品として ColdFusion（Adobe Systems）[108] がある．ColdFusion ではタグを拡張することで，繰り返しや条件分岐を始めとして一般的なプログラミング言語に匹敵する機能を実現し，多様な要求に対応できる仕組みを備えるに至っている．そのため，手続き的なプログラミングと同様にアルゴリズムの記述を正確に行う必要がある．INTER-Mediator では独自タグは定義しておらず，HTML の規約を外れないでページファイルを宣言的に記述できる．

手続き的なプログラミング言語を用いた MVC を基調とするフレームワークでは開発が大規模になりがちであり，シンプルな手法でエンドユーザーによる開発を可能にする必要性も主張されている [63]．XForms や XQuery といった標準的な XML の規格をベースにしたフレームワーク XFormsDB[64] も開発されているが，INTER-Mediator は最終的な HTML ページをテンプレート処理の基本とするため，より直接的な手法と言える．

サーバーサイドで稼働する Web アプリケーションの多くは，ページのテンプレートを記述した上で，コントローラーやモデルをクラスとして定義する．ここでのテンプレートは HTML を基調にはしているが，独自のコマンド的な記述を追加したり，あるいはプログラムと混在させて記述しなければならなず，ビューの解析を困難にしたり [147]，あるいはロジックをビューに含めてしまう原因にもなる．INTER-Mediator は，純粋な HTML でテンプレートを記述でき，テンプレート自体がそのままブラウザで確認ができ，HTML 編集アプリケーションでの編集で問題が生じない．

宣言的な記述をもとにした Web アプリケーション開発フレームワークやツールとしては，Web サイトのモデリングを行うための言語である WebML[16] を記述言語として用いたものがある．WebML により Web サイトを構成する要素やそのリンクをダイアグラムで記述可能である．その記述を基にした Web サイトを構築するシステムも開発されている [10]．さらに，WebML をベースにコンテキストを考慮したモデル設計ができる拡張版も開発されている [15]．WebML により Web サイト全体の構造やデータのやりとりを記述でき，系統的な設計が期待できる．一方，INTER-Mediator は HTML によるページそのものを直接的に記述するため，作成結果が分かりやすい．

宣言的な言語で Web アプリケーションを設計できる Hilda[106][105] は，動作を SQL 文を交えた宣言的な記述としてテキストで表現したものを利用する．アプリケーションの機能とデータ操作をページ表示部分とは別々に管理できる点で構造的な開発ができるが，宣言的な記述が可能にするためには要求される処理を抽象化が必要であり，実質的には手続き的なプログラムを記述する能力は必要になると言える．Web アプリケーションのアーキテクチャ上の構成としては評価できるが，エンドユーザーが開発や保守に関われるかどうかという点では，INTER-Mediator の方が HTML への直接的な記述で可能な点で，より適合度があると言える．

モバイルデバイス向けの宣言的な開発言語 Mobl[51] は，ユーザーインタフェースを宣言的な記述で開発でき，モバイルデバイスでの開発に適合したものである．ローカルのデータベースのモデル化も可能であり，ユーザーインタフェース自体を宣言的な記述で指定する．Web ベースで稼働するため，典型的なモバイルアプリケーションを異なる OS で実行する手段としては評価できるが，モバイル以外のサポートやサーバーベースのデータベース利用に関する直接的なサポートはない．INTER-Mediator は，HTML を利用してページ内容を自由に構成でき，共有データベースを利用できるため，用途はより広くなる．

データベースと Web アプリケーションの連携を容易に行う開発ツールとしては，Microsoft VisualStudio LightSwitch[20] やジャストシステム社の UnitBase[125] などがある．また，ASP サービスについてもサイボウズ社の kintone[124] がある．これらは，例えばフィールドをページ上にドラッグするなどしてデータベースと連動する要素の配置ができるなど，ツールによる利用を主体とした使い勝手の高さが特徴となっている．これらのツールを用いた場合も INTER-Mediator と同様に，手続き的なプログラムの修正を行うことなしに一定範囲の保守は可能である．これらは総じて開発環境や動作環境が限定されるのに対し，INTER-Mediator はオープンソースであり利用者による活用の幅は広い．kintone では API も提供されているが，特にサーバーサイドでのプログラムを組み込める点では INTER-Mediator の方が拡張性が高い．

エンドユーザーがデータベース構築を手軽にできる ASP サービスとしては，Salesforce[127] がある．Salesforce クラウドベースの CRM（Customer Relationship Management：顧客管理）ソリューションである．あらかじめ定義されたさまざまなテーブルを利用でき，必要に応じて利用者が独自にフィールドの追加もできる．システム標準のリスト表示などの使い方から，アプリケーションとしてさまざまな機能を統合したような使い方までができる．こうしたスキーマが決められていることでの容易さと不便さはある．INTER-Mediator はデータベースを自由に設計できる点で違いがある．

「Forguncy[119]」は表計算のような画面で Web ページを作成できるツールである．Excel の方眼紙のように Web アプリケーションを作成でき，Excel で培った知識を Web アプリケーションに活かせるとしている．ただし，HTML への直接的な記述ではないこともあり，ツールの持つ機能に制限される．INTER-Mediator は，HTML を直接編集する点での柔軟性があると言える．また，INTER-Mediator は手続き的なプログラミングを書かないでもある程度の開発は可能だが，宣言的な記述でできないことにも対処可能である．

3.7 INTER-Mediator の概要に関するまとめ

本章において，INTER-Mediator の機能概要と，INTER-Mediator を利用した Web アプリケーション開発の基本的な手法を解説した．機能概要に示した通り，INTER-Mediator は宣言的な記述でデータベース連動

Web アプリケーションを開発に必要なさまざまな機能を利用できる．スキーマ適用したデータベースを用意した上で，定義ファイルとページファイルを記述する．ページファイルは PHP で記述するが，キーと値のセットを記述することが開発に必要な作業で，手続き的なプログラムを記述しているわけではない．定義ファイルにはデータベースを利用するための設定や，アプリケーションの動作などを指定する．ページファイルは Web ページのテンプレートとなり，そこにターゲット指定として，例えばテーブル名とフィールド名などの記述を行う．すると，その要素はデータベースのテーブルのフィールドをバインドして，フィールドの値を合成してページ上に表示し，テキストフィールドであれば値をユーザーが変更するとデータベースへの更新を行う．ページネーション，新規レコード作成，レコード削除，マスター/ディテール形式のユーザーインタフェースといったデータベースアプリケーションに必要な機能も宣言的な記述で利用できる．手続き的なプログラミングが必要な Web フレームワークと INTER-Mediator で同等な Web アプリケーションを開発してコードの行数を比較した結果，INTER-Mediator では約半分の記述となった．宣言的な記述を中心にしながら，Web アプリケーションに必要な諸機能を持っている．そして一般的なフレームワークより短い記述で同等の仕組みが実現できることから，開発時間の短縮化や，あるいは開発結果を解析作業をやりやすくするといった利点がある．

第4章　INTER-Mediatorを用いたシステムの改変

本章において，INTER-Mediator を用いて作成したシステムを保守する状況を検討し，宣言的な記述の範囲内で可能な作業範囲を特定する．本章の内容は筆者らによる論文 [136] での記述をもとにしているが，その後に追加された機能を含めて記述している．通常，システムの構築は専門家による分析，設計，実装，テストを経て完成される．そのシステムに対する要求に変化がある場合，同様にして専門家に対して変更の依頼をするのが一般的である．しかしながら，少しの修正であっても手続き的なプログラムの修正が必要な場合が多く，エンドユーザーがシステム改変に主体的に関わることは難しい．

INTER-Mediator で構築する作業においては，分析とその結果をデータベースとして蓄積可能にするためのスキーマ構築，そしてクライアントサイドでの応答などに対応する JavaScript でのプログラミング，さらにはサーバーサイドでの処理の追加など，データベース設計や手続き的なプログラミングの知識が必要な場面がある．一方，ページファイルや定義ファイルの修正については手続き的なプログラミングの知識は必要なく，システム構築の専門家でなくても行える範囲である．INTER-Mediator で構築するシステムに関して，開発プロセスの中でエンドユーザーによる主体的な参画が可能なのは保守作業である．本章では，保守作業で発生する作業を分類した上で，宣言的な記述の修正で行える範囲を特定する．

4.1　INTER-Mediatorを利用した開発

Web アプリケーション開発を行う場合，図 4.1 に示すような開発プロセスを経由することになる．開発における一般的な手順としては，業務分析を元にデータベースのスキーマやアプリケーションが備えるべき機能が決定され，実装し，テストするといった作業が発生する．上流工程はユーザー自身が行うこともあるが，複雑な展開があるような場合には専門家による分析が必要になる場合がある．特に，データベースのスキーマを決定することは，一般にはエンドユーザーには難しい作業である．そして，実装においても，複雑な要求を満たすためには手続き的なプログラミングが必要になる場合が一般的である．本論文で対象とした保守作業の前に，こうしたシステム稼働までの開発作業がある．エンドユーザーが作業に関わるとしても，そこまでの作業はエンジニアが主体的に行う必要がある．一方，作ったシステムの改変を中心とした保守作業は，宣言的な記述部分の修正で済む場合が多くある．その点に注目し，保守作業を分類した上で，エンドユーザーによる保守作業ができる点を示す．

なお，アプリケーション開発においては，「後からプログラムでなんとかする」といったプランが立てられ，スキーマをシンプルなもので構成するようなこともある．しかしながら，INTER-Mediator の特徴を生かすには，宣言的な記述で処理可能な範囲を広げることでもあり，そのためにはリレーションシップを含むスキーマの定義の段階で，さまざまな機能を実現できるようにすることが望ましい．例えば，あるフィールドの値に依存した選択肢を持つポップアップメニューを作りたい場合を考える．手続き的なプログラミ

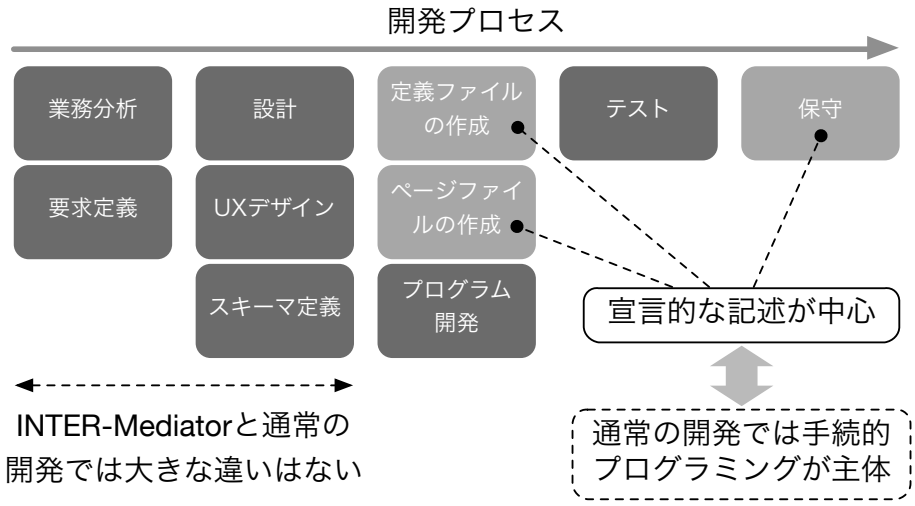

図 4.1: INTER-Mediator を利用した開発のプロセス

ングができる状況では，単にポップアップの選択肢のためのマスターの定義だけをスキーマとして記述し，そのマスターから取り出した一連のデータを IF 文等で絞り込むような手法が取られる可能性もある．一方，INTER-Mediator では，関連するリレーションシップを定義したコンテキストから選択肢が得られるため，そうしたポップアップメニューも宣言的な記述で構築できる．そのとき，マスターテーブルが単に存在することだけではなく，依存元のフィールドとのリレーションシップを考慮する．そして，リレーションシップ自体は定義ファイルに記述する必要がある．

4.2　システム改変の分類

データベースを利用する Web アプリケーションにおける改変作業を分類したのが表 4.1 である．図 4.2 には表 4.1 の番号で示した改変を適用する場所を示した．なんらかの改変作業を行う場合，1 つの分類項目である場合もあるが，複数の分類項目にまたがる場合もある．この分類は作業対象を基準に導き出しているのであり，要求を基にしているものではないからである．本章では作業対象を特定しやすくするために，この分類に基づいて分析を行うことにする．本章で，INTER-Mediator では①〜③についてはプログラムの変更なしにできることを示す．なお，④〜⑥は専門家によって実施される内容と位置付けている．

4.3　ページ要素の改変

表 4.1 の①の改変については，HTML で記述されたテンプレートの一部の修正のようなきわめて小規模な改変であれば，フレームワークや開発手法の種類を問わず手続き的なプログラムの知識は不要である．少し複雑な事例として，テキストフィールドで入力していたフィールドをポップアップフィールドに変更したい場合を検討する．まず，ページ要素の改変だけの場合を示し，より実用的な意味でマスターから取り出した内容をポップアップメニューに表示する場合の改変の手順についても検討する．

表 4.1: システム改変の発生する状況

	概要	例
①	ページ要素	• 表示する順序変更 • 非表示フィールドを新たに追加表示 • 文字の色の変更
②	データベース要求	• レコード取得のための検索や並べ替え条件の変更 • リレーションシップを伴うコンテキストを記述 • レコード作成や削除機能を追加する
③	単一フィールド応答	• 小数以下の桁数を 2 桁から 3 桁に増やす • 一定の文字列を前後に追加する • 計算フィールドを定義する
④	ユーザーインタフェースのカスタマイズ（クライアントスクリプト）	• 特別な処理を行うボタンの設置 • ページ生成のための拡張処理の記述や変更
⑤	データベース応答（サーバーサイドスクリプト）	• データベースだけで実現できない集計処理を追加 • PDF 生成などの追加処理が必要なデータ
⑥	スキーマ変更	• 新たなビューを作成する • テーブルやフィールドを新たに作成する
*	新規ページ追加	• スクラッチから作成する • 既存ページを複製後，①〜⑥の改変をする

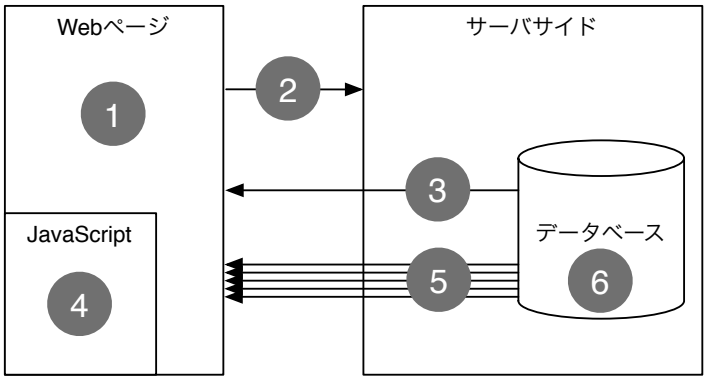

図 4.2: ページ改変の分類

4.3.1 一定の選択肢のポップアップメニューに変更する

INTER-Mediator を利用して作成したアプリケーションに対して，表 4.1 の①に相当する要素の種類を変更する改変作業を示す．図 3.3 の Web ページにある「分類」を，テキストフィールドではなくポップアップメニューから一定の文字列を選択して入力できるようにしたいとする．ポップアップメニューに変更するために，ページファイルの一部をリスト 4.1 のように input タグ要素から select タグ要素に変更し，加えて選択肢を option タグで記述する．

リスト 4.1: ポップアップメニューで選択できるようにする

```
【変更前】
<input data-im="assetdetail@category" />
【変更後】
<select data-im="assetdetail@category">
    <option value="個人用">個人用</option>
    <option value="共用">共用</option>
```

```
7 </select>
```

ページ要素の修正が必要になれば，タグの修正はフレームワークの種類に限らず必要である．しかしながら，INTER-Mediator ではこの記述だけで，ページ表示したときに category フィールドの値に応じて対応する選択肢が選択される．また，ポップアップメニューを選択することで，選択した項目に対応する値を自動的に元のレコードの category フィールドに書き込む．こうしたユーザーインタフェースとモデルの連動を取る手続き的なプログラムの記述は不要であり，バインディングの仕組みによって自動的に背後で処理されている．

一般的な手続き的なプログラミングを主体としたフレームワークでも，テンプレートを HTML 主体で記述し，テンプレート処理のために少数の拡張記述を加えた程度で記述できる場合もある．表 4.1 の①の改変を行う場合，HTML の変更だけで済まない場合には，プログラムの中にある変更可能な要素を探し出すためにはビューやコントローラーがどのファイルであるのかなどの構成全体を把握する必要があり，小規模な修正でも全体的な知識が必要である．また，タグの種類に応じたプログラムの変更が必要になることもある．一方，INTER-Mediator では，ページファイルの記述のみの変更で対処できる．

4.3.2　ポップアップメニューの選択肢をマスターテーブルより取得する

アプリケーションを開発した当初は文字列でデータを入力していたが，いくつかの決まった文字列である場合には，その文字列をレコードのフィールドに持つマスターテーブルを利用する方が記述がばらばらにならないなどの利点がある．このとき，マスターテーブルがない場合には，表 4.1 の⑥の改変が必要になる．SQL データベースでは一般にはエンジニアが検討をしてテーブルの追加等を行うことになり，この部分はエンドユーザーだけでなくエンジニアの援助が必要である．

ここで，新たに「分類」フィールドでの選択項目をマスターテーブルの「category」から取得することになったとする．ポップアップメニューでは category テーブルに入力されている文字列を選択肢にする．category テーブルでは主キー値を整数で保持するフィールドもある．asset テーブルでは文字列を記録する category フィールドではなく，数値を記録する category_id フィールドに置き換える．category テーブルには主キー値の category_id と，カテゴリ名を記録する name の 2 つのフィールドを用意する．データの移行を含めて，ここまでの作業はスキーマの設計に精通したエンジニアが行ったとする．

以後の作業は，リスト 4.2 のように宣言的な記述の修正で可能である．まず，category テーブルを利用するための category コンテキストを定義ファイルに追加する．単純なマスター参照だけなら，name キーの値を指定するだけで良い．そして，ポップアップメニューのための SELECT タグの data-im 属性にあるターゲット指定のフィールド名を category_id に変更する．選択肢のための option タグは 1 つだけ記述する．ここで詳細ページでは，TABLE タグ内にさらに TABLE タグがある場合，内部でさらにクエリー内容がページに展開されることを示した．同様な関係がこの場合は TABLE タグと SELECT タグで実施される．ここでは SELECT タグの下位の要素として，category テーブルのレコードの数だけ OPITION タグ要素が並ぶことになり，マスターの内容がポップアップメニューの選択肢となる．OPTION タグのターゲットしては 2 つの項目があり，それぞれ「category@category_id@value」「category@name」である．後者は category テーブ

ルの name フィールドの値を OPTION タグのテキストの子要素として追加することを示す．前者は@が 1 つ多くなっており，category テーブルの category_id フィールドの値を，OPTION タグ要素の value 属性の値に設定することを支持している．結果として各レコードに対して，value 属性は category_id フィールドの値，そして要素の内容にはカテゴリの名前を示す name フィールドの文字列が設定される．

リスト 4.2: ポップアップメニューでマスターより選択できるようにする

```
【定義ファイルに追加されるコンテキスト】
array{
   'name' => 'category',
},
【ページファイルでの変更箇所】
<select data-im="assetdetail@category_id">
   <option data-im="category@category_id@value category@name" value=""></option>
</select>
```

なお，category フィールドには文字列で「個人用」などと記録していたが，この改変により category テーブルの主キーである category_id フィールドにある数値を記録する．その結果，図 3.2 に示す一覧表示の「分類」列では，category フィールドの値がそのまま見えるようになり，文字列ではなく数字が見えるようになる．元のように文字列が見えるようにする修正は，4.6 節で解説する．

4.4 データベース要求の改変

表 4.1 の②の改変に相当するような例として，図 3.2 ではすべてのレコードを一覧しているが，破棄していない資産（discard フィールドが未入力）だけの一覧を出したくなったとする．

INTER-Mediator で改変を行う場合，リスト 4.3 のようなコンテキストを定義ファイルに記述することになる．作業手順は次の通りである．定義ファイルではすでに記述されたリスト 3.1 のコンテキストをコピーして name キーの値を「asseteffect」に書き換え，query キーの定義を追加する [1]．ページファイルは既存のものを複製して，ターゲット指定のコンテキスト名が asset のものを asseteffect に変更すれば良い．

リスト 4.3: 定義ファイルを修正してクエリーに検索条件を適用する

```
array(
   'name' => 'asseteffect',
   'view' => 'asset',
   'sort' => array(
      array('field' => 'purchase', 'direction' => 'ASC'),
   ),
   'query' => array(
      array('field' => 'discard','operator' => '>', 'value'=>'1990-1-1'),
   ),
   'repeat-control'=>'insert delete',
   'records' => 5,
   'paging' => true,
   'navi-control'=>'master-hide',
),
```

一般的なフレームワークでは，SQL で記述した要求の変更やあるいはモデル部分のプログラムの変更，あるいはコントローラーでの受け入れ場所の確保など，手続き的なプログラミング作業を通じて記述の追加

[1] MySQL では未入力の日付型フィールドの値は「0000-00-00」となるので適当な日付（ここでは 1990 年 1 月 1 日）より前であれば，discard フィールドは未入力とみなせる．

や編集が必要になるような改変である．INTER-Mediator ではデータベース利用を宣言的に記述できるので，それに関連する改変も宣言的な記述の変更で対処できる．

4.5　単一フィールドに対するデータベース応答の改変

表 4.1 では，データベースからの応答を変更する仕組みとして，③の 1 フィールド単位と，⑤の単一あるいは複数のレコード単位の応答を変更する場合に分けた．⑤に対応する改変については，INTER-Mediator を使った場合においても，手続き的なプログラムの作成を必要とする．③のような改変の例として，データベースから得られる日付の書式を整えたいとする．MySQL では日付型のフィールドのデータは「2012-12-19」のような ISO8601 形式で得られるが，年月日をスラッシュで区切った日付で表示したいといった要望が発生したとする．加えて，INTER-Mediator では計算プロパティをサポートする．計算プロパティは同一レコードの他のフィールドや，あるいは他のコンテキストのフィールドを元にした計算結果の表示を行う読み出しのみのフィールドを定義できることである．⑤の改変と見ることもできるが，単一フィールドで得られるためここでは③の改変に分類する．

4.5.1　フィールドで得られた値の書式を整える

INTER-Mediator で作った Web アプリケーションでは，定義ファイルでの IM_Entry 関数の 2 つ目の引数にリスト 4.4 のような配列を定義することで対処できる．この記述により，asset テーブルの purchase フィールドについては parameter キーに従った「12/12/19」，discard フィールドについては既定の書式である「2012/12/19」といった形式に変換する．ページ上のテキストフィールドに表示された日付を変更した場合，データベースに書き込むときに，これらの形式から ISO8601 形式への逆変換も行う．日付や数値，文字列の追加などの一般的な変換の仕組みはフレームワークに組み込まれており，1 フィールドのデータの変換については，カンマ付きや通貨記号付きなどいくつかの典型的な書式処理は定義ファイルへの記述を追加するだけで行える．

リスト 4.4: 定義ファイルに追加してフィールドに書式設定を適用する

```
1  array(
2    'formatter' => array(
3      array( 'field' => 'asset@purchase',
4            'converter-class' => 'MySQLDateTime',
5            'parameter'=>'%y/%m/%d'),
6      array( 'field' => 'asset@discard',
7            'converter-class' => 'MySQLDateTime'),
8      ),
```

同じようなことを一般的なフレームワークで行う場合は，プログラム中にフォーマット変換を行う記述を追加することになる．一方，INTER-Mediator は日付や数値などについては定義ファイル上の設定として対処できる．

4.5.2 計算プロパティの追加を行う改変

貸出履歴は rent テーブルに残っているが，その情報を元にして貸出期間を求めて，さらにその平均値を求めるといった要求が出てきたとする．

まず，貸出期間は，貸出日から返却日までの日数を求めればよい．いずれも同一のレコードにあるフィールドから求められるので，リスト 4.5 の rent コンテキストにあるように，calculation キーに対する値を定義する．値も配列で指定する．式を expression キーで記述するが「$if(backdate =\!'',\!'', date(backdate) - date(rentdate))$」で記述した式を各レコードに対して計算し，field キーの値である datelength という名前のフィールドがあたかもあるかのように振る舞う．if や date は関数であり，backdate と rentdate はそれぞれ返却日と貸出日が記録されたフィールド名である．date 関数は文字列から 1970 年元日からの日数を示す整数に変換する．この計算はクライアントサイドで行っているため，計算式の項のフィールドがテキストフィールドなどの変更可能な要素にバインドされている場合，そのデータを修正すると自動的に再計算も行う．

もう 1 つの要求は，計算結果で求めた貸出期間の平均値を求めるというものである．元データは，rent コンテキストにある datelength フィールドである．平均値は 1 つの資産に対して計算されるものであり，詳細ページに表示するとしたら，assetdatail コンテキストに計算結果が得られる必要がある．ここで，assetdatail と rent は 1 対多の関係がある．このような「多」に相当するコンテキストが,「1」に相当するコンテキストのさらに内部にある場合,「1」のコンテキストに「$average(rent@datelength)$」という式，つまり assetdetail コンテキストから見て内側にある rent コンテキストの datelenght フィールドを集合として取得して平均値を求める average 関数に渡すことで，不定数のレコードにあるフィールドの値を計算することができる．

リスト 4.5: 定義ファイルに計算プロパティを追加する

```
array(
   'name' => 'assetdetail',
      [省略]
   'calculation'=>array(    //この定義を追加した
      array("field"=>"avglength","expression"=>"average(rent@datelength)"),
   )      //式の意味：rent コンテキストの datalength フィールドの値を合計する
),
array(
   'name' => 'rent',
      [省略]
   'calculation'=>array(    //この定義を追加した
      array("field"=>"datelength",  //この名前を読み出し専用のフィールドのように使用できる
         "expression"=>"if(backdate = '', '', date(backdate) - date(rentdate))"),
   )      //式の意味：backdate フィールドが空欄でなければ，rentdate からの経過日数を求める
),
```

リスト 4.5 の変更を施し，その後にページファイルをリスト 4.6 のように変更した．datalength フィールドのためのテーブルのセルを追加し，さらに平均値を表示するために avglength フィールドの値を SPAN タグ要素に展開した．変更した結果，図 4.3 に示すように，貸出日数を表示する列が貸出履歴に増え，さらにその平均値が求められている．

リスト 4.6: ページファイルにフィールドを追加する

```
<tr>  <td colspan="4">
   <table>  #以下の表の見出しは 1 列増えるので「貸し出し日数」を表示した
      <thead><tr><th>貸出日</th><th>返却日</th><th>貸出日数</th><th>担当者</th><th>メモ</th></tr></thead>
      <tbody><tr>
         <td data-im="rent@rentdate"></td>
```

```
6           <td data-im="rent@backdate"></td>
7           <td data-im="rent@datelength"></td> #計算プロパティで得られた結果はフィールドと同様に記述
8           <td>  <select data-im="rent@staff_id">
9                 <option data-im="staff@staff_id@value staff@name"></option>
10              </select>
11          </td>
12          <td><input type="text" data-im="rent@memo"/></td>
13      </tr></tbody>
14    </table> #以下の行に，計算プロパティで得られた平均値をページ上に表示するための要素を追加した
15    <div>平均貸出日数：<span data-im="assetdetail@avglength"></span></div>
16  </td> </tr>
```

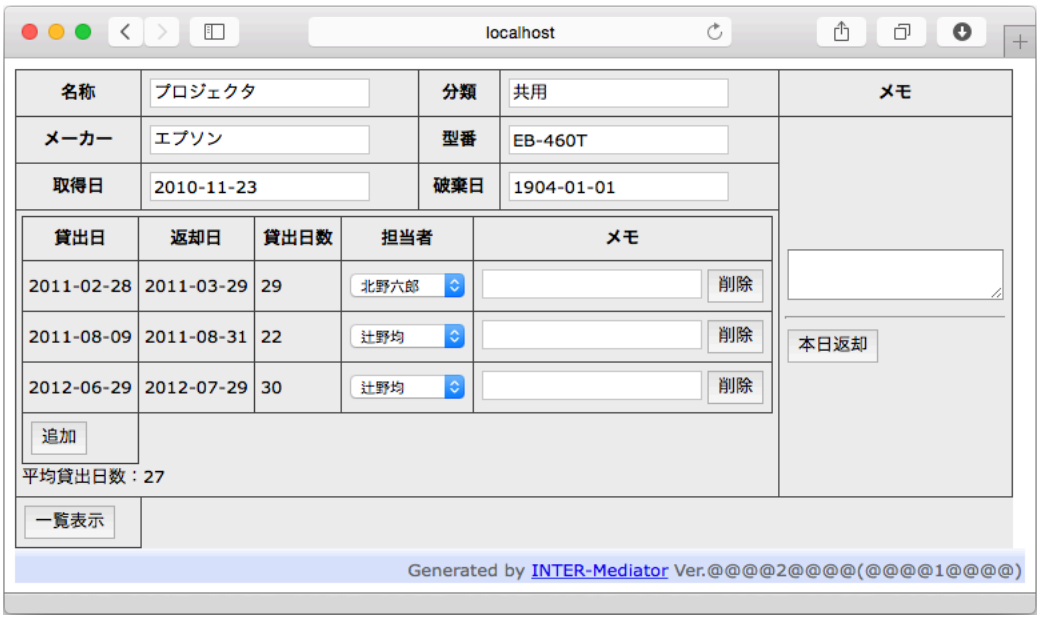

図 4.3: 改変した結果

一般的なフレームワークを使う場合，計算結果を含むビューを定義する方法が考えらえるが，クライアントサイドで計算結果を更新するための手段を講じる必要がある．MVC タイプのフレームワークの場合はモデルで計算結果を扱えるようにすることや，その計算結果をコントーラが適切にビューに伝達されるようにするなど，手続き的なプログラムでの修正が発生する．INTER-Mediator では宣言的な記述で計算プロパティを定義ファイルに追加し，それらが表示されるようにページファイルを修正することで改変ができる，宣言的な手法で解決できる．

4.6　リレーションシップを伴う改変

4.3.2 項では，1 つのフィールドの値をマスターテーブルより取得し，そのフィールドにはマスターテーブルの主キー値（連番の数値の 1 つ）を記録するようにした．このままでは，図 3.2 に示す「分類」列には数字が見えてしまう．ここで，改変による副作用の修正が必要になる．すでに 4.3.2 項での修正が終わっているとすれば，残りの修正はフィールドの記述を変更するという外見的な要因では③の改変である．また，データベースの応答についての記述を加えるため，②に関する改変でもある．

一覧表に提示する asset テーブルの category_id フィールドにある値は，category テーブルの主キー値であ

る．したがって，asset テーブルと category テーブルとのリレーションシップを通じて，category テーブルのどのレコードと関連付けられているのかを規定する必要がある．そのために新たなコンテキストとしてリスト 4.7 のような定義を定義ファイルに追加する．このコンテキスト「category-in-list」を利用する場合は，view キーに対応する値を参照して category テーブルにアクセスし，そのとき relation キーに対応する設定を参照して，一覧表示側の category_id フィールドの値と，category テーブルの category_id の値が一致するものを取得する．

リスト 4.7: 定義ファイルへのリレーションシップ情報の追加

```
array(
   'name' => 'category-in-list',
   'view' => 'category',
   'relation' => array (
      array(
         'foreign-key' => 'category_id',
         'join-field'=> 'category_id',
         'operator' => '='
) ) )
```

一方，ページファイルではリスト 4.8 のように，td タグ要素による 1 つのセルの中に，2 重になる span タグ要素を記述し，それぞれに data-im-control 属性を指定する．これによりテーブルに対して行がレコードの数だけ複製されるのと同じ動作が，外側の SPAN タグ要素に対して内側の SPAN タグ要素がレコードの数だけ複製されることになる．ただし，主キーとのリレーションシップのため，通常は内側の SPAN タグ要素は 1 つあるいは存在しないかのどちらかになる．内側の SPAN タグ要素には data-im 属性により，category-in-list コンテキストで得られた name フィールドの値が要素の内容として設定されるようにする．これで，表のセルの中には「個人用」などの文字列が表示されるようになる．

リスト 4.8: マスターテーブルから取り出した関連レコードを表示する

```
【変更前】
<td data-im="asseteffect@category"></td>
【変更後】
<td>
   <span data-im-control="enclosure">
      <span data-im-control="repeater" data-im="category-in-list@name"></span>
   </span>
</td>
```

一般的なフレームワークでは，このような変更が生じた場合，スキーマレベルでのビューの作成や，あるいはモデルの応答を変更するといった作業が必要になる．INTER-Mediator では宣言的な記述で，リレーションシップの追加を伴う改変も可能である．

4.7 関連研究

ISO/IEC 25010 に規定された「システム／ソフトウェア製品の標準品質モデル」の中に「保守性」というカテゴリがあり，さらにその中で「モジュール性」「再利用性」「解析性」「変更性」「試験性」といった項目が挙げられている [122]．INTER-Mediator の機能と本章での検証をこれらの項目に照らした結果を表 4.2 にまとめた，モジュール性や再利用性は HTML の記述の特性に概ね一致するため，テキストそのものの活用という意味ではあると言える．また，定義ファイルの 1 つのコンテキスト定義は，1 つのモジュールとし

て扱うことも可能であり，複数のページでの再利用もできる．解析性については，手続き的なプログラムのような長いコードが出てこないこと，HTML が実際にページ上に見える内容と対比しやすいこと，定義ファイルの記述は単純なキーと値の組み合わせの羅列であることを考えれば，解析はしやすい状況にある．変更性については，本章で示した通り，さまざまな動作を宣言的な記述の変更で対処でき，その結果は他の部分に大きく影響を与えないものも多い．ただし，INTER-Mediator はデータベースエンジンを別のソフトウェアに委ねているため，INTER-Mediator で作った Web アプリケーションについては変更しやすい箇所と，例えばコマンドでのデータベース管理といった変更を行うための知識が必要な箇所が発生する．試験性については INTER-Mediator 自身は特に機能を持たない．Web アプリケーションの自動テストに使われる Selenium[91] などの別のツールを使う必要がある．結果として品質モデルの基準で言えば，HTML/CSS をベースにしたレベルでのモジュール性や再利用性があり，宣言的な手法での変更性を持つフレームワークとなる．

表 4.2: システム／ソフトウェア製品の標準品質モデルの「保守性」

品質特性	INTER-Mediator に対する評価
モジュール性	HTML テキストや項目の羅列である定義ファイルではこの特性はあると言えるが，オブジェクト指向のクラスのような意味でのモジュール性はない
再利用性	「モジュール性」と同様
解析性	記述がシンプル，HTML は表示結果と対比しやすい，定義ファイルは項目の羅列だけであり把握はしやすい
変更性	宣言的な記述により変更ができ，他の部分への影響は比較的少ない．一方，データベースの管理に及ぶと変更は困難になる
試験性	作成したアプリケーションの試験を行う仕組みは持っていないので，他のツールを使う

エンドユーザーによる Tailorablity[2] を持つシステムを設計する手法が研究されており，Tailorablity があることを示すために開発したシステムの変更をケーススタディ形式で進めたときの知見をまとめている [94]．本章で保守作業の可能性を例で示している点では共通の手法である．Tailorablity とは，既存のアプリケーションをベースにした開発や，システムやフレームワークの開発時にユーザーの要求に応じて改変ができる性質を示している [66]．保守や再開発は Tailorablity によってその範囲が広がることが指摘されており，その結果エンドユーザーとソフトウェアエンジニアが共同作業する機会が増える [26]．エンドユーザーによる保守を促進するためには，保守の段階でもなんらかのエンジニアのサポートが受けられることが望ましく，それぞれの開発プロセスでエンジニアとエンドユーザーが協調して開発に取り組む必要性 [61] と合致する．

開発プロセスとエンドユーザーによる保守に関して，エンドユーザーソフトウェアエンジニアリング [12] の中では保守作業はプロセスの 1 つとして捉えられている．エンドユーザーソフトウェアエンジニアリングは，要求定義（Requirements），Design and Specification（設計と仕様），Reuse（再利用），Implementation（実装），Testing and Verification（テストと確認），Debugging（デバッグ）といった 6 つの作業に分類されている [61]．保守は Reuse の中の作業の 1 つと位置付けている．Reuse を行う上でのエンドユーザーにとっての問題として，ある目的で作られたものを別の用途に転用できるかどうかの判断ができないことがあるとしている．その解決策には Tailorablity を持たせることが示されている [65]．すなわち，Reuse の観点からは知識や開発物を別の用途に転用できることによって，より効率的に保守が進められると言い換えるこ

[2] コンピュータの分野では適切な訳語が定義されていないので，英語で記述をする．

とができる．INTER-Mediator の定義ファイルやページファイル内の宣言的な記述に関しては，それほど多くのポターンが存在するということではない．筆者のさまざまなフレームワークを使ってきた経験から言えば，さほど多くないルールを理解すれば，Web アプリケーションは開発できる．また，HTML の便利な点はタグ単位でコピー&ペーストをし，ペースト先で修正するといった記述が一般的に行われている．これらの点から INTER-Mediator は知識の転用は十分に可能な仕様となっていて，Tailorablity があると言える．

4.8 INTER-Mediator を用いたシステムの改変に関するまとめ

以上のように，表 4.1 で示した①〜③の改変については，一般的なフレームワークでは手続き的なプログラミングが必要になる場面でも，INTER-Mediator は宣言的な記述の変更で改変が可能となる．④〜⑥については，もともと手続き的な記述を行っていたり，スキーマ変更のようにエンジアリングの知識が必要な箇所であり，INTER-Mediator を用いても手続き的なプログラムの修正が必要である．分類上は半分ではあり，比較的マイナーな修正に相当するのが①〜③ではあるが，宣言的な記述の範囲内でエンドユーザーによって主体的に作業を進められるとすれば，システムの継続的な利用を促進する要因になる．

第5章　フレームワークの動作上の特徴

INTER-Mediator による Web アプリケーション作成に必要な定義ファイルとページファイルの作成方法について第 3 章で説明した．本章では，これらの情報を利用して，どのように Web ページのテンプレート処理やデータベースへの更新が行われているのかを説明する．最初に全体的なアーキテクチャについて議論し，テンプレート処理を始めとするさまざまな機能を解説する．

5.1　INTER-Mediator のアーキテクチャ

INTER-Mediator を利用した Web アプリケーションの構成は図 3.1 に示した通りで，開発者が用意するのはデータベース，定義ファイル，ページファイルである．こうした手法が Web アプリケーションに関して持つ意味付けを最初に議論する．

5.1.1　モックアップ駆動開発

一般的な手続き的プログラミングを行って利用するフレームワークは MVC アーキテクチャを採用しているのが一般的であり，開発者自身が，アプリケーションの動作を行うためのモデル，ビュー，コントローラーのそれぞれのコンポーネントを，フレームワークに用意されたクラスを継承するなどのさまざまなサポートを得て定義して利用する．しかしながら，INTER-Mediator では，明確な MVC のコンポーネントの定義は開発時には行わない．内部的動作の上では，ある部分はコントローラーの一部などと言える機能を持っているものの，エンドユーザー開発者に対してはその部分を意識しなくてもいいような実装をした．アーキテクチャを意識した構成をしなくても Web アプリケーションをユーザーインタフェースである HTML のテンプレートを中心に開発できる点が，エンドユーザーにとって開発物自体が解りやすいと感じられることに注目した．

成果物に近いものを最初に作る「プロトタイピング」は従来からある手法であるが，Mockup Driven Development[8]（MockupDD）といった手法が提唱されており，アジャイル開発プロセスの 1 つとして位置づけられている [84]．Web アプリケーション開発で言えば，HTML をベースにしてユーザーインタフェース部分をまず作り，そこから開発を始めるといった手法である．一般にはモデル駆動の開発が推奨されている．抽象的な意味ではモデルはシステム動作の前提となる基本的な構造を定義することもある．実装の上では，MVC の階層をいずれも手続き的なプログラミング言語で記述する場合，モデル部分がすべての処理の基礎になることもあって，インタフェースや機能等を早めに確定した上で，残りの階層を構築することで，後々の大幅な変更を避けるといった意図がある．

予算規模の大きなプロジェクトでは設計のプロフェッショナルが参画して，モデル駆動の開発が行われて

いる一方，従来から小規模な開発では HTML でまずはデザインを確定した上で，アプリケーション開発に取り掛かる事例もあった．また，近年のモバイルアプリケーションの開発の中でも手書きやあるいは手書き風のモックアップ作成ツール等を用いて，プログラミングにかかる前に誰もが判断できる形式のビジュアル（「ワイヤーフレーム」と呼ばれることもある）を作成した上で機能を検討するという手法は一般的になった（一例として [75]）．MockupDD は，現場サイドでは従来から行われている手法である．Web 開発での MockupDD では，HTML でページを作るため，この作業はエンドユーザーでもできる．それが完成品でなくても，どんなページが必要で，ページ表示に必要なフィールドの特定も具体的にできる．また，クリックしても動かないボタンでも，このページにボタンを押せばこうした機能が実現できる必要があるといったユーザーの要求が明白になる [85]．モデリングの専門家ではない人でも取りかかれる点として，宣言的な記述でのモックアップ作成が必要である点も指摘されている [8]．

一方，モックアップを最初に作れば全てがうまくいくというわけではない．最初にモックアップを作ったとしても，その後は開発者が作業を行い，完成した結果を改めてユーザーに納品したら，要求と違っていたということも発生する．最初のモックアップにすべての要求が具体化されているわけではなく，ユーザーがこの状態ならできると思っていた作業が実際にはできなかったり，あるいはすっかり忘れていたということもある．一方，開発する側も，より実装しやすいものへと小さな変更を加えるかもしれないが，その結果，要求を満たさなくなることも考えられる．この解決策の 1 つがアジャイルプロセスの導入であり，例えばモックアップを毎日確認する作業を続け，確定した段階でモデル設計に入るという手法が提唱されている [84]．実現すべき機能がモックアップ上で可能な限り明確化されることで，すべての要求が見える化することを意識することで，MockupDD は意義を持つ．

INTER-Mediator を利用した開発は，MockupDD として提唱されている手法に合致する．そして，稼動するための修正は，属性の追加などであり HTML のモックアップそのものに対して大きな変更を加えてはいない．そのため，実装後の修正も行いやすく，その点が保守に対する適合性を発揮している．

5.1.2　コンテキスト指向

定義ファイルに記述するデータベースとのやりとりのための一連の情報を「コンテキスト」と名付けた．INTER-Mediator の動作をおおまかにわかりやすく説明する場合,「属性にテーブル名とフィールド名を書けばバインドできる」と表現しているが，正確にはテーブル名ではなくコンテキスト名を指定する．同一のテーブルでも，異なるコンテキストを定義することもできるようになっている．

コンテキストを導入した 1 つの理由は，データソースの意味付けを明確にするためである．例えば，社員のテーブルがあったとき，そこから抽出をすることで,「部長のテーブル」「女性のテーブル」などの意味付けがより明確になった一連のレコードの集合が得られる．そうした意味付けされたリレーションをコンテキストと位置付けた．従って,「検索条件とソート条件を交えたテーブル」というのがコンテキストの 1 つの見方でもある．加えて，そこにリレーションシップの定義やページネーション時の表示範囲も加わる．

コンテキストを利用するもう 1 つの理由は，リレーショナルデータベースの動作を考慮したものである．SQL で定義するビューは，最初の理由の意味ではコンテキストである．もちろん，ビューをそのまま利用するテーブルとして使用できる．しかしながら，ビューの更新やレコード削除等には制限もある場合もある．

そこで，読み出しはビューを使い，更新はテーブルに対して直接行えることが要求されることを想定した．そのため，コンテキストの中の view キーと table キーでそれぞれ読み出し，更新のためのエンティティを指定できるようにした．ただし，この仕組みを使いこなすには，書き込めるフィールドとそうでないフィールドの見極めが必要になり，SQL とデータベースエンジンに関する知識は不可欠である．なお，単純な指定ができるように，view および table キーの指定がなければ，name キーで指定したコンテキスト名をエンティティ名に使用して，データベースへのアクセスを行う．

コンテキストによってデータの意味付けを明確にすることにより，必要な機能の抽出と実装をしやすくすることを意図している．コンテキストとして必要十分なデータ群を得れば，それを HTML 上に展開するというイメージで開発できることを期待した機能である．

コンテキストと同等な考え方は SQL の初期の時代からあり，セマンティックデータモデルとして，データベースのモデル設計の 1 つの手法として研究されていた [46][1]．コンテキストをユーザーの位置や物理的な状況といった概念まで広げた context-aware の仕組みを持つアプリケーション開発を行うフレームワークも提唱されている [24]．また，モデルベースでの宣言的な Web アプリケーション開発においても，コンテキストを考慮したモデル化を行う手法も提唱されている [15]．さらに，コンテキストをベースにしたデータ交換やユーザーインタフェース作成が可能な開発ツールとして「コンテキサー [137]」が開発されていて，業務アプリケーション開発の実績もある．

システム全体のアーキテクチャとして DCI（Data, Context and Interaction）[19][83] といった手法も提唱されている．この手法はオブジェクト指向プログラミングにおける問題点を解決するものとして，データとは別にコンテキストして，要求に応じたオブジェクトを提供するものとしている．手続き的プログラミング言語でのふるまいに関するものなのでより抽象的な表現となっているが，データベース操作の上でのシンプルな要求に置き換えれば INTER-Mediator での実装との共通点がある．

データベースアプリケーションソフトの FileMaker は，明確ではないものの「コンテキスト」の考え方が根底にある．テーブルの定義とそれを実体化した「テーブルオカレンス」を別々に扱い，テーブルオカレンスの段階でリレーションシップを定義する．1 つのテーブルから複数のテーブルオカレンスを定義でき，用途に応じたテーブルオカレンスを用意できる．テーブルオカレンスでレコードの集合と選択しているとみなす 1 つのレコードを保持する．テーブルオカレンスとレイアウトを組み合わせることで，ウインドウ内にデータが表示されるようになる．ただし，ユーザーセッションに相当するレイアウトおよびウインドウの組み合わせもコンテキストの形成に影響を与えており，ユーザーによる検索処理によって同一のテーブルオカレンスを指定したレイアウトでもウインドウが違えば異なるレコードセットを保持する．

データベースアプリケーションソフトの Microsoft Access よりも FileMaker の方が少ない開発作業で利用できる点が見られる。その理由を検討した結果，テーブルとレイアウトの間に存在するテーブルオカレンスが大きな要因であると判断した [135]．FileMaker はコンテキストをテーブルオカレンスが保持し，同じテーブルオカレンスを利用する別のレイアウトに切り替えても，コンテキストは切り替えたレイアウトにも反映される．その結果，例えば 20 レコード中の前から 5 番目のレコードを選択したという状況で別のレイアウトに切り替えても，同じレコードセットが適用され，選択レコードも同じものが適用される．結果的に，マスター/ディテール形式の利用がレイアウトの切り替えだけでできる．これに対して Microsoft Access は一覧でクリックしたレコードを表示させるために検索条件を込めたデータセットの取得が必要になる．マ

クロをウィザードで追加できるとは言え，簡単な作業なら SQL の考え方を知っていないと構築できない．FileMaker のこうした使い勝手の良さは，設計のしやすさにもつながると考え，それを Web アプリケーション開発に当てはめて「コンテキスト」という概念を導き出した．

5.2　テンプレート処理

INTER-Mediator でのテンプレート処理は，図 3.1 に示した「コア機能」のコンポーネントで行われる．ページファイルを元にデータベースから得られたデータを埋め込み，ページとして表示する HTML が生成される．Web アプリケーションでは，ロジックとユーザーインターフェースを分離することが可能な点から [79]，フレームワークにテンプレート処理が組み込まれるのが一般的である．通常はテキスト処理によるテンプレート処理がサーバーサイドで行われるが，INTER-Mediator では DOM をベースにした処理を行う．執筆時点ではクライアントサイドでの処理を行っている．DOM を一定のルールで解析する手法により，独自のタグを定義しなくても純粋な HTML で記述されたテンプレート処理が可能となる．図 5.1[1] にはページの合成とページ上での編集結果からデータベース更新を行う流れの概要を示した．

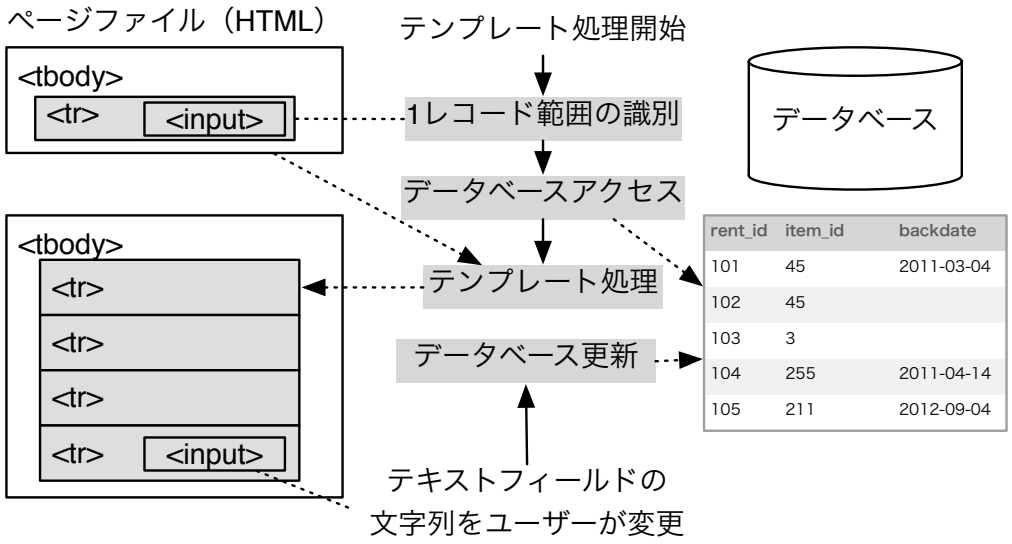

図 5.1: テンプレート処理と更新処理の概要

5.2.1　1 レコード範囲の識別

テンプレート処理では，まずページファイルの解析が行われ，1 レコードに対応付けられる範囲を識別する．その結果，1 レコード分のデータを挿入するノード群の「リピーター」と，リピーターを束ねる「エンクロージャー」が求められる．典型的なリピーターは TR タグ要素群であり，その場合にエンクロージャーとなる要素は TBODY タグである．INTER-Mediator が識別できるエンクロージャーとリピーターは表 5.1 にまとめた．OPTION タグを除いて，エンクロージャーの子要素に含まれる複数の要素を 1 つのリピーターと

[1]本章の図版では定義ファイルを PHP のコードではなく，値とキーをコロンで区切り，配列はブランケットで囲むなど，JSON の記述に近い略記法で記述する

して識別する．DIV および SPAN タグ要素に関しては，data-im-control 属性に表にあるような値を記述することで，エンクロージャーとリピーターとして識別させることができる．これらの要素は HTML ページ内の任意の場所に記述可能であるが，エンクロージャーの子要素にリピーターが定義されている必要がある．

表 5.1: 識別可能なエンクロージャーとリピーター

要素の種類	エンクロージャー	リピーター
テーブル	TBODY	TR
ポップアップメニュー	SELECT	OPTION（単一要素のみ）
箇条書き	UL	LI
番号リスト	OL	LI
汎用タグ	DIV(data-im-control=enclosure)	DIV(data-im-control=repeater)
汎用タグ	SPAN(data-im-control=enclosure)	SPAN(data-im-control=repeater)

リピーターとエンクロージャーの識別手順は次の通りである．ページ内にある BODY タグ要素の内部を前順走査を行い「ターゲット指定」（data-im 属性の記述がある要素）がある「リンクノード」を見つける．そのノードを含めた上位のノードについて，エンクロージャーとリピーターのセットを見つける．その後，すべてのリンクノードを収集するために再度リピーター内を走査する．ただし，さらに内部に異なるエンクロージャーとリピーターが存在すれば，その内部のリンクノードは収集しない．収集したリンクノードのターゲット指定の情報から，定義ファイルのどのコンテキストを利用するかを求める．もし，いくつかのコンテキストを指定していれば，指定しているノードが最も多いコンテキストを使用する．リスト 3.2 の最初の TABLE タグ要素内のエンクロージャーでは asset というコンテキストを使うことになり，定義ファイルの view キーの設定より asset テーブルからレコードを取得する．

5.2.2 フィールドの値をタグ要素に表示する

エンクロージャーとリピーターが確定すると，それに対応するコンテキストが確定する．その結果，コンテキストに記述された情報を利用してデータベースへアクセスを行い，レコードセットが得られる．レコード設定に含まれているフィールドのデータを，リピーター内部のリンクノードとして定義されたタグ要素にマージする様子を示したのが図 5.2 の前半部分である．フィールドの名の突き合わせを行っている．ターゲット指定が「invoice@issueddt」のように 2 つの要素からなる場合には，タグの種類に応じてフィールドのデータを追加する．type 属性が text の INPUT タグでは，フィールドの値を value 属性に設定するので，ページ上ではテキストフィールドの中にフィールドの値が表示される．一般には，そのタグ要素のテキスト子要素として追加するが，フォームで利用するタグ要素については個別に処理をする．

5.2.3 データベースの更新

Web ページに呼び出されたデータベースのデータを，逆にデータベース側に更新をかける仕組みも用意されていて，INPUT タグによるテキストフィールドに合成されたデータは，変更することでデータベースに書き戻せる．図 5.2 の後半にその仕組みを記載した．テキストフィールドにデータベースのフィールドの

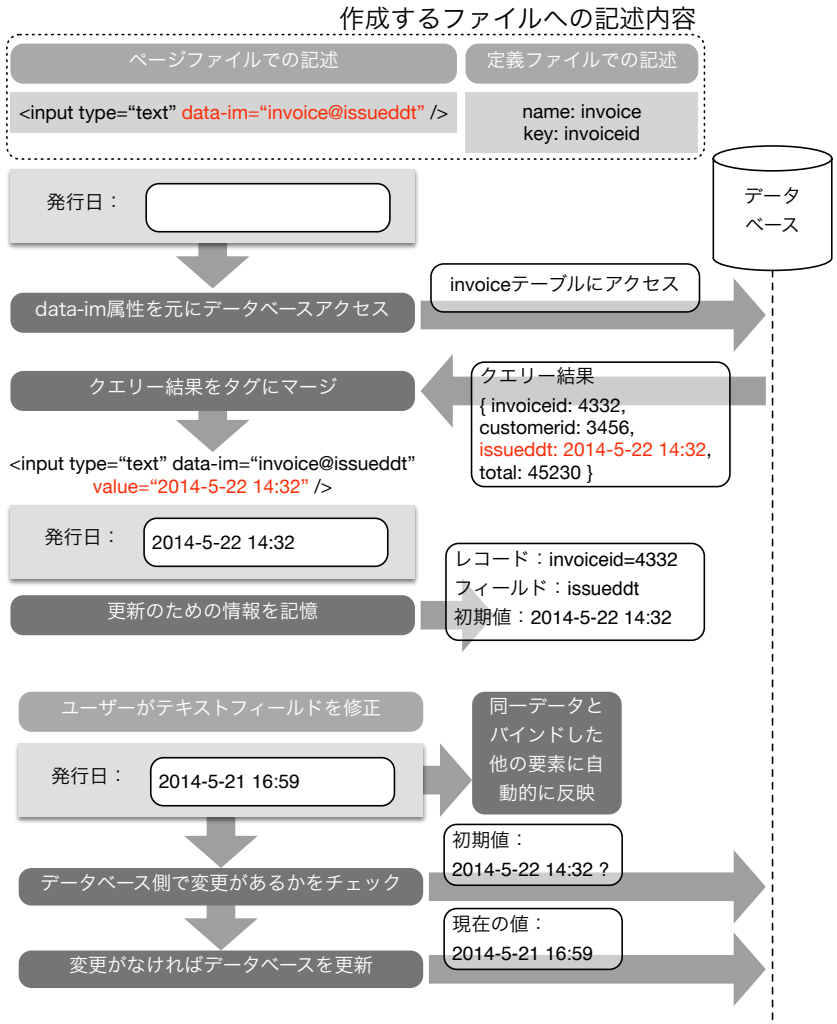

図 5.2: フィールドへのバインドと更新処理

値を挿入するときに，JavaScript のオブジェクトに，そのレコードのコンテキスト名，キーフィールドの値，そして挿入した値などを記録する．ページ生成時に要素ごとに固有の id 属性を割り当てており，その id 属性についても記録する．すなわち，画面に見えているテキストフィールド 1 つ 1 つに対して，コンテキストやキー値を記録しておく．このテキストフィールドの値をユーザーが書き直して別のフィールドに移動するタイミング（change イベント）でスクリプト起動するようにしておき，コンテキスト名からテーブル名を取得し，キー値をもとに検索条件を作成して，更新処理を行う．

以上のような，データベースのフィールドをページに表示し，修正して更新するまでの一連の処理動作を「バインディング」と呼ぶ．INTER-Mediator はバインディングを定義ファイルへのコンテキストの定義と，ページファイルへの属性の記述のみで実現している．

5.2.4　複数のレコードの展開

データベースから得られたレコードが複数ある場合，1 レコードの内容を 1 つのリピーターに合成する．その手順を示したのが図 5.3 である．具体的には，エンクロージャー内のリピーターを一度子要素から取り

除いておく．レコード 1 つ 1 つに対して，取り置いてあるリピーターの複製を作成し，リンクノードに該当するフィールドの値をマージし，そしてそのリピーターをエンクロージャーの子要素とする．この繰り返し処理を行うことで，複数のレコードをページ上に展開できる．TABLE タグ内に TR タグ要素が 1 つだけの場合，その 1 つの TR タグがレコードの数だけ複製され，例えば 10 行のテーブルがページに表示される．検索により取り出したレコードが 0 の場合，何も表示されなくなるが，リピーターの中の特定の要素（data-im-control 属性が noresult のもの）は，レコードが 0 の場合にのみに利用される．この手法により，レコードが存在しなかったことを表示する方法も，宣言的に記述できる．

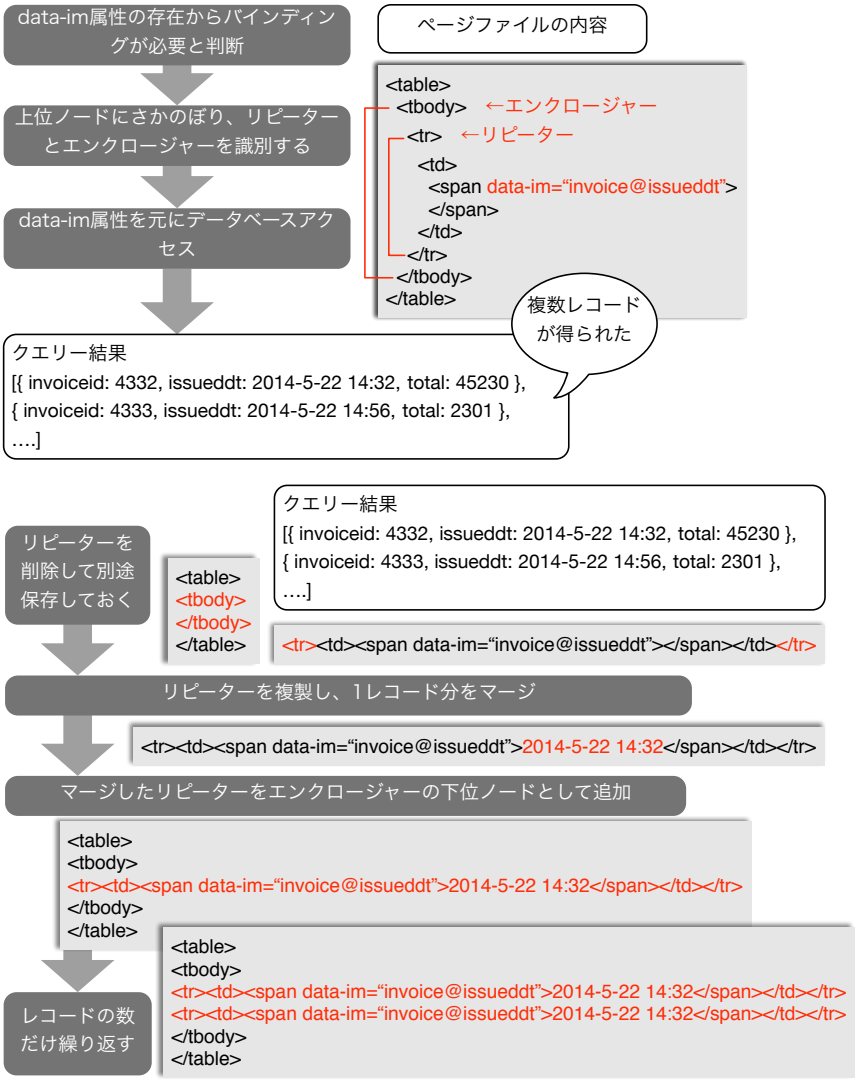

図 5.3: 複数のレコードに対する処理

5.2.5 リレーションシップの実施

あるレコードに対する関連レコード（例えば伝票と明細のような関係）があるようなリレーションシップが存在し，それを Web ページ内に展開したい場合にも対応できる．図 5.4 ページファイル内で，リピーターとエンクロージャーのセットの中に，さらにリピーターとエンクロージャーのセットを記述する．上位

のセットが基準となるレコードで，それに対する下位のセットは 0〜複数個のレコードを展開する．

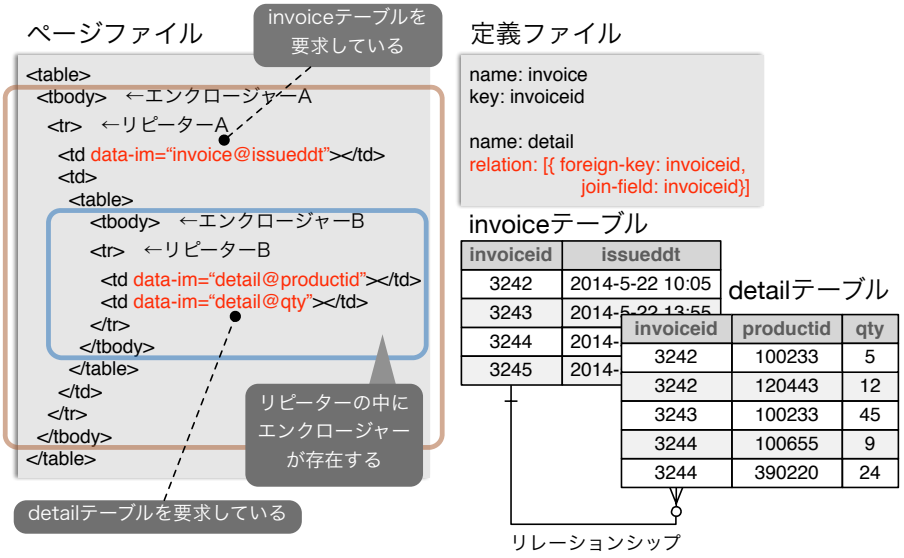

図 5.4: リレーションシップがある場合

実際の処理手順を示したが図 5.5 である．階層的なリピーターとエンクロージャーのセットを見つけるために，1 レコード分のテンプレート処理が終わった後にリピーターの内部を探索する．内部にリピーターとエンクロージャーのセットがさらに見つかると，そこで新たに合成処理を行う．見つかった内部のリピーターを分離し，使用するコンテキストを確定してデータベースアクセスを行う．定義ファイルに relation キーによる定義があれば，上位のリピーターのレコードにある値を検索条件に追加してデータベースアクセスを行い，結果として関連レコードの取り出しが可能となる．その後は，識別されたエンクロージャーの子要素にレコードの内容をマージしたリピーターを追加して行けば良い．

5.2.6　コンテキストモデルの記録内容

データベースの内容をページ内の要素に展開するときに，内部的に展開したデータを記録するオブジェクトを確保している．そのオブジェクトを「コンテキストモデル」と呼ぶことにする．定義ファイルで記述した 1 つのコンテキストは「コンテキスト定義」と呼び，ここでは区別をする．1 つのコンテキストに対して，コンテキストモデルは 0〜複数個作られる．コンテキスト定義に対応したエンクロジャーの数だけ，コンテキストモデルが作られる．

単にページにデータベースの内容を表示するだけならばコンテキストモデルは不要である．コンテキストモデルが必要な理由の 1 つは，ユーザーインタフェース側でデータを変更したときに，その結果をデータベースに書き戻す情報を残しておくことである．あるテキストフィールドに見せている値が，どのテーブルのどのレコードのどのフィールドかを記録する役目を持つ．もう 1 つの理由は，テキストフィールドの値を変更したときに，その変更結果を異なるユーザーインタフェースコンポーネントに伝達することである．伝達は，同一ページにある要素だけでなく，異なるクライアントに対しても行いたい．そのためには，Observer パターン [118] に基づく実装を行うことで，1 つの更新処理が，さまざまなオブジェクトに伝達す

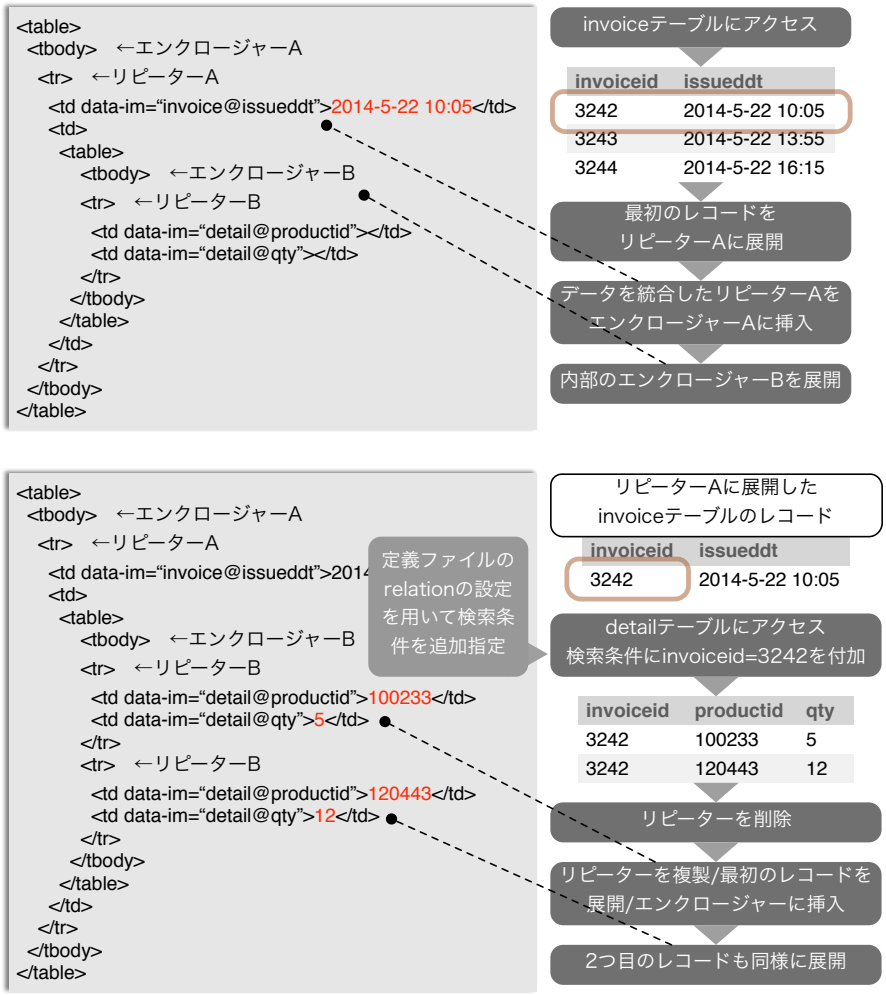

図 5.5: 内部のエンクロージャーでの関連レコードの展開

る仕組みで実現することが効率良いと考え，コンテキストモデルを実装した．クライアント間でのデータ共有の詳細は 5.8 節で説明する．

コンテキストモデル内では，データベースより得られたリレーションをそのまま保存するだけでなく，ページ内のどの要素が，特定のレコードの特定のフィールドとバインドしているのかを記録できるようにしている．ページの合成時には要素の id 属性が一意になるように，フレームワークによって付け直す．あるレコードのあるフィールドに対して，複数の id 属性値を記録しておくことで，データベースのデータと要素の間の対応付けができるようになっている．また，要素の value 属性だけでなく，さまざまな属性やスタイルシート項目へのバインドを実現しているため，指定された要素の何にバインドしているかといった情報も持たせている．

5.2.7 テンプレート処理のアルゴリズム

ここまでに説明したページ合成のアルゴリズムを説明する．図 5.6 は，ページ合成の概略を示したもので，ボックスに記載された「ExpandRepater」などは，この後に具体的に示すアルゴリズムで定義されているプ

ロシージャ名である．ルートノードとなる BODY 要素から SeekEnclosure プロシージャーを呼び出し，エンクロージャーを発見するまで前順走査を行う．エンクロージャーが見つかれば，ExpandEnclosre プロシージャーが呼び出され，内部のリンクノードを探索してコンテキストを確定し，関連するレコードをデータベースから取り出す．そして，ExpandRepeater プロシージャーが呼び出されて，各レコードのフィールド値をリピーター内のリンクノードに挿入する．リピーター内部にエンクロージャーがあるかをさらに SeekEnclosure プロシージャーを呼び出して走査を続ける．内部にエンクロージャーがあれば，リレーションシップの定義を考慮したデータベースアクセスを行い，リピーターを複製して追加する作業を行う．アルゴリズム上は，再帰呼び出しの形式になっていることから，エンクロージャーとリピーターの組み合わせは，何段階にも定義されていても正しくレコードの取り出しと HTML 要素への展開ができる．

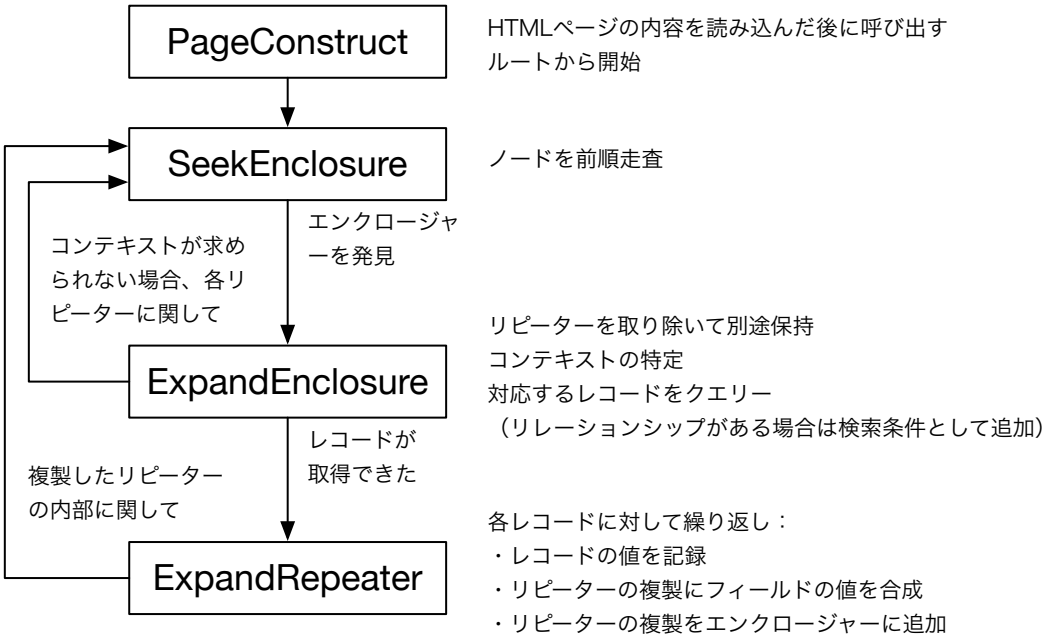

図 5.6: ページ合成処理の概要

以下，それぞれのプロシージャーを詳細に説明する．ページ合成は，例えばページをロードした直後のイベントを捉えて，アルゴリズム 5.1 に示した PageConstruct 関数を呼び出すことで実施される．以下，複数のアルゴリズムを記述するが，下線が引かれた動作は，そのアルゴリズムあるいは別のアルゴリズムに procedure として記述されたものを示す．

SeekEnclosure プロシージャーはエンクロージャーを求めて，存在すれば ExpandEnclosure プロシージャーを呼び出す．エンクロージャーであるためには，決められたタグ名あるいはタグ名と属性であって，その要素よりも下位の階層にはリンクノードが存在している必要がある．しかしながら，SeekEnclosure プロシージャーの最初の条件判断では，ここではタグの種類と属性を確認するだけで判定している．例えば TBODY タグ要素であってその子要素として TR が 1 つあるいは複数存在しているかどうかを確認しているのみである．以下の一連のアルゴリズムでは，リンクノードが存在していない場合でもエンクロージャーとみなしてしまうが，リンクノードが存在しない場合は何もしないように実装している．

ExpandEnclosure プロシージャーのアルゴリズムはアルゴリズム 5.2 に記述した．一度リピーターを削除

アルゴリズム 5.1 ページ合成のアルゴリズムの開始

```
procedure PageConstruct
    SeekEnclosure(ページの BODY タグ要素, NULL)
end procedure

procedure SeekEnclosure(node, record)
    if node がリピーターを持つエンクロージャーかどうか then
        ExpandEnclosure(node, record)
    else
        for each childNode in node の子ノードの配列 do
            SeekEnclosure(childNode, record)
        end for
    end if
end procedure
```

するエンクロージャー以下を解析して，リンクノードを収集するが，その中にあるエンクロージャーより下位の階層の要素は無視し，これらは展開後に改めて解析する．リンクノードを収集して，それぞれの data-im 属性の値を調べて，設定されているコンテキスト名を集める．複数のコンテキスト名が混在する可能性があるが，ここでいちばん多く使われているコンテキスト名を，そのエンクロージャーで利用するコンテキスト名として，コンテキスト名を確定する．リンクノードがないなどコンテキストが決定できない場合には，元の状態に戻して，リピーターを出発点として SeekEnclosure プロシージャーを呼び出し，エンクロージャーの探索を再開する．コンテキスト名が決まれば，コンテキスト定義を参照し，さらにコンテキストモデルを生成し，リピーターの複製をコンテキストモデルのプロパティとして記録する．ページの合成を始めて，最初に ExpandEnclosure プロシージャーを実行するときには，引数 record は NULL になるので，リレーションシップに関する処理はない．コンテキスト定義に指定されたテーブルや検索条件に従ってデータを取得して，ExpandRepeaters プロシージャーを呼び出す．

ExpandRepeaters プロシージャーのアルゴリズムはアルゴリズム 5.3 に記述した．引数には，コンテキストモデル，エンクロージャーの要素，データベースから得られたレコードが渡される．クエリー結果にレコードが含まれていない場合には，リピーターの中からレコードがないときに表示される要素（data-im-control 属性が"norecord"のリピーター）を選択して，それをエンクロージャーの子要素とする．レコードがあれば，それぞれのレコードに対して，レコードがないときのリピーターを除いたリピーターを複製する．そして，複製したリピーター内の各リンクノードに対して，ターゲット指定を元にフィールド名と，要素のどこにデータベースの値を設定するかを割り出して，実際にノードに追加する．ここでは，例えば INPUT タグによるテキスト要素ならば value 属性，DIV などの一般的なタグ要素ならばテキストノードを子要素として追加するなど，要素の種類に応じた処理を行う．データベースから得られた値を要素に合成した後，複製したリピーターをエンクロージャーの子要素とする．ここで，リピーターの中に，エンクロージャーがあるかどうかを SeekEnclosure プロシージャーを呼び出して調べる．最初に SeekEnclosure プロシージャーを呼び出した時には，record 引数は NULL であったが，エンクロージャー/リピーターによる合成処理をした後は，現在のレコードを引数として SeekEnclosure プロシージャーを呼び出す．次に見つかったエンクロジャーに対して ExpandEnclosure プロシージャーを呼び出すときに，上位のエンクロージャーのレコードが引き渡さ

アルゴリズム 5.2 エンクロージャーを元にデータベースからデータを取得

```
procedure ExpandEnclosure(enclosureNode, record)
    repeaters ← リピーターの複製を作る
    リピーターをエンクロージャーから取り除く
    linkedNodes ← repeaters からリンクノードの配列を取得する
    contextDef ← linkedNodes からコンテキストを確定する
    if (コンテキストが確定できない（リンクノードがない，コンテキスト定義がないなど) then
        for each repeater in repeaters do
            repeater をエンクロージャーの子ノードに戻す
            SeekEnclosure(repeater, record)
        end for
    else
        contextModel ← contextDef からコンテキストモデルを生成する
        contextModel.repeaters ← repeaters
        if record! = NULL then
            record からリレーションシップに関連するデータを得てクエリー条件に付加
        end if
        recordSet ← コンテキストに従ってクエリーを実施してレコードセットを得る
        ExpandRepeaters(contextModel, enclosureNode, recordSet)
    end if
end procedure
```

れ，そのときには record 引数は NULL でなく，コンテキスト定義に従って，外部キーに対応するフィールドが上位レコードの主キー値と同じであるという検索条件が付加されてクエリーが実施される．

テキストフィールドなどのバインドされたユーザーインタフェースオブジェクトにおいてユーザーが表示されているデータを変更すると，アルゴリズム 5.4 で示した処理が実行される．定義ファイルに従った値のチェックを行い，その後に，テキストフィールドとバインドされたコンテキストモデル上の値とと，データベース側の現在の値を付き合わせる．データベースへの検索の手がかりはコンテキストモデルに残されている．もし，値が異なるとしたら，別のユーザーによってデータベースが更新されていることが考えられるので，そのまま上書きするか，あるいはデータベース更新をキャンセルするかのダイアログボックスを表示して，ユーザーに判断させるいわゆる楽観的ロックの仕組みが稼働している．そのまま更新をするとすれば，テキストフィールドの値をコンテキストモデルに反映させ，その後にデータベースアクセスしてデータベース側を更新し，さらに同じフィールドとバインドしている他の要素への更新も行う．

以上のアルゴリズムによって，テンプレートとなるページファイルの状態から，データベースアクセスを行ってそのデータを合成したページを作成し，さらにユーザーによる書き戻しを行うまでの一連の処理が実現している．エンクロージャーとリピーターを識別する処理を，再帰的な呼び出しによって内包するものを探索して処理を進めることで，個別のレコード単位でのリレーションシップも考慮されたページ合成ができる．

アルゴリズム 5.3 得られたレコードからリピーターを繰り返し合成する

```
procedure ExpandRepeaters(contextModel, enclosureNode, recordSet)
    if recordSet のレコード数が 0 の場合 then
        レコードがない場合のリピーターをエンクロージャーに追加
    else
        for each record in recordSet do
            repeaters ← contextModel.repeaters の複製（レコードがないときのものは無視）
            linkedNodes ← repeaters からリンクノードの配列を取得する
            for each linkedNode in linkedNodes do
                linkedNode に対して id 属性値を設定
                targetField ← linkedNode の対応するフィールドを取得
                targetValue ← record から targetField の値を取得
                targetValue を linkedNode の値あるいはテキストノードに設定する
                contextModel に linkedNode の id 属性とともにフィールドのデータを記録
            end for
            for each repeater in repeaters do
                repeater をエンクロージャーに追加する
                repeater に対して id 属性値を設定
                contextModel に repeater の id 属性とともにフィールドのデータを記録
                SeekEnclosure(repeater, record)
            end for
        end for
    end if
end procedure
```

5.3 ページネーションの実現

ページネーションとは，たくさんのレコードを表示する時に，10 レコードずつなど一定数ごとのレコードに区切って表示する手法である．数万件のレコードをブラウザにロードして表示することも不可能ではないが，表示に時間がかかるなど操作性が悪くなる．数十程度のレコードで区切ることで，操作性が悪くなることを最小限にとどめることができる．一方で,「次の 10 件」を表示するなどのボタンを配置するなど，表示範囲を変更するためのユーザーインタフェースが必要になる．INTER-Mediator ではこうしたページネーションに必要な処理を，ページファイルと定義ファイルの記述だけで実現している．図 3.2 に見られる「更新」ボタンのあるグレーのボックスが INTER-Mediator によって自動的に生成されたページネーションの例である．制限としては，1 ページ内では 1 つのコンテキストにのみページネーションを適用することができるという点があるが，一般的なアプリケーションではこの制約は大きな問題にはならない．

ページネーションを行うには，定義ファイルのコンテキストの中に，records キーの値で 1 ページあたりのレコード数を指定し，paging キーの値を true に指定する．この設定がある 1 つのコンテキストだけが 1 ページ内でページネーションを設定できる．リスト 3.1 に示した定義ファイルのソースでは，name キーの値が「asset」のコンテキストが，ページネーション対象となる．一方，ページファイルでは，id 属性に「IM_NAVIGATOR」という値を持つ要素を定義する．要素は特に指定はないが，DIV タグや P タグを利用するのが一般的である．サンプルアプリケーションのリスト 3.2 内にもその記述がある．そのタグ要素の下

アルゴリズム 5.4 データベースへの更新処理

```
procedure ValueChange(idValue)
    if コンテキストに定義した値チェックを行い，違反がある場合 then
        return
    end if
    nodeValue ← idValue から対応するノードに入力された値を得る
    contextModel ← idValue から対応するコンテキストモデルを得る
    contextRecord ← contextModel 内の idValue に関連付けられたレコードを得る
    currentValue ← contextRecord から idValue に関連付けられたフィールドの値を得る
    primaryKey ← contextRecord から主キー値を得る
    record ← contextModel の元になったテーブルに対して primaryKey が主キーのレコードを得る
    if currentValue と record から得られた該当フィールドの値が違う then
        if ダイアログボックスで問い合わせ，更新のキャンセルが指示された then
            return
        end if
    end if
    nodeValue を contextModel に記録する
    同一レコードの同一フィールドにバインドした他のノードの表示値を更新
    データベースに対して更新
    更新が必要なコンテキストに対して更新する
end procedure
```

位要素としていくつかの要素を自動的に追加し，ページネーションの機能を持たせる．なお，ボタンなどのペーネーション内部の要素については CSS によるカスタマイズが可能である．

フレームワークの内部では，ページネーションを実施するコンテキストに対して，最初のレコードを表示するためにスキップするレコード数を記録するプロパティをグローバル変数のオブジェクトに記録させておき，ページネーションをクリックすることで，レコード数分スキップ数を前後させて，再度データベースを検索し，表示内容を更新する．1 ページあたりのレコード数も定義ファイルで指定するだけでなく，実際にはスキップ数と同様に，グローバル変数で参照できるオブジェクトのプロパティに設定する．これらのデータを記録しておくことで，ページ送りの仕組みが実装できる．

5.4　新規レコードとレコード削除

データベースのデータを表示することと，表示したデータを修正することでデータベース側の更新をするというバインディングの動作は前節までに解説した通りである．CRUD に基づくデータ処理では，他に，新規レコード作成とレコードの削除が機能として必要になる．本節では，宣言的な記述でそれらの機能を実現するボタンを用意する方法と，入力専用のページを作成する仕組みを説明する．

5.4.1 コンテキストに対する新規レコードとレコード削除

新規レコードや，レコード削除のボタンは，定義ファイルで指定することで自動的にページ上に配置される．

コンテキストに新規レコード作成ボタンの生成を指示すると，一連のレコードの最下部あるいは，一定数ごとのレコードの表示範囲を切り替えるページネーション上のいずれかに，新規レコード作成ボタンが追加される．新規レコード作成時の初期値は通常はデータベースのスキーマで設定できるが，コンテキストでも指定できる．INTER-Mediator では，新規レコードを作ったときには，そのレコードが編集状態になるように動作する．これは，Microsoft Access や FileMaker での動作に習ったものである．

レコード削除については，定義ファイルに指定を行うことで，リピーターの内部にボタンを用意するか，ページネーション上にボタンを用意できる．フィールドの更新と同様，コンテキストとレコードを特定するためのキーフィールドと値を保持し，ボタンをクリックすることでそれらの値をもとにデータベースにコマンドを送ってレコードの削除を行う．レコード削除後には，フレームワークが対応するリピーターを削除する．ただし，ページネーションに関連したコンテキストの場合は，レコード削除後にページの合成をフレームワークがやり直す．

5.4.2 新規レコードの作成のみを行うページ

アンケートや参加申し込みのような，入力のみをもっぱら行うようなアプリケーションも Web ではよく利用されている．INTER-Mediator はテキストフィールドなどをデータベースのテーブルにあるフィールドとバインドして自動的に値の表示や更新ができるが，入力のみのページを作成するのはバインドの仕組みを利用できなくはないものの，あらかじめレコードを用意するなど準備が必要になる．つまり，入力専用のページのように見せるような工夫が必要になってしまう．そこで，入力専用のページを作成する「ポストオンリーモード」という動作をサポートした．

5.4.3 入力専用ページの作成方法

ポストオンリーモードでは，テキストフィールドは，データベースへのバインドを行わない単なるユーザーインタフェースオブジェクトである．作成するには，テーブル名との対応付けを行うために，通常通り定義ファイルを作成する．ここで，asset という名前のコンテキストがある場合，リスト 5.1 のような HTML の記述をページファイルの作成することで，入力専用のページが稼働する．テキストフィールドにキータイプして「新規入力」ボタンをクリックすると，asset テーブルに新しいレコードが作成され，テキストフィールドに入力したデータが対応するフィールドに入力された状態になる．ここでの「新規入力」ボタンの複数クリックを防ぐため，クリックを受け付けた後はボタンを消すことができる．定義ファイルの設定により，ボタンを消した後に表示するメッセージや，あるいはボタンをクリックした後に移動するページを指定することもできる．

リスト 5.1:「資産一覧」のページファイルの例

```
1 <html>
```

```
<head>
    <script src="asset_contexts.php"></script> 定義ファイルの読み込み
</head>
<body onload="INTERMediator.construct()"> フレームワークをスタートさせる
<table>
    <tbody data-im-control="post"> この属性によりポストオンリーモードであることを示す
    <tr>
        <th>分類</th><td><input type="text" data-im="asset@category"/></td>  テキストフィールドに
        <th>名称</th><td><input type="text" data-im="asset@name"/></td>       ターゲット指定を記述する
        <th>メーカー</th><td><input type="text" data-im="asset@manifacture"/>
        <th>型番</th><td><input type="text" data-im="asset@productinfo"/></td>
        <th>取得日</th><td><input type="text" data-im="asset@purchase"/></td>
        <th></th><td><button data-im-control="post">新規入力</button></td> サブミットボタンに相当
    </tr>
    </tbody>
</table>
</body>
</html>
```

ポストオンリーモードのフレームワーク側の処理の概要を示す．エンクロージャーとして識別される要素に data-im-control 属性を記述し値を「post」とする．これにより，このエンクロージャー内は入力専用であると認識するので，この例のコンテキストである asset テーブルへのクエリーは実行されない．エンクロージャー内では，data-im-control 属性の値が「post」である BUTTON タグ要素も配置する．チェックボックスやラジオボタンの利用も可能であり，マスターテーブルから取り出した値をポップアップメニューに表示することもできる．ボタンをクリックすることで，エンクロージャー内のターゲット指定のあるフォーム要素から値を取り出し，それらをサーバーに送付し，SQL データベースであれば INSERT ステートトメントを実行して，新規レコードを作成する．

5.4.4　入力確認ページをサポートしない理由

この種の入力フォーム的なアプリケーションでは，最初にフォームで入力をして，それを確認するページを表示して，最終的にユーザーが確認してボタンを押せば完了するという流れが一般的である．INTERMediator のポストオンリーモードを使ってこの種のユーザーインタフェースを構築することは不可能ではないが，同一ページにフォーム部と確認部を用意して，それぞれ表示/非表示をコントロールすれば良い．ただし，JavaScript によるプログラムの作成が必要になる．こうした仕組みを簡単に実現する方法が必要であるという考え方もできるが，入力専用フォームでの確認処理の是非を検討した結果，不要であると判断し，現状では実装はしていない．その理由を以下に示す．

入力専用フォームでの確認処理は，(A) 入力内容の確認，ないしは (B) 入力内容に基づく処理結果の確認の，2 通りが考えられる．前者はシンプルな申し込みやアンケート回答に対応し，後者は例えば通信販売のようなページで送り先から送料が決まりそれをもとに支払い合計金額が求められるような場合である．

(A) のタイプの入力専用フォームの場合，なぜ，確認ページがあるのかということを，さまざまな Web サイトに記載されている内容から検討した結果，次のような理由があることがわかった．

1 フォームのページでは入力した情報がすべて見えているとは限らない．

2 Return キーや Enter キーによって，submit ボタンが押されたのと同じになり，意図せずサブミットしてしまうときにやり直しが効かない．

3 積極的な理由はない．一般にそうだから，あるいは，とにかく確認させたい．

[1] の理由は，確かに 10 文字分しかないテキストフィールドに 30 文字を入力するような状況では発生する可能性がある．しかしながら，入力する内容にもよるが，入力結果がその場で見えないのはデザインに問題があるとも言える．長い文章の場合には TEXTAREA 要素が使われるが，現状のブラウザではユーザーが TEXTAREA 要素の大きさを変更できるようになっているのが一般的であり，長い文字列を入力するから見えなくなるというのは短絡的な意見である．仮に画面に収まりきれないようなフォームが必要だとしても，本来は分割して入力できるようにするなどの対処をすべきである．[A] の理由は，適切なデザインを行うことで解消できると考える．

[2] の理由は，キーボードショートカットによる操作性の向上を意図したものではあるが，キータイプに慣れている人ほど癖で Return キーを押してしまうことは確かにある．これは，FORM タグで囲ったフォームで発生する問題である．INTER-Mediator は FORM タグを一切使わないため，テキストフィールドで Return キーを押しても確定ボタンを押した処理は一切行われない．したがって，[2] の問題は一般的な FORM タグの問題であり INTER-Mediator では発生しない問題である．

[3] は心理的な問題であり，技術的な問題ではない．以上より，適切なデザインを行うのであれば，INTER-Mediator では確認ページは不要と言える．

ここで，(B) の場合を検討する．(B) のような形式のアプリケーションも,「入力フォーム」とひとくくりにされることが一般的であるが，実際には入力に加えて，入力した結果から求められた結果が追加されたものが確認ページで表示される．そこでは，入力結果の確認もあるかもしれないが，むしろ求められた結果の確認が主要な目的である．入力の確認は，(A) の理由と同じく適切なデザインにより確認ページは不要である．そして求めた結果の確認は，入力フォームではなく，処理結果の表示に置き換えてみる．

INTER-Mediator で実装するには，入力した結果でレコードを作成して，次のいわゆる確認ページは，入力したレコードから求められた結果を表示するようにする．例えば，通信販売ならば，品目や個数，送り先などを入力フォームで入力し,「確認」ボタンでレコードを作る．作成された販売レコードは,「未確定」「注文確定」「発送済み」「キャンセル」などのステータス管理をする．レコードを新規に作った時には「未確定」にして，そのレコードをユーザーに提示し，キャンセルかあるいは注文確定させるボタンを用意する．これらのボタンは，レコードの特定のフィールドを更新するのが主要な作業となる．このように考えれば，(B) のタイプのアプリケーションの確認ページは，レコードの表示のページで構築する方法をとることができる．ただし，この種のアプリケーションでは要求が複雑になることが一般的で，画一的な解決方法が適用できないことが多い．結果的に JavaScript でのプログラミングを併用しないと，要求を満たされないことになる．つまり，アプリケーションとしては複雑な部類に入ると言える．

結果的に，従来の FORM タグを利用した Web ページのような「確認」のページをワークフローに入れる必要性は，心理的な抵抗感という以上の理由はないと言える．その結果，フレームワークがサポートする機能としての優先度は非常に低くなり，現在でも実装していない．

5.5　フィールド単位のデータ変換

フィールド単位でのデータ変換の仕組みがあり，日付のデータの変換などに利用できる．図 3.1 に示した「フィルタクラス」において，レコードの中の 1 つのフィールドに対する変換処理を記述できる．例えば，データベースから「2013-02-24」という情報が得られれば,「2013 年 2 月 24 日」という情報に変更する．一方で，テキストフィールドで編集して更新したときには逆の変換を行う．日付以外に数値のフォーマットやテキストを前後に付加するもの，HTML 文字列に関するものが用意されている．リスト 4.4 に示すような記述で，定義ファイルに記述を行う．

変換処理はサーバーサイドで行う．クエリーの場合はデータベースから取得したデータに対して適用する．更新の場合は，データベースエンジンにコマンドを与える前にクラスの機能を利用して変換を行う．

5.6　ラジオボタン，チェックボックスの実装

ラジオボタン，チェックボックスともに，INPUT タグ要素で実現されるユーザーインタフェースを構成する部品である．データベースとのバインドにおいては，タグ要素内で value 属性で指定された値を基準にチェックが入るかどうかを判断する．例えば，チェックボックスで value 属性が 1 の場合，データベースから得られた値が 1 であればチェックが入る．一方，1 以外の場合にはすべてチェックが入らないようにする．従って，フィールドに入力される値と，ページファイル内のタグ要素の value 属性を一致させないといけない．データベースへの更新を行うときには，チェックがオフからオンの場合は，value 属性値をフィールドに書き込む．オンからオフの場合は，データベースエンジンに応じて，空文字列あるいは NULL を書き込む．

ラジオボタンの場合は，複数の INPUT タグ要素がセットになって，同一のフィールドとバインドする必要がある．チェックボックスと同様に value 属性に設定した値と，データベースから得れた値が同一であれば，そのボタンが選択される．ラジオボタンが複数ある場合には，それぞれのタグ要素の value 属性値は異なるはずである．したがって，クエリー結果のバインドは，チェックボックスと同様，それぞれのタグ要素に対してフィールドの値と同一かどうかを判定すれば良い．ラジオボタンでは，複数のボタンをセットにするため，同一の name 属性を与える必要がある．ページファイル上では，同一の name 属性を指定する．テンプレート上では 1 組のラジオボタンであるが，複数のリピーターとして展開されると，レコードをまたがって同一の name 属性を持つラジオボタンのセットになってしまう．そのため，リピーターを複製してバインドするときに，name 属性に番号を与えて，レコードごとに name 属性値が異なるようにしている．

5.7　画像などのメディアの利用

画像を表示するために IMG タグ要素を使用したり，ビデオ等へのリンクのための A タグ要素についての取り扱いを説明する．これらテキスト以外の要素を「メディア」と総称する．認証を必要としないサイトで，画像自体がファイルとして公開しているような場合には，一般にはそのファイルのファイル名や部分パス名等がテキスト型のフィールドに設定されることになる．例えば，path フィールドに「img/picture.jpg」といったテキストデータが記述される．ここで，定義ファイルに name キーが「photos」のコンテキストで，

path フィールドの情報が得られるとする．リスト 5.2 のようにページファイルでの記述を行う.「path@#src」は「path フィールドのデータを src 属性の最後に追加する」という動作を行う．そのため，ここでは src 属性が「http://mediasite.msyk.net/data/img/picture.jpg」となり，この URL に画像ファイルが存在すれば，それがページ上で見えることになる．

リスト 5.2: テキストで得られるパスを利用した画像表示

```
<table>
    <tbody><tr><td>
        <img src="http://mediasite.msyk.net/data/" data-im="photos@path@#src" />
    </td></tr></tbody>
</table>
```

この方法以外に，プロキシを通じてメディタデータを取得することもできる．プロキシは INTER-Mediator の動作の 1 つであり，URL のパラメータの与え方によってはメディアのプロキシとして動作する．この仕組みが必要な理由は，認証や認可を経てメディアデータを取得できるようにするためである．認可と認証に関わることは，5.9 節で解説する．メディアプロキシとしての動作をさせるには，定義ファイルの 2 つ目の引数に media-root-dir キーの値を記述する．値は運用しているサーバー上の絶対パスを指定する．そして，IMG タグ要素の src 属性などが,「定義ファイルへの URL?media=パス」の形式になっていれば，media-root-dir キーで与えたディレクトリから，パラメータに指定した media キーの値を相対パスとして得られたファイルの内容を，MIME タイプ等を付与した HTTP で返す．

5.8 マルチクライアントでの更新処理の伝達

Web アプリケーションは，サーバーへのリクエストとレスポンスだけで成り立っている．Web の初期の頃より「ページを更新するにはブラウザの更新ボタンを押す必要がある」という制約が常識化しており，通信処理を最小限にするためにも，必要に応じて手作業で更新するということは許容範囲となっていた．この考え方は現在でも残っている．しかしながら，共有しているデータが更新されたとき，その更新結果が他のクライアントにも多少の遅延があっても支障のない範囲内で更新されることは，決して不要な仕組みではなく，むしろそれが必要な場合もある．初期の Web アプリケーションでは複雑なことや回線が細いことなどを理由にそうした実装はあまりされなかったが，Ajax の進展により基本的な仕組みは備わった．

クライアント間で更新されるデータの共有を行う手法は，それぞれのクライアントが保持するドキュメントオブジェクトの同期を行うことである．インターネット以前の時代より，グループウエアの研究 [32] によってアルゴリズムが考案され，Operational Transformation[95]（操作変換）として一般化されている．クライアントサイドで稼働する JavaScript で開発されているフレームワーク上で実現されており [50]，オブザーバブルなモデルをベースにして，モデル間での同期を行うことで，共有データの更新が通知される仕組みが提唱されている．モデルを経由した更新を実装することで，モデルの特定のフィールドと，いくつかの要素がバインドされることになる．その結果，同一クライアント内でも，同一のフィールドからのデータ表示を更新するといったことも実現する．

INTER-Mediator での実装にあたっては，Operational Transformation のアルゴリズムの実装やあるいは既存ライブラリの利用を検討したが，複雑な構造を持つドキュメントではなく，データベースはシンプルなリ

レーションである．また，競合の問題は，楽観的なロックによる手法をすでに組み込んでいる．結果的に，更新や新規レコード作成，レコード削除を通知するという実装で小規模な業務システムのような利用は可能ではないと考えて実装を行った．ただし，モデル部分の同期という考え方は持ち込んだ．コンテキストの定義に対応して，実際にクエリーにより得られた結果を保持するコンテキストモデルを内部で管理し，更新をモデル経由で行う実装を行った．しかしながら，実運用上の問題点が今後は出てくる可能性もある．そのときには，Operational Transformation のアルゴリズムが参考になる．

データベースのクエリーをしたときに，テーブルと各レコードの主キー値情報を，サーバー側のデータベースに記録する．同時に，クライアントごとに一意な識別番号を生成して付与し，テーブルおよびレコード情報とともに保持する．別のクライアントが同一のクエリーを実施した場合も同様に，テーブル，各レコードの主キー値，クライアントの識別子を記録する．ページを閉じるときには，サーバーに記録されたこれらの情報をクリアする．

そして，あるクライアントであるフィールドを更新した場合，そのレコードをクエリーにより取得したクライアントに変更があったとことを通知する．ただし，自分自身には通知はせず，クライアント内部での通知の伝達はそのクライアント内で実施する．コンテキストモデルは，更新，新規レコード，レコード削除に対する応答を行い，その結果を最終的には要素の更新にまで波及させる．

通知の伝達には，Pusher[82] というサービスを利用した．WebRTC[9] を利用した通知サービスであり，開発者向けには無償で使える点があることと，ライブラリの利用が極めて容易である点から採用した．一定規模以上は有償になるが，桁外れの価格でもないため，当初の実装ではこの仕組みを利用することにした．

以上のような方針のもとで実装を行った結果，Pusher を利用するための ID 番号をいくつか設定するだけで，クライアント間での変更結果の同期が実現している．追加で必要なことは，データベースに，管理用のテーブルを追加するだけで，機能を利用するための追加の手続き的なプログラムは必要ない．INTER-Mediator が目指す開発モデルに合致した機能の組み込みが実現できた．

5.9　認証と認可の実装

INTER-Mediator は認証や認可を伴うアプリケーションの開発を，宣言的な記述のみで行える．アカウントは，データベースに用意したテーブルを利用するか，あるいはデータベースエンジンに登録したユーザーを利用できる．コンテキストやあるいは定義ファイル全体に対して認証を必要とするような設定を行うと，そのコンテキストに関わるすべての処理において，正しいパスワードを元にしたハッシュ値を含めた応答が必要となる．

5.9.1　認証と認可に必要とされる機能

Web アプリケーションで必要とされる認証や認可の機能に関して，INTER-Mediator での実装状況を表 5.2 にまとめた．

表 5.2: 認証と認可に組み込まれた機能

分類	機能	対応状況	補足
認証	ログインパネル（ブラウザ）	Web サーバーで実現	HTTP 認証下でのフレームワークの稼働は可能
	ログインパネル（HTML）	フレームワークが用意	標準のログインパネル以外にカスタマイズも可能
	ログアウト	ページネーション上に表示	API を JavaScript で呼び出すことでも可能
	ユーザー記録データベース	既定のフィールド名で用意	サーバー側でデータベース処理を実行するときに利用
	認証処理・エラー処理	サーバー側で判定	クライアント側では繰り返し認証を求めるなどの処理を実装
	ログイン状態の継続	定義ファイルに指定	JavaScript の変数あるいはクッキーで保持
	データベースのユーザーでログイン	定義ファイルに指定	データベース側に設定した認可の設定を利用できる
	ユーザーがパスワードを変更	可能	標準のログインパネルに用意
	メディタデータの取得	定義ファイルに指定	認証されたアクセスのみに設定可能
ユーザー管理	ユーザー管理システム	サンプルに用意	ユーザーの登録やグループ作成，割り当てなどが可能
	新規登録	サンプルに用意	サンプルの「ユーザー管理システム」に機能を用意
	パスワード変更	サンプルに用意	サンプルの「ユーザー管理システム」に機能を用意
	メールアドレスの利用	ファイルに指定	ユーザー ID 以外に登録時のメールアドレスでのログイン
	サインアップ	サンプルで対応	メールアドレスに応答することでパスワードを発行
	パスワードリセット	サンプルで対応	パスワードリセットをユーザーへのメールと連携して実施
認可	グループ管理	データベースを利用	グループおよび所属情報のテーブルを定義して運用できる
	グループのロール	ファイルに指定	CRUD 個別あるいは全部に対してアクセス権限の設定ができる
	適用範囲	定義ファイルに指定	ページ単位，コンテキスト単位，レコード単位
	メディタデータの認可	定義ファイルに指定	メディアアクセス専用トークンを発行
セキュリティ	パスワード漏洩対策	ハッシュ値の利用	データベースにはソルトを付与したハッシュ値で保存
	セッション対策	チャレンジ-レスポンス	データベースのアクセスごとにチャレンジを受け取り，次の処理でチャンレンジをもとにパスワードの暗号化を行う．チャレンジは通常は 1 のみの利用
	データベースのユーザーのパスワード送信	非対称鍵で暗号化	サーバー側に用意した公開鍵をクライアントに送り，パスワードを暗号化して送付
	更新不可のフィールド	定義ファイルに指定	主キーや外部キーなど書き換えしたくないフィールドの指定
	読み出し不可のフィールド	定義ファイルに指定	

5.9.2　コンテキストで認証を必要にする

Web アプリケーションで認証を行い認可を適用させるには，定義ファイルに設定を記述する．コンテキストに記述することでコンテキストごとに異なる設定が可能だが，定義ファイル内のすべてのコンテキストに共通の設定を適用することもできる．コンテキストごとの設定例をリスト 5.3 に示した．設定は authentication キーに記述する．定義ファイル全体に適用させる場合には，同様に authentication キーの記述を，定義ファイルに記述する IM_Entry 関数の 2 つ目の引数の要素に記述する．

リスト 5.3: コンテキスト定義での認証の設定例

```
array(
   'name' => 'message1',
   'authentication' => array( ),
),
array(
   'name' => 'message2',
   'authentication' => array(
      'all' => array( 'group' => array( 'admins' ), ), ),
),
array(
   'name' => 'message3',
   'authentication' => array(
      'load' => array( 'target' => 'field-user', 'field' => 'username'),
      'update' => array( 'user' => array( 'sysadmin', 'ceo' ), ),
      'new' => array( 'group' => array( 'admins' ), ),
      'delete' => array( 'group' => array( 'not-exist-group' ), ), ),
),
array(
   'name' => 'message4',
   'view' => 'message',
   'table' => 'not-exist-table',
   'authentication' => array(
      'load' => array( 'group' => array( 'admins' ), ), ),
),
```

name キーの値が message1 のコンテキストは，あらゆるデータベース処理において，認証を必要とする．ただし，認証されたユーザーはすべての処理が可能となり，認可についてはすべてが許可されるようになる．name キーの値が message2 のコンテキストでは，admins グループのユーザーだけがすべてのデータベース処理ができる．認証されていないユーザーや，認証されていても admins に所属していないユーザーはあらゆる処理が拒否される．

name キーの値が message3 のコンテキストは，CRUD の各操作において，異なる認可の設定をしている．load キーはクエリーの取得であり，この場合は message3 で参照されるテーブルにある username フィールドの値が，現在ログイン中のユーザーのユーザー名と同一のレコードだけが取得される．つまり，レコード単位での認可である．この仕組みは,「username = ’ ログインユーザー名’」という検索条件を，すべての検索条件式全体に対して AND で適用することで実現している．さらに update キーはレコードの更新であり，この場合は sysadmin と ceo という 2 つのユーザーにのみ更新が許可され，他の認証済みおよび認証されない接続に対しては更新は拒否される．new キーは新規レコード作成で，admins グループのみが許可される．delete キーはレコード削除の権限であり，この場合，意図的に「存在しないグループ名」を指定することで，誰も削除ができなくしている．load, update, new, delete で個別に設定はできるが，どれか 1 つを記述したときは他のものを省略すると省略した処理は認証しなくても利用できてしまうので，原則は 4 つの操作をすべて記述する必要がある．

name キーの値が message4 のものは，クエリーの対象は view キーで指定された「message」テーブルであり，更新，新規レコードおよびレコード削除を行うのが table キーの「non-exist-table」テーブルのように，読み出しと書き込みのエンティティを別々に指定している．「non-exist-table」テーブルは，その名前のテーブルがあるということではなく，現実には存在しないテーブル名を記述する．こうすれば，更新，新規レコードおよびレコード削除はすべて存在しないテーブルに対して行うためエラーになる．したがって，最小限の設定として，load キーつまりクエリー時の認証だけを設定することで，意図しないデータベース処理を防ぐことができる．

認証が必要になると，図 5.7 のような認証パネルが表示される．これは別のページにリダイレクトしているのではなく，現在のページのウインドウ上に，オーバーラップしてパネルを表示させている．読み込み時に認証が必要な場合には，ページを開いた直後に表示される．読み込みは認証が必要なものの修正時に認証が必要になると，修正直後にこのログインパネルが表示される．ログイン状態は時間を決めて保持したり，あるいは保持できないようにも可能である．

図 5.7: コンテンツサイトの認証画面

以上のような仕組みの認証に構造的な欠陥がないかどうかは重要な点である．JavaScript の場合，クライアントサイドのグローバル変数は，ブラウザのアドレスバーから変更できてしまうため，クライアント側で保持したデータは改変される可能性がある．もちろん，改変の度合いによっては動作不良になる可能性もあるが，それはすべての JavaScript プログラムが持つ問題点でもある．

そうした改変がなされたとしても，認証や認可の限定範囲を変えられないことが必要である．そのことを担保する手法として，すべての設定と処理をサーバー側で完結させるようにし，クライアントサイドは結果を得てユーザーインタフェースを提供するだけに留めた．定義ファイルはサーバーにあり，認証の設定や検索条件式の適用はサーバーサイドで行う．クライアントサイドでのグローバル変数の値を変えるようなことが仮にあったとしても，認証の動作自体は変更できないため，意図しない認証認可の設定が与えられることはない．潜在的なバグがあればそこはセキュリティホールになる可能性があるが，バグがないとすれば記述通りに動作する．

5.9.3　認証の仕組み

アカウントはテーブルに記録できる．テーブル名はデフォルト値は決めているが，異なる名前のテーブルも指定できる．一方，フィールド名は決められたものを利用する．既存のデータベースがある場合は，ビューを作成してフィールド名やテーブル名を INTER-Mediator の規則に合わせることで対処できる．表 5.3 は，用意するテーブルとそのフィールドである．「ユーザー情報」と「チャレンジ記録」は最低限必要である．グループを利用する場合には 4 つのテーブルが必要になる．「対応付け」のテーブルは，ユーザーあるいはグループを，グループに対応付けるための譲歩を保持する．ユーザー情報およびグループ情報は，対応付けのテーブルでは，主キーの id フィールド値を利用する．user_id フィールドあるいは group_id フィールドのどちらか一方のみ，実在する主キー値を入力し，もう一方は NULL のままにする．dest_group_id フィールドはそのユーザーあるいはグループが所属するグループの id フィールドの値を記述する．「チャレンジ記録」については 5.9.4 項で解説する．

表 5.3: 認証に必要なテーブルとその構造

テーブルの種類 (既定の名前)	フィールド名	型
ユーザー情報 (authuser)	id	整数（主キー）
	username	文字列
	hashedpasswd	文字列
	email	文字列（オプション）
グループ情報 (authgroup)	id	整数（主キー）
	groupname	文字列
対応付け (authcor)	id	整数（主キー）（オプション）
	user_id	整数
	group_id	整数
	dest_group_id	整数
チャレンジ記録 (issuedhash)	id	整数（主キー）
	user_id	整数
	clienthost	文字列
	hash	文字列
	expired	タイムスタンプ

5.9.4　認証のためのプロトコル

クエリーを行う時のプロトコルを，図 5.8 にシーケンス図で示した．その他のデータベース操作も，同様な手順になる（以下の丸数字はメッセージの番号）．

①ページの合成を開始し，②内容に応じてデータベースのデータを要求する．最初は認証がないものと仮定してアクセスをする．③クライアントがサーバーに通信を行い，④サーバー側ではデータベース処理を行う前に認証や認可の適用があるかどうかを調べる．ここでは認証が必要だが，ログイン処理を一切行っていないので，エラーを返す．⑥通信モジュールは認証が必要であることを示すエラーを受けて，⑥フレームワークにログインパネルの表示を促し，⑦ログインパネルが表示され，⑧ユーザーはログインパネルを見る．

⑨ユーザーがログインパネルで正しいユーザー名とパスワードを入力し，⑩「ログイン」ボタンをクリックしたとする．⑪フレームワークはログインパネルを消去し，⑫から⑭までの一連の流れで，サーバーからチャレンジとクライアント ID，そしてログインしようとしているユーザーのソルトを取得する．それをも

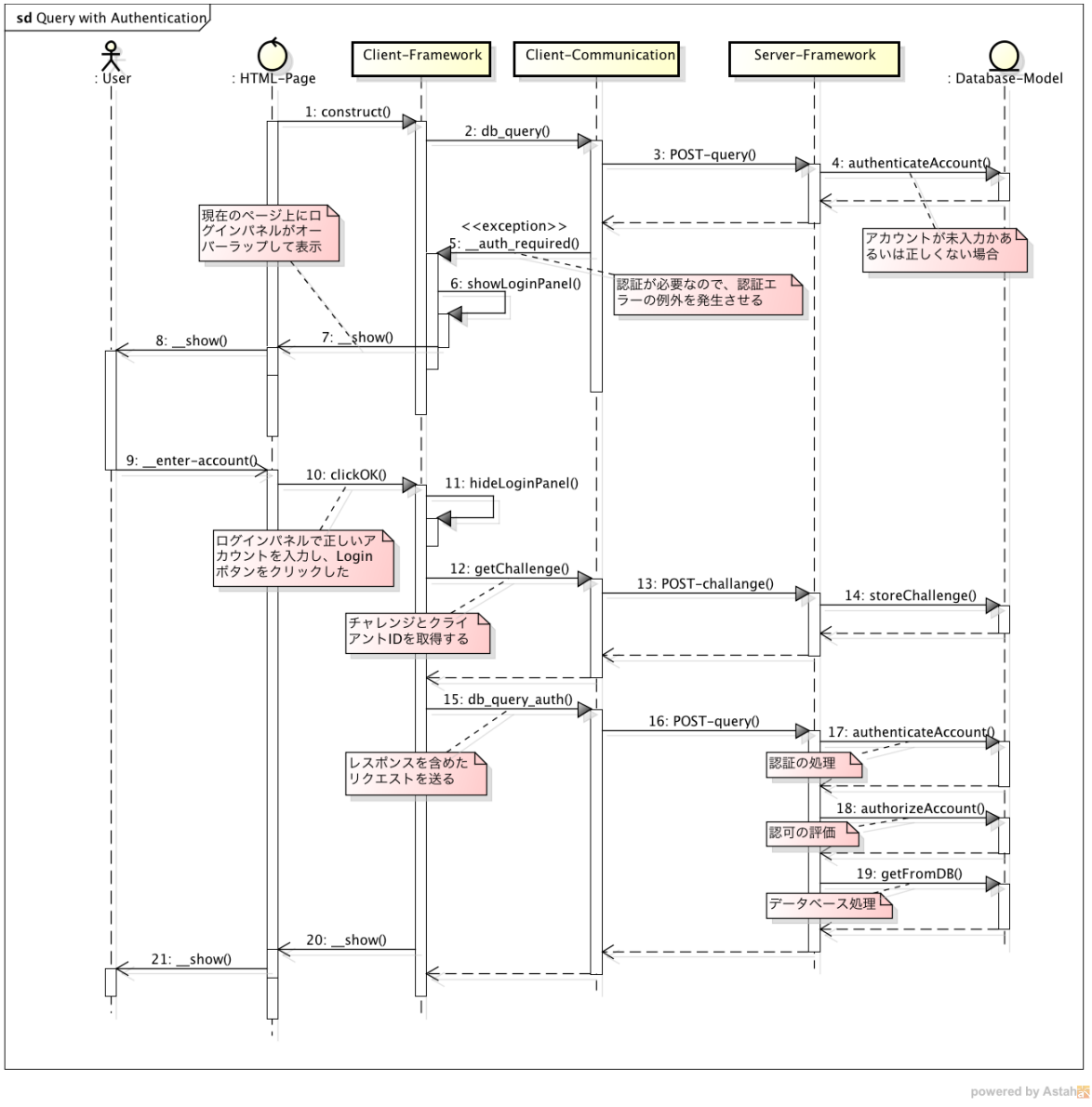

図 5.8: 認証や認可を伴うデータベースアクセス

とにして，⑮より実際のデータベースアクセス処理を行う．サーバーサイドで⑰認証が成功し，⑱認可により許可された状態なら，⑲データベース処理を行うという流れになる．データベース処理を行った結果には，新たなチャレンジが得ており，次回のデータベースアクセスはそのチャレンジを利用するので，⑫から⑭のチャレンジの取得のみのアクセスは，認証直後の 1 回だけになる．

認証に絡んだデータの以下に説明する．まず，各データを以下の変数名で記述することにする．

$$
\begin{aligned}
user &= \text{ユーザー名} && (5.1)\\
pw &= \text{本来のパスワード} && (5.2)\\
salt &= \text{ユーザーごとに異なるソルト（4 バイトの } ASCII \text{ 文字）} && (5.3)\\
hpw &= \text{データベースに保存されているパスワードのハッシュ値} && (5.4)\\
pw' &= \text{入力したパスワード} && (5.5)\\
cid &= \text{クライアント } id && (5.6)\\
ch &= \text{サーバーから送られるチャレンジ} && (5.7)\\
res &= \text{クライアントから送られるレスポンス} && (5.8)\\
res' &= \text{サーバー側の情報から得られるレスポンスの期待値} && (5.9)\\
SHA1(m) &= SHA-1\text{[72] による } m \text{ のハッシュ値を求める関数} && (5.10)\\
HMAC(k,m) &= HMAC-SHA256\text{[6][73] で求められる } MAC \text{ 値を求める関数} && (5.11)\\
&\quad \text{ハッシュ関数は } SHA-256\text{，鍵が } k\text{，メッセージが } m && (5.12)\\
\text{+演算子} &= \text{文字列の結合} && (5.13)
\end{aligned}
$$

データベースに保存されるハッシュ値 hpw は，本来のパスワード pw，ソルト値 salt より以下の式で求めたものを利用している．ソルトが ASCII 文字であるのは JavaScript 側での計算が必要になる場合があり，文字列処理の関係上 ASCII 文字のみとした．

$$hpw = SHA1(pw + salt) + salt \tag{5.14}$$

表 5.4 には，認証に関わるチャレンジとレスポンスの手順を示した．チャレンジ要求があると，そのユーザーの hpw の最後の 4 バイトより salt が求められる．最初のチャレンジ取得ではクライアント id とチャレンジ値は両方とも乱数で求められる．クライアント id はその後のやりとりでは同一のものを利用する．user，cid，ch，現在の日付時刻を表 5.3 に示した「チャレンジ記録」のテーブルに新たなレコードを生成して残しておく．salt，cid，ch をクライアントに返す．

クライアント側では次の式でレスポンスを求める．サーバーから返された値以外は，ユーザーが入力したパスワードを使用している．

$$res = HMAC(SHA1(pw' + salt) + salt + cid, ch) \tag{5.15}$$

表 5.4: 認証のために転送されるデータと動作

クライアント側での処理	転送される認証情報	サーバー側での処理
チャレンジ要求	→ user →	データベースより hpw を取得し salt を求める ch を乱数より生成，cid が未発行なら乱数で生成 user, cid, ch, 日付時刻をデータベースに記録
res を求める	← salt, cid, ch ←	
データベース要求	→ cid, user, res →	データベースより，cid からチャレンジを取得

データベース要求に，res および cid，user を含めてサーバーに送信する．受け取ったサーバー側は，user と cid の値を元に,「チャレンジ記録」テーブルより検索し，レコードが存在し，指定された時間内での応答であるかどうかを確認する．条件に合えば，チャレンジ記録にある該当するレコードは削除される．そして，以下の式で求めたレスポンスの期待値 res' とクライアントが返した res が一致していれば認証したとみなす．

$$res' = HMAC(hpw + cid, ch) \tag{5.16}$$

$$= HMAC(SHA1(pw + salt) + salt + cid, ch) \tag{5.17}$$

データベース処理した結果を返すときには，チャレンジ要求で行ったことと同一の処理を行う．ただし，cid は要求に含まれたものをそのまま利用する．こうして，次回のデータベース要求のためのチャレンジを応答の段階でクライアントに送り届ける．

ネットワーク上を飛び交うデータは，user，salt，cid，ch，res であり，パスワードから求められるものは res のみである．しかしながら，ハッシュ値なので，res からはパスワードは合理的な時間内での解析はできない．MAC 値を求める鍵は，パスワードをもとに求められるものなので，交換済みの秘密鍵としての機能がある．一度レスポンスを返したチャレンジは 2 度と使われない．チャレンジの値が毎回違うので，レスポンスの値も毎回異なり，これらの値をネットワーク経路上などで横取りをして同じリクエストを送ったとしても，そのときはチャレンジは無効になっている．従って，パスワードを得ない限りは認証処理を通すことはできない．

5.9.5 認証が必要な場合にログインパネルを表示する仕組み

必要ならばログインパネルを表示し，その後にデータベース処理を再度行うという仕組みの実装方法について説明する．図 5.4 は，既存の construct 関数を，認証を適用しユーザーインタフェースを適切に表示する construct 関数に変更した結果である．JavaScript の例外処理と関数オブジェクトの機能を使って既存の関数を認証対応にラッピングするような実装を行った．元の construct 関数の名前を construct_main に変えておく．construct_main 関数の呼び出しで認証が正しくされていない場合は認証エラーが例外で返ってくるのをキャッチして，その場合にログインパネルを表示する showLoginPanel 関数を呼び出す．showLoginPanel 関数の引数は，パネルで「ログイン」ボタンをクリックしたときに呼び出される関数オブジェクトを指定するが，そのときにもう一度同じ条件で construct 関数を呼び出す．認証エラーがあれば，何度もログインパネ

ルが表示され，認証結果が正しければ続けて処理を行うという仕組みを実現できる．認証エラーの回数をカウントし，一定以上であれば別のエラーを出すことで，繰り返される呼び出しを終了させることができる．

リスト 5.4: 通信処理関数をベースに認証付きの通信処理関数を作る

```
function construct(x)  {
   try {
      construct_main(x);   //元からある処理の呼び出し
   } catch (exception) {
      if (exception == AUTH_EXCEPTION) {
         showLoginPanel(
            function()  {  construct(x);  }
         );
      } else {
         //その他のエラー処理
      }
   }
}
```

5.9.6　メディアに対する認証・認可

メディアデータについては 5.7 節で説明した通り，プロキシを経由したアクセスをサポートしている．メディアデータについても認証を行う場合，プロキシを利用した上で，ここまでに設定してきた設定を行う．加えて，コンテキストの authentication キーの media-handling キーの値を true にする．

サーバーとクライアント間のやりとりで，随時チャレンジとレスポンスを返す方法で認証を行っているが，画像等のメディアデータについては，この方法が取れない．1 回のアクセスで複数レコードが得られるが，その後，メディアデータのプロキスのアクセスは何回行われるかわからないことと，ブラウザによって並行的に行われることもあり，異なる認証の方式が必要になる．あるコンテキストにアクセスすると，media-handling キーの値が true なら，メディア用の認証トークンをサーバーはクライアントに渡し，クライアントはその値をクッキーに記録して，メディアアクセス時にサーバーに送信できるようにする．サーバー側はそのクッキーで記録された値を参照して，正しいトークンならば認証を許可する．ただし，複数回のアクセスで同じトークンが使われるので，通信経路の傍受によって，なりすましによるアクセスは可能である．したがって，メディアデータの認証が含まれる場合には TLS の利用が必須となる．また，トークンは認証継続時間の指定にしたがって，無効になる．

レコード単位の認可の設定も，メディアデータに対して適用される．そのためには，メディアデータのパスの中に,「コンテキスト名/主キーフィールド名=値/ファイル名」という記述が必要であり，実際に記述されるパスにファイルが存在するようにする必要がある．

5.10　上位概念の機能の実現

この節では，アプリケーションでよく利用されるような一覧と詳細を行き来するようなユーザーインタフェースを作成する機能や，検索機能を宣言的な記述のみで実現する機能，メール送信の機能をどのように実装しているのかについて解説をする．

5.10.1 一覧と詳細を行き来するユーザーインタフェース

データベースに蓄積されたデータを参照する場合，各レコードの主要なフィールドのみを一覧表示し，選択したレコードに関して必要なすべてのフィールドを表示するという形式のユーザーインタフェースはよく利用する．それぞれ一覧と詳細と呼ぶが,「マスター/ディテール形式」とも言われる．iPad を横長に使用した時に，一覧と詳細が左右に並び，一覧で選択したレコードが詳細に表示される形式もある．

INTER-Mediator では，一覧と詳細のユーザーインタフェースを，宣言的な記述のみで実現する．原則として同一のテーブルをもとにした，2 つのコンテキストを用意し，それぞれに navi-control キーを指定する．一覧側のコンテキストの navi-control キーには「master」ないしは「master-hide」と記述する．詳細側のコンテキストの navi-control キーには「detail」ないしは「detail-bottom」と記述する．

ページファイル内に，一覧と詳細のコンテキストを利用した 2 つの TABLE タグのテーブル（あるいは 2 セットのエンクロージャー/リピーター）を記述する．一覧と詳細が切り替わるタイプは，一覧側のコンテキストの navi-control キーに「master-hide」を指定する．すると，ページ合成後に，フレームワークは自動的に各リピーター内に詳細を表示するボタンを配置する．同時に詳細部分は非表示になる．詳細を表示するボタンをクリックすると，一覧部分が非表示になり，詳細部分が表示される．同時に，クリックしたレコードが表示されるように，エンクロージャーを更新する．詳細側のコンテキストは navi-control キーの値が「detail」に設定されており，ページテンプレートからの展開時にそのコンテキストのリピーターの前に，「detail-bottom」であればリピーターの後に，一覧を表示するボタンを配置する．そのボタンをクリックすると，詳細と一覧の表示/非表示を切り替える．

iPad の形式の表示を行う場合は，一覧側のコンテキストの navi-control キーに「master」を指定する．一覧も詳細も最初はどちらも見える状態になっているため，CSS の設定を利用して，2 つのエンクロージャーによる展開領域を左右に並べるような配置にしておく．この場合，一覧側に詳細を選択するボタンは表示されるが，詳細から一覧に戻るボタンは配置されない．一覧側でボタンをクリックすると，詳細側に表示されるレコードがボタンに対応したレコードに切り替わる．

5.10.2 検索処理を実現する仕組み

テキストフィールドに検索条件を入れると，そのキーワードを含むレコードが検索されて一覧されるといったユーザーインタフェースへのニーズは高い．コンテキストに検索条件を記述できるが，これは固定的な条件になる．そこで，当初は JavaScript の API を利用し，決められたプロパティにオブジェクトとして検索条件を代入しておくことで，検索時にその条件を適用したクエリーを発行できるようにした．ボタンを押した時に，検索条件のテキストフィールドから条件を取り出し，適切な演算子とともにオブジェクトを生成し決められたプロパティに与え，ページ全体を更新するなどして検索結果を表示できるようにした．しかしながら，ここでは数行ながらも手続き的なプログラミングが必要になる．そこで，プログラムを不要にする実装ができないかを考えた．

そのためのベースになる仕組みとして，データベースとはバインドしていない，ブラウザ側に存在するキー/バリュータイプのストレージ（ローカルコンテキスト）を用意した．そのストレージは，コンテキス

ト名の代わりに「_」を記述する．また，フィールド名がキーとなり，ローカルコンテキストはページファイル内のノードとバインドする．

検索条件のテキストフィールドの data-im 属性に，例えば「_@condition:postalcode:f3,f7,f8,f9:*match*」と記述する．最初の_はローカルコンテキストを示し，@以降はローカルコンテキストのキー名である．キー自体はコロンで区切ってパラメータとしている．最初の「condition」はこのキーに対する値が検索条件であることを示す固定のキーワードである.「postalcode」は適用するコンテキスト名を指定する．次の「f3,f7,f8,f9」はフィールド名を示すが，複数のフィールドをカンマで区切ることで，各フィールドに対する条件式の OR を条件として付加する．最後の「*match*」はフレームワークで規定した演算子であり，ここでは部分一致になる演算子をデータベースエンジンに応じて適用する．指定されたコンテキストに対するクエリーを実行するとき，ローカルコンテキストを探してこの設定があれば，検索条件を追加する．このテキストフィールドの入力値が「新宿」なら，MySQL をデータベースエンジンに使っている場合,「WHERE ((f3 LIKE '%新宿%) OR (f7 LIKE '%新宿%) OR (f8 LIKE '%新宿%) OR (f9 LIKE '%新宿%))」という WHERE 句を追加する．

同様な仕組みで，ある要素の data-im 属性を「_@addorder:postalcode:f3:asc」とすれば，この要素をクリックすると，f3 フィールドの昇順でクエリー結果を得るようにもした．これにより，どのフィールドを基準にソートをするのかといったユーザーインタフェースを追加できる．また，data-im 属性が「_@update:postalcode」であれば，その要素をクリックすると postalcode コンテキストが更新されるようにした．つまり，検索条件の右にある「検索」ボタンをこの方法で追加できる．

これらの仕組みにより，検索を伴う Web アプリケーションについても，宣言的な手法で対処できるようになっている．

5.10.3　メール送信機能の実現

Web アプリケーションでは，データの入力後などに確認のメールが送られることも多い．Web サーバーでは，なんらかの準備をすることで，メールサーバーを起動することができる点に着目して，サーバー側の機能として，メール送信の仕組みを組み込んだ．メールサーバーは送信用としてのみ利用する．別サーバーに対して SMTP でメール送信を行うこともできる．

メールの送信は，特定のコンテキストに対するデータベース処理の後に行われる．新規レコードを作成するとき，レコードを更新したとき，クエリーを実施したときに，それらの処理の後にメールを送ることができる．ただし，レコード削除後はサポートしていない．定義ファイルのコンテキストにメール内容に関する設定が行え，例えば，宛先に固定の値や，フィールドから得られる値を使用できる．メール本文は，テンプレートとなるテキストファイルを用意することができる．テキストファイル内には文字列のプレースフォルダを記述することができ，新規レコードを作った場合だと，その作ったレコードの各フィールドを，プレースフォルダに割り当てることができる．通信販売サイトにあるような，挨拶文と入力結果を含めたようなメールの作成が可能である．

5.11 プログラミングインタフェースのための拡張点

INTER-Mediator は宣言的な記述で多くの機能を実現できるようになっているが，どうしてもその範疇を超える機能の組み込みが必要になった時には，手続き的なプログラミングでの拡張が可能である．

5.11.1 クライアントサイドの拡張点

クライアントサイドでは JavaScript でのプログラムが可能である．ブラウザ上で稼働しているので，一般的な JavaScript によるプログラミングを行える．jQuery 等の JavaScript のライブラリを利用することもできる．極力，他のライブラリと競合するような動作をしないように配慮はしている．

一方，INTER-Mediator で展開したページは自動的に生成した要素が多く，実データはそこから得るというニーズが発生する．そこで，基準となる要素を指定して，ターゲット指定の文字列（data-im 属性の値）を指定して，要素に割り当てられた id 属性を得るメソッドを用意した．

検索条件やソート条件の動的な追加も，JavaScript で可能である．プロパティに決められたプロパティ名を持つオブジェクトを追加するなどして，条件を付与できる．

ページ合成処理の途中や最後に呼び出されるメソッドの定義も可能である．リピーターを追加した直後，エンクロージャーへのすべてのリピーターの追加が終わった後，さらにページ合成が終わった後や，値のチェックを行った後に呼び出すメソッドの定義もできる．

データベースから取り出したデータを保持するコンテキストモデルや，あるいはローカルコンテキストの処理も記述可能である．サーバー側モジュールと通信をして，データベース処理を行うメソッドも利用できる．

5.11.2 サーバーサイドの拡張点

サーバーサイドのモジュールは PHP で記述されており，PHP 言語を利用した拡張が可能である．1 つの方法は，データベース処理を行うクラスを独自に定義する方法がある．別の方法として，データベース処理の前後に，決められた名前のメソッドを実行するようなアスペクト指向的な手法での処理の拡張もできる．クエリーの場合だと，拡張するメソッドにクエリーで得られた結果が引き渡され，クライアントに返すデータを独自に構築できる．SQL データベースであれば，集計処理をしたビューの定義により集計は可能だが，FileMaker Server では集計結果の取り出しは効率的にはできない．そこで，集計元データを得て，集計結果を返すメソッドを定義して，集計処理を記述することで，複雑な処理への対処ができる．

実装においては，データベースエンジンごとのクラスを定義しつつ，クライアントからのリクエストはプロキシとなるクラスが受け付けるパターンを利用した [144]．プロキシが適切なデータベースドライバを選択し，拡張する処理プログラムの指定があれば，それを呼び出すように動作する．

5.11.3　JavaScript コンポーネントの統合

JavaScript で開発されたさまざまなコンポーネントは，アダプター部分を作成することで，INTER-Mediator でも使用できる．HTML エディタとして広く使われている TinyMCE や，ソースコードエディタの CodeMirror については，アダプターをサンプルファイルに含めてある．アダプターがあれば,「 」のように，data-im-widget 属性を記述することで，この場合は testtable コンテキストの text1 フィールドの文字列を，HTML エディタの編集文字列として表示し，編集結果はフィールドに書き戻される．

アダプター部分では，既定の初期化メソッドを定義し，その中で，決められたメソッド名のセッターとゲッターを記述する．初期化メソッドは，リピーター内部の展開時に行う処理と，すべてのページ生成処理が終わった後にまとめて行う処理の両方が記述できる．INTER-Mediator の処理をすべての JavaScript コンポーネントで共通に行うためのラッピングを行うことができる．また，さまざまなオプションを設定することも，このアダプター部分で行える．TinyMCE 用のアダプターで 90 行程度，CodeMirror 用のアダプターで 50 行程度である．いずれも，通常はページ内の TEXTAREA タグ要素に対して初期化をかけるが，アダプターでは，data-im-widget 属性がある要素の子要素として TEXTAREA タグ要素を追加し，初期化の処理をすすめるなどしている．

5.12　セキュリティ面への対策

INTER-Mediator でのセキュリティに対する配慮は，一般的なフレームワークと基本的には変わりない．SQL インジェクションを防ぐために，PDO が提供するクォート処理を実装している．また，XSS に対処するために既定の状態では，ページ合成ではテキストはすべてテキストノードとして処理をする．

JavaScript の eval 関数も一切使用していない．計算式のパースと計算処理はオープンソースで得られたパーサー [23] を利用して，独自に機能拡張して開発し組み込んである．

JavaScript のセキュリティ面では，innerHTML プロパティへの代入もセキュリティホールを発生させる可能性がある．INTER-Mediator では開発者が意図的に innerHTML を指定しない限りは属性ないしは下位のテキスト要素としてデータベースから得られた値を追加する．指定を可能にしているのは，CMS のような HTML のテキストをデータベースに記録して，それをページ上に表示するような用途が存在するからである．セキュリティを配慮した上で innerHTML の使用を許可するように，開発者に配慮を呼びかけている．

ユーザー認証や認可の機能も持っており，CRUD の各操作およびレコード単位での認可の管理までできる．認証や認可に関する設定はサーバー側でのみ評価されるようになっており，クライアントからのアクセス等で変更することは一切できないようになっている．

INTER-Mediator では，クライアントからのリクエストをフレームワークが生成するため，ブラウザのセキュリティ対策が完全ではない場合，それと同等な処理が HTML の改変等によって行われる可能性もある．しかしながら，仮にできたとしても，認証・認可の設定はクライアント側からのリクエストでは一切変更ができないため，認可の設定によって排除することができる．例えば読み出ししかしないテーブルへの利用は，更新などの残りの処理は存在しないグループにだけ権限を与えるといった手法で意図していないアク

セスを排除できる．

5.13 フレームワークに至る着想点

フレームワークの実装について，技術的な手法や工夫点はこれまでに説明した通りである．ここでは，技術的な面ではなく，開発を思い立ったきっかけや，開発前に検討したこと，あるいは開発しながら考えた方向性などを説明する．

5.13.1 テンプレート処理の着想を得た開発案件

INTER-Mediator を開発するきっかけになったのは 2009 年に受注した Web サイトの開発にさかのぼる．FileMaker をデータベースに使い，CodeIgniter をフレームワークとして利用することになったが，設計書類は HTML で作られたモックアップの画面ショットに注釈をしたものであった．顧客と筆者の間に SI 業者が入り，SI 業者が仕様を作成した．システムは，勤怠管理と業務に連動した発注元への請求処理である．スタッフの稼働時間に応じた請求を行う点では一般的なものだが，発注の単位と請求の単位が対応付けられていないため，発注と請求は多対多の関係になる．それらを結合するのが，個別のスタッフが特定の日時に働いた記録である．スクラッチから開発したものではなく，既存のデータベースを元にした開発の必要があった．モデリングの上では改良すべき点はあったものの既存の機能を損なわないためにデータベースの構成を変えることはできないため，そのままの状態で Web アプリケーション部分を作り込んだ．

本案件の本質的な問題点は，仕様の一部が「不明」のまま開発に入らざるを得ない点であった．ある機能について，説明を受けた SI 業者が理解できず，仕様化できなかった．開発して動きを見ながら探るという方針が立てられ，ある程度作ったところで全貌が見えてきたが，非常に複雑であった．要約すれば，1 アクションを受けてそれに関連するレコードを，ページ内の複数の異なる形式のリスト表示部分に追加したり，あるいはそのアクションのもとになっている項目をキャンセルすることで，複数のリストから関連するレコードを削除するといった仕組みを作成する必要があった．

関連レコードの付与と削除ということで，アクションによりサーバーでリンク設定をしてページを再生成するのが一般的な手法であるが，FileMaker の Web 機能はパフォーマンス的に弱く，それまでの開発で，すでに 1 ページの展開にかなりの時間がかかるようになっていたため，そのままサーバー側で処理を追加するのはさらなるパフォーマンス低下とアクションごとにユーザーの待ち時間が長くなるというデメリットが出ると予想された．そこで，アクションに対応した必要なレコードを Ajax で取得し，ページ上に追加する仕組みを構築した．つまり，クライアントサイドで DOM ベースでのテンプレート処理の原型をこの案件で構築した．

この経験から，すべてをサーバーで稼働させる必要がないことに気づき，DOM ベースでのテンプレート処理をクライアントサイドで行うことを考えた．しかしながら，他のフレームワークでは，レコードの繰り返しに対して何かしら特別な処理を行う．例えば，CodeIgniter は，ビューも PHP 言語で記述する．したがって，複数のレコードは配列でコントローラーより渡され，ビューの中で foreach 文等を使って繰り返しのプログラムを記述し，ループの中でタグの間に echo でフィールドを出力するといったことを行う．しか

しながら，エンクロージャーとリピーターの関係を HTML ページから導出することで繰り返しを HTML に特別な記述を一切加えることなくできることを確認し，フレームワーク化を本格的に開始した．

5.13.2　HTML に直接記述する Web アプリケーション

HTML に少しの記述を加えたり，独自の拡張を施すことで Web アプリケーションを作成する製品（「タグ言語」などと呼ばれる）は，すでに 90 年代に登場している．当時から存在していて現在残っているのは Cold Fusion[108] くらいである．当時よく使われたのは，FileMaker で利用できた CDML（Claris Dynamic Markup Language）である．FileMaker で作ったデータベースを公開できるもので，FileMaker 自身に搭載されていた．属性としてフィールド名を指定するのに加えて，一部に独自に拡張した記述を行うものの，HTML をベースにしながらも FileMaker のデータベースを取り出して手軽に表示できる点は評価されていた．しかしながら，データベースから取り出した内容を書き戻すようにするには，あらかじめ書き戻すための CGI を呼び出す URL を生成するようなフォームの中に，読み出したデータを埋め込むような作りにしなければならないため，簡単なものはすぐにできたとしても，複雑なものを作成するにはパズルのような作業が必要になってしまう．FileMaker 側での計算フィールドなどを利用することである程度のロジックは組めるとは言え，要求が複雑になると開発は困難になった．また,「ドキュメント化しない」ことが一般的な FileMaker デベロッパーが試行錯誤的に開発した複雑怪奇なサイトもあり，保守のしようもなく手付かずのまま極めて古い FileMaker を運用しながらいまだに使われている形跡もある．実際に筆者もそうしたサイトを 2013 年に INTER-Mediator で作り直した．

当時のままのタグ言語で現在も残って使われている Cold Fusion は，タグ言語である仕様を残しつつ，Java ベースでのロジック部分の開発機能を統合し，さらにほとんど手続き的プログラミング環境と言えるタグ言語の拡張を行うことで，複雑な要求に耐えうるフレームワークとして残っている．FileMaker は 2004 年にリリースされた FileMaker Ver.7 で CDML の機能を排除し，その後は，XSLT，PHP を使った Web インタフェースと，データベースをそのまま Web で公開できる IWP（Instant Web Publishing）や WebDirect という機能を搭載した．結果的に数々あったタグ言語の製品はほとんどが現在は使われていない理由は，簡単なページが簡単にできたとしても，複雑な要求に対処できなかったということがある．加えて，多くの人に「タグ言語はだめ」という心理的な意味でのマイナス評価を残す結果にもなった．

INTER-Mediator の開発を始めた時，タグ言語の不利な点をどのように克服するのかを考えた．テンプレートの記述のために HTML の拡張を考えたが，DOM の処理を効果的に利用するためには純粋な HTML である方が望ましい．その上で，クライアントサイドやサーバーサイドでの手続き的なプログラムによる拡張ができるようにする必要性はあると考えた．そこで，主要な処理を JavaScript をベースに記述することも考えたが，データベース側の更新を含むバインディングや，繰り返し処理をアプリケーション側で手続き的なプログラミングをせずに実現できたことから,「可能な限り宣言的な記述で動作させる」ことで，既存のフレームワークとの差別化や，FileMaker などの開発ツールとの差別化ができる点に注目した．

5.13.3 開発ツールとしてのスペック

開発素材は利用する開発者に評価されなければ意味はない．INTER-Mediator がオープンソースであり，さらに MIT License にした理由の 1 つはその点にある．フレームワークを広く世間に問う上で，流行しているフレームワークにはない形式のものを作るというのは，現実的には上手い方法ではない．例えば，MVC フレームワークが常識化している上では，他とちょっと違う MVC フレームワークの方が受け入れられやすいのは当然である．しかしながら，こうした風潮のためか，PHP の Web アプリケーションフレームワークは結果的に大同小異なたくさんの製品が出回る状態になっている．

一方，手続き的なプログラミングを主体にしないような開発ツール自体は広く使われている．開発コミュニティの中ではそうしたツールを「素人が使うたいしたことがないもの」と一蹴する空気もあるが，顧客やユーザーのニーズを満たすために合理的であれば甲乙つけるものではないと考える．こうした開発ツールはかなり以前には「4GL[96]」（第 4 世代言語）と呼ばれた経緯がある．1〜3 世代は，機械語，アセンブラ，高級言語を示し，その後に出てきたものという意味での 4GL であるが，その特徴の 1 つとしてグラフィカルな開発ツールという項目が，その後のアプリケーション形式での開発ツールに結びつくものである．また，非手続き的なコマンド言語を用いている点が特徴としてあげられ，その結果学習が容易であるといったメリットをもたらすとされている．

筆者のキャリアの中で，手続き的なプログラミングを業務として行いながら，FileMaker の開発にも関わった．その中で，あまり広くは認知されていないが，FileMaker での開発需要が決して小さなものではないことを知った．具体的な市場規模まではわからないが，米国で毎年行われる開発者向けのカンファレンスでは，参加費がかかるものの世界中より 1000〜2000 人規模の参加者がここ数年は集まる規模である．簡易ツールであるという点だけで評価に値しないと考える人もいて，さらによく「FileMakerって本当に使われているのですか」という質問を受けるほどであり，加えてメディアでも滅多に紹介されないほどあまり認知されていない FileMaker であるが，そこでの開発はインハウスで業務として開発している人や受託開発をしている人を中心としてビジネスとして成り立っている．

なぜ，FileMaker に市場があるのかという点の 1 つの答えは，エンドユーザー開発が可能なツールである点だ．実際，インハウスで開発をしている人が多くいる．だが，内製する時間がない，あるいは自分たちで作れる範囲を超えたという判断があれば，外部の開発会社に依頼することになる．FileMaker 社は，FileMaker Business Alliance（FBA）として，開発会社を組織化する一方，開発会社のリストを公開して発注側の利便を図っている．FileMaker 社自身は製品に関するマーケティング活動で「誰もが簡単にデータベースを作れる」というイメージを広めてはいるものの，現実にはデータベースアプリケーションを「誰も」が「簡単に」とは行かない．ただし，手がけたユーザーががっかりするには至らず，自分ではできないので開発者に頼もうという決定するユーザーが一定数はいるということである．この点は「ユーザーをだましている」とも取れなくはないが，FileMaker 社を中心とした FileMaker コミュニティとしては，ビジネスを生むエコシステムが作り上げられている．

業務システム開発経験の中では，ユーザーサイドで保守をしたいという意向はよく聞く．FileMaker であればそれが可能かもしれないが，手続き的なプログラミングを駆使して作った Web アプリケーションでは相対的な意味では無理と言っていいだろう．ユーザーは保守を自分で行えると考えて，開発を専門会社に発

注することもある．加えて Web での運用の希望もよく寄せられる．残念ながら，FileMaker の Web ソリューションは完全ではない．それならば，FileMaker のいいところを取り込んだ Web アプリケーションをフレームワークを作てしまおうと考えたことも，INTER-Mediator の開発を始めたきっかけである．FileMaker で複雑な処理はスクリプトで記述するが，スクリプトを組まなくても一定の範囲が使える．この落としどころがエンドユーザー向けかつ高度なシステム開発を可能にしていると分析した．

5.14　関連製品および関連研究

Web アプリケーション開発のための製品については第 3 章でも言及した．ここでは，JavaScript ベースの Web 開発の素材について比較する．これらのツールは「フロントエンドツール」と呼ばれることもある．なぜ，フロントエンドツールが登場したのかという背景に加えて，代表的なフロントエンドツールおよびそれらと INTER-Mediator の比較についても言及する．

5.14.1　JavaScript ベースの開発が注目される理由

Web の黎明期より CGI といった仕組みが整備されたこともあって，Web アプリケーションの実行は，サーバーサイドで行われてきた．その後，Java アプレット [76] や Flash[2] といったクライアントサイドで高機能なソフトウェアを実行する環境も 90 年代より存在し，特に Flash は 2000 年以降コンテンツ制作の重要な手段になってきた．ブラウザ上で稼働するスクリプトである JavaScript は，1995 年に Netscape Navigator 2.0[102] で実装され，各社のブラウザでも実装された．その後，90 年代末ごろからスクリプト内で通信機能が組み込まれて Ajax[39] として知られるようになりユーザーインタフェースの向上に寄与した [80]．ページ内の要素を木構造のオブジェクト群として扱う DOM[101] など，より高度な機能が利用できるようになり，JavaScript は Web 2.0[25][77] を実現する技術の 1 つとして認識されるようになった．その後に登場した HTML5[100] により，技術的な仕組みは多くがその中に包含された．

JavaScript の規格化団体として Ecma International[28] が 1997 年に登場し，ECMAScript[29] として標準化されている．初期の頃は補助的なスクリプト機能であったが，ECMAScript 5[30] はシステム開発に必要な十分な機能を持つに至り，最終的な仕様が公開される 2009 年に近づく頃，すべてのブラウザが新しい規格に追随するとともに，JavaScript のパフォーマンスを高めたこともあって，Web ブラウザ上で JavaScript を中心とした手法で開発される「狭義の Web アプリケーション」への注目が集まった．一般に複数のページを行き来する Web ページとは異なり，単一のページでアプリケーションを作成できるというコンセプト [38] が，提唱されたのは 2003 年からやや時間を経て現実的な手法となった．

同時期に HTML5 の制定が進んだこと，スマートフォンというネットワーク利用を前提にしたデバイスが普及したこと，さらには Apple の iPhone 等に搭載されている iOS での Flash 非サポートや，Apple の CEO だったスティーブジョブズ氏による「Thoughts on Flash[58]」により，クライアントサイドでの主要な開発言語（環境）が Flash から JavaScript へと移行した．

スクリプトの面でも独自性が強かった Internet Explorer も，2009 年リリースの Internet Explorer Ver.8 より HTML5 や CSS3[99] などの Web 標準規格への対応を強化するという戦略の変更を行い [103]，ブラウザ

上で稼働するアプリケーションの互換性が高まることが期待された．加えて，ブラウザのシェアも以前は Windows にプリインストールされている Internet Explorer[70] が圧倒的だったものの，オープンソースとして発展した Firefox[71] や，Google の Chrome[41] がそれぞれ大きなシェアを持ち，これらがいずれも Web 標準規格をサポートしていたこともある．結果的にブラウザ間の非互換性についてはノウハウの蓄積も含めて次第に緩和されてきた．このようにブラウザ間での非互換が緩和されたことが，狭義の Web アプリケーションへの注目が集まった理由である．

クライアントサイドでのソフトウェア実行環境が整備されたことで，Web アプリケーションの稼働環境として，JavaScript をベースにした手法に注目が集まるようになった．その前より Flash とサーバーサイドの動きを連動させるような仕組みで，航空機チケットの予約システムのような Web アプリケーションは開発されてはいた [3][128]．一方，現在の Web アプリケーションは JavaScript などの Web 標準規格を拠り所にした世界なので，ページの中身全体に対するアクションを構築できる．さらに，HTML は Web コンテンツの基本であり，多くの開発者やデザイナに馴染みもある．その HTML をベースにした実行環境は，HTML そのものが新しいものではないとは言え，JavaScript での開発に現実味が出てきた 2000 年代後半より新たな形式の Web アプリケーションフレームワークの登場を後押しすることになった．

5.14.2 Web ページを DOM として扱うフレームワーク

サーバーサイドの Web アプリケーションの多くは，HTML ベースのテンプレートを使用しているが，それをテキストとして扱った上で，データベースから得られたデータをマージするなど加工して，クライアントへ送付していた．しかしながら，本章で解説するように，テンプレート処理をクライアントサイドでも実現できるようになり，さまざまなフレームワークが開発されている．ここでは，バインディングの仕組みをクライアントサイドで実現するフレームワークについて説明する．

早くに登場したのが現在の「AngularJS[11]」である．2009 年に<angular/>として Brat Tech LLC より登場した [53] が，その後に Google に開発チームが設けられ，コミュニティを形成し，AngularJS という名称になって開発が続けられている．HTML で記述した要素の属性に記述を加えることで，バインドつまりモデル内の特定のデータと要素に表示される内容が連動する仕組みがあり，すなわち宣言的な記述でモデルとページ要素との連動ができる．さらに，Directive という仕組みで，タグや特定の属性などの HTML の要素が，JavaScript のプログラムの中でオブジェクトとして参照できる仕組みもある．この仕組みを利用すると，HTML で記述したものに対して JavaScript で独自に機能を組み込むことができる．こうした機能を持つことから，AngularJS は「フレームワークを作るためのツールキット」という位置付けを現在は主張している．以前は MVC フレームワークと表現していたが，方針を転換した．コミッターは 2014 年 10 月現在，1,004 人に及んでいる．

2010 年には，Microsoft より，MVVM パターン [43] を基調としたバインディングやテンプレートの仕組みを持つクライアントサイドで稼働するフレームワークの「KnockoutJS[89]」がリリースされている．単独での利用も可能ではあるが，Microsoft の Visual Studio 上での開発において，サーバーサイドとクライアントサイドを 1 つのプロジェクトとして開発する場合の重要なコンポーネント言った位置付けになっている．商用品でも利用されているが，開発はオープンソースコミュニティーで行われており，レポジトリでは 2014

年 10 月時点で 43 人のコミッターがいた．

2011 年にリリースされた「Ember.js[60]」も，宣言的な記述でバインドを実現可能なフレームワークである．JavaScript ライブラリとして利用されていた「SproutCore」の Ver.2 で MVC パターンでのアプリケーション開発機能を組み込む機会に，SproutCore より分離して Ember.js となった．Ember.js のテンプレート記述は純粋な HTML ではなく，一部に独特な記述を用いている．オープンソースであり，2014 年 10 月時点で 407 人のコミッターがいた．

5.14.3　INTER-Mediator と他の JavaScript フレームワークとの比較

INTER-Mediator は宣言的な記述でバインディングを実現するが，同様に AngularJS，KnockoutJS，Ember.js も実現している．しかしながら，INTER-Mediator は Web アプリケーションの構築全体を宣言的な記述を主体にして行えることを主眼としているのに対して，他のソフトウェアは宣言的な記述だけでなく JavaScript でさまざまな機能を構築することを目的としている．基本構造に共通点があるが，エンドユーザーでも保守できる INTER-Mediator に対して，他のソフトウェアは手続き的なプログラミングにより系統的なコード作成が可能な開発者が利用するフレームワークである．

バインディングとテンプレートの仕組みを比較すると，INTER-Mediator はエンクロージャー/リピーターの識別を通じて，HTML で記述した構造から複数のレコードを展開するために複製する範囲を自動的に判別している．そのため，単一のデータソースが複数のレコードを持つ場合，レコードの数だけリピーターのノード群を複製してバインドすることで，リスト形式にレコードを並べることなどができる．加えて，リピーターに内包するエンクロージャーを識別することで，別のデータソースをさらに展開し，場合によってはリレーションシップに基づくレコードセットを得てページ上に必要な要素を追加してバインドすることができる．AngularJS[45]，KnockoutJS[7] 共に，階層的に異なるデータソースを得て展開することはできず，別ページとして定義したものを挿入するという記述を行う．従って，INTER-Mediator の方が簡潔に記述ができる．業務系のアプリケーションでは複数のテーブルからのデータを統合してページに表示することは一般的に発生する要求であり，バインディングやテンプレートにおいて，複数のデータソースを最初から考慮して INTER-Mediator は構築した．

INTER-Mediator には新規レコード専用の動作モードがあり，手続き的なプログラムを記述しなくても，入力値の検査と新規レコードを作成する機能を利用することができる．また，定義ファイルへのキーワードの追加で，レコードに対する「削除」ボタンの付与や，一連のコンテキストに対する「挿入」ボタンの付与により，やはり手続き的なプログラミングをしなくても，レコードの新規作成や削除ができる．例えば，AngularJS で入力フォームを作成する例 [132] では，コントローラーやモデル部分を JavaScript のプログラムで記述する必要がある．Knockout でのフィールド編集やレコードの追加や削除を行うような例 [93] でも同様である．手続き的プログラミングの柔軟性は評価できるものの,「削除」ボタンなどの典型的な仕組みを宣言的な記述だけで実現する INTER-Mediator の方が，時間や予算が限られているエンドユーザーに向く手法であると言える．

5.15 フレームワークの動作に関するまとめ

エンドユーザー開発に利用できるために，手続き的なプログラミングが必要な箇所を少なくする工夫を行なった．特に，テンプレート処理において，1 レコードをバインドする範囲をエンクロージャー/リピーターのセットを自動的に判別することで複数レコードの展開を一切の記述がなくてもできるようにした実装は，他の関連ソフトウェアには見られない仕組みである．

第6章　学習可能性に対する評価実験

第 4 章で示したように，作成されたページの改変を行うような保守作業では，宣言的な記述で行える範囲は，一般的な手続き的言語での開発作業を主体としたフレームワークよりも広範囲に渡る．その結果，エンドユーザーが保守作業に関われる可能性は指摘できるもの，本フレームワークを使用した保守作業をエンドユーザーが実際にできるかどうかについての検証が必要である．そこで，被験者に対する評価実験を通じて，INTER-Mediator が学習可能であるかを検証した．評価実験では，学習を行った後に試験を行った．学習や試験の手法と被験者について説明する．

6.1　評価実験の目的と評価方法

INTER-Mediator を使って開発されたシステムに対する保守作業ができるには，前提として HTML やデータベースの知識が必要である．その上で，フレームワークに特有な記述を適切に行える知識が獲得できるかどうかが問題である．そこで，フレームワーク特有の知識が対象とするエンドユーザーにとって学習可能であるかどうかを，被験者を対象にした実験によって検証した．もし，学習が可能であるとすると，システム改変をエンドユーザーの手で行うことがより現実的になり，業務システムユーザーであれば，ビジネス環境に応じたシステム変更を利用者によって行えることにもつながり，即時対応やコスト削減につながる．

また，学習ができるとしたら，どのくらいの時間がかかるのか，学習しづらい話題があるのかどうかという点も，開発フレームワークの今後の改良や，開発を希望する人たちへの普及活動に対しての方向性を示すことができる．被験者の手続き的なプログラミングの経験に関係するのかどうかや，フレームワークでの開発を理解するための前提知識は何になるのかということも疑問として生じる．これらの疑問に答えることができれば，利用を考えている人たちに対してどの程度の学習が必要なのかを判断する材料を与えることができる．

被験者はまず，フレームワークの利用法を学習した．その後に試験に臨み，開発時に発生するような課題に対する記述を被験者が実際に行った．試験の正答率を学習可能性の基準とした．

6.1.1　評価方法について

本フレームワークを使用した開発では，HTML による記述や，定義ファイルの記述を行う必要がある．開発や保守の作業ができるためには，要求や問題を特定した上で，記述を行ったり，あるいは記述を変更することができる必要がある．そのためにはフレームワーク固有の事情に沿ったフレームワークの利用方法の学習が必要である．

学習を行った被験者が記述能力を獲得できているかを確認するために試験を行い，その正答率をもとに

評価を行った．学習可能性についてはさまざまなメトリックが提唱されており [44]，本論文での手法はその中にある「コマンドの使用に基づくメトリック」に分類される「学習後にコマンドを正しく利用した割合」に相当する．この手法は作業を行う上で機能の利用を正しく行った割合を基に学習方法を比較した論文で使用された．本論文では機能の利用ではなく記述を正しく行った割合を基準にする．

学習を行った効果を測定するためには，学習前後での成績の向上を測定するのが 1 つの方法である．しかしながら，本フレームワークに対する学習内容は，独自の概念や記述方法があり，一般的な知識からの類推は困難である．本フレームワークは知名度が低いので，被験者が事前にフレームワークに関する情報を得る機会はほとんどない．仮に学習前に本フレームワークに関する問題を解答しようとしても，理解できない用語が並び，知識を適用できる状態とは言えない．そのような状況での試験結果には意味がないと考え，学習前は一様に知識がない状態であると仮定した．

6.1.2　被験者について

被験者は 12 名で，Web 業界で働くデザイナーやコーダーが中心であり，講師，ディレクターといった職種の被験者がそれぞれ 1 名ずつ含まれていた．すべての被験者は HTML については知識が十分にある．したがって，HTML の記述力があるという前提をもとに，フレームワークの知識を学習して開発に活かせるかどうかを評価できる．

評価実験の被験者は，手続き的なプログラミングを主業務としないような Web 業界のデザイナーやコーダである．エンドユーザーによる開発は，そのユーザーが持つ自身のドメイン知識を開発に適用することによる質的な向上を期待できる点がある．しかしながら，エンドユーザーが持つ知識は千差万別である．INTER-Mediator による Web アプリケーション開発において，ユーザーが持つドメイン知識が活用できることは期待できるが，それを実現するための基礎的な知識は必要である．つまり，HTML や CSS といったページ作成の知識，データベースやサーバーに対する知識が前提としてあって，INTER-Mediator 特有の概念を理解した上で Web アプリケーションが稼働するための記述ができるようになる．エンドユーザーを対処に実験をする場合だと，被験者ごとの知識のばらつきを考慮しなければならない．一方，Web 業界のデザイナを被験者とすることで，前提となる HTML の記述の知識を持っていることが仮定できる．そこで，HTML の記述を理解している集団として，Web デザイナを中心とした Web 業界の人たちを被験者とした．

Web デザイナは，Web アプリケーション開発の上では，ソフトウェアエンジニアとエンドユーザーの中間的な立場でもある．また，Web 開発ではエンジアだけではなくデザイナも加わることが一般的である．Web デザイナが INTER-Mediator を学習可能であれば，Web アプリケーションの開発や保守に，Web デザイナが関わる場面を増やすことができる．Web デザイナは直接的な意味でのエンドユーザーではないものの，エンジニアが実施していた範囲の作業ができることにより，Web アプリケーション開発者が増えることが期待できる．

プログラミング経験に関して被験者にアンケートを取ったところ，8 人が「時々プログラミングを行う」を選択し，残りはプログラミングは一切行っていないと回答した．プログラミング経験としては，デザインの一環として JavaScript のプログラムを追加したり，CMS でプラグインを稼働させるための小規模な修正が中心であった．被験者はプログラミングを主業務にはしていない．

6.1.3 学習コンテンツと試験問題について

宣言的な記述による Web ページ開発手法を学習するためのコンテンツを用意した．ページの一例を図 6.1 に示した．被験者は各自で内容を読むことで学習を進めた．全 10 ページで文字数は約 25000 文字あり，このうち INTER-Mediator に特有の内容は 7 ページで，残りは HTML やデータベースに対する基本的な概念の説明をしている．各ページのタイトルは以下の箇条書きの通りである．

1. Web ページ制作についての基本
2. データベースについての基本
3. Web アプリケーションの概略
4. INTER-Mediator でのバインディングの概念
5. 繰り返しの実装
6. コンテキストでの検索とソート
7. マスター参照の組み込み
8. テキスト以外の UI 要素の利用
9. INTER-Mediator のその他の機能

最初の 3 ページでは，HTML の基本に加えて，データベースを表として捉えることと，Web アプリケーションについての動作の基本についても解説している．被験者は HTML の基礎は身についているが，学習コンテンツの最初に明らかに知識を持っている内容を記載することで，学習の導入をしやすくすること意図した．4 ページ目以降は，INTER-Mediator 特有の概念を例を示しつつ解説した．INTER-Mediator の全機能を説明はせず，宣言的な記述で行える機能のうち，主要な機能に絞って学習コンテンツを制作した．

被験者には自由な時間に自分のペースで学習を進めた．学習時間はページを開いた時間および読み終わった時のボタン操作の時間から測定した．同程度の学習にかかる時間には個人差があることから [14]，時間制限はせず被験者は自己判断で学習時間を確保した．

被験者にはあらかじめ試験を行う事を伝達しており，学習した上で試験が可能と被験者が判断した段階で，試験に臨んだ．試験は Web ページで 8 ページ分を用意し，学習コンテンツと同様な方法で自己申告での作業時間を測定した．このうち，INTER-Mediator 特有の事情を問うものは後半の 5 ページ分で，解答欄は 45 個あった．試験の Web ページの一例を図 6.2 に示す．試験時間も自由とした．また，学習コンテンツを参照することは妨げなかった．通常の開発作業では，ドキュメントを見ながら行うのが一般的であり，これと同等なものとして資料や閲覧は必要に応じて行って良いとした．

試験の各ページは，学習コンテンツのページと対応付けているが，学習コンテンツの 9 ページ目の内容はそれ以前のページに含めており，10 ページ目は試験には含めていない．

試験問題のうち，フレームワークに関わる回答部分はすべて記述式とした．選択問題は何の知識がなくても一定割合の正答を得る確率があるので，試験問題から排除した．また，問題の例や解答として記述すべきことは，学習コンテンツの例や説明との類似をなるべく避けて，学習コンテンツの記述からの類推をしづらいものとした．答案作成はより困難になるが，知識の習得と問題解決の能力が必要になり，勘で当たるようなことがないことを意図している．

図 6.1: 学習コンテンツの一例

試験問題の解答作成を実際に試してみることができるように検証環境を用意して，試行錯誤する環境を整えた．Web デザインでも記述した結果をつどつどブラウザで確認するようなことは一般的であり，通常の作業フローに近くなることを意図している．試験後に，プロフィールや感想などのアンケートを行った．

6.2　実験結果

12 人の被験者についての実験結果について，試験の得点と学習および試験に費やした時間に関して検討する．そして，問題の分野ごとの正答率や，アンケートから得られた結果についても検討する．

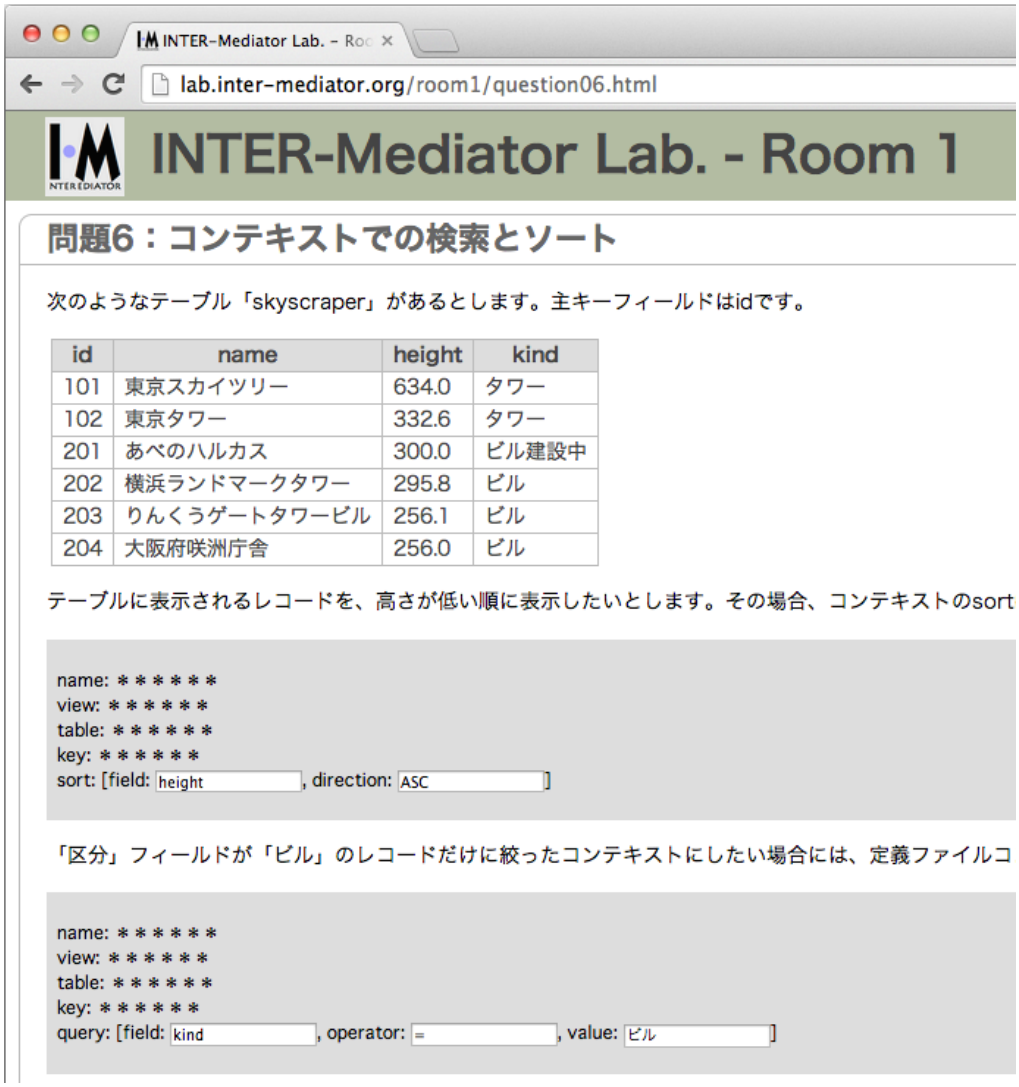

図 6.2: 試験問題の一例

6.2.1 学習状況と試験結果

図 6.3 には，被験者ごとの試験の得点と各被験者のプログラミング経験をグラフに示した．試験問題の結果は 0〜93%と広い範囲に分布した．図 6.4 には，被験者ごとの学習および試験に費やした時間を示した．得点が 13%以下の 4 人の被験者（グループ A：S9, S6, S4, S8）は，試験時間が 15 分以下となっている．一方，それ以外の被験者（グループ B）は得点が 42%以上であり，試験時間も 45 分以上である．試験に費やした時間が短く得点も低いグループ A と，試験の答案のために十分に時間を使用し一定以上の得点を得ているグループ B の，2 つの大きなグループに分類できる．得点および試験時間に対してグループ A と B の平均値の差に関する t 検定を行ったところ，いずれも有意水準 1%で有意差があるという結果が得られた．

グループ A は学習ができず，試験を早々にあきらめたことが推測できる．一方，グループ B の 8 人は，1 人を除き 50%以上の得点を得ており，満点に近い被験者もいる．しかも，学習や試験の平均時間は 62 分および 72 分で，十分な時間を利用している．もし，回答に時間を費やしたにもかかわらず試験結果が極端に低い得点であれば，学習が困難であることを示している．あるいは，極端に高い得点に固まっているのであれ

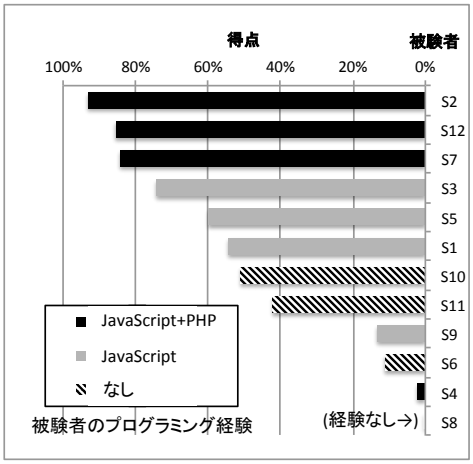

図 6.3: 試験結果とプログラミング経験

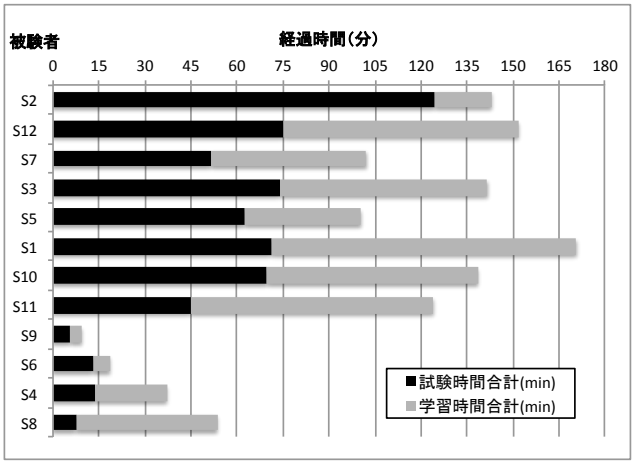

図 6.4: 学習および試験に要した時間合計

ば，学習コンテンツや試験が不適切であることなどが考えられる．グループ B では比較的高い目の得点で分布しており，学習の結果が表れていることが理由の 1 つとして挙げられるとすれば，この結果より学習が可能であると言える．試験結果より，十分な時間をかけていないグループ A と，時間をかけたグループ B の間には明らかな得点差がある．学習前には知識がない状態であると仮定すれば，グループ B の被験者には学習効果が現れていることを示す．ただし，理解の度合いには個人差がある．

6.2.2　プログラミング経験と得点の関係

試験後のアンケートに，手続き的なプログラミングをどの程度行っているかを設問として含めた．JavaScript とその他の言語について,「ほとんど毎日行う」「よくやる」「時々行う」「たまに行う」「滅多にやらない」の選択肢を用意し，その他の言語については言語も回答するようにした．JavaScript については,「時々行う」と「滅多にやらない」のみ選択があり,「時々」行うと答えた被験者を図 6.3 に記載した．その他の言語については，PHP のみが回答され,「たまに行う」と「滅多にやらない」の回答のみがあり,「たまに行う」行うと答えた被験者を図 6.3 に記載した．

言語でのプログラムに関して，アンケートや一部の被験者に直接問い合わせた．PHP を使うのは WordPress などの CMS の改造を行うことがほとんどであった．WordPress はプラグインを PHP 言語で開発して追加できるが，その設定を言語で書かれたソース内の記述で行え，そうした場面で PHP を使うことが一般的であった．JavaScript については，デザイン上必要な短いプログラムを主として検索結果を参考に付与する程度の作業が中心であった．いずれも，自分でクラスを定義したフレームワークをもとにロジックを組み立てるような作業まではしておらず，プログラミングの専門家が行うような作業はしていなかった．

図 6.3 からプログラミング経験者と得点の関係を見る限り，特に PHP 言語を利用する機会がある被験者が上位にあることや，上位 6 人がプログラミングを時々行うことを考えれば，関係はあると言える．一方，試験をあきらめたグループ A の半数もプログラミング経験はある．この結果から，プログラミング経験があれば学習しやすい傾向にあるとは言えるものの，確実に学習できるということは言えない．

しかしながら，グループ B については，上位の被験者は JavaScript と PHP の両方を行い，続いて JavaScript

だけの被験者となり，プログラミングを滅多に行なわない，あるいは行っていない被験者は得点が低い傾向にある．プログラミング言語の領域まで，知識として身につけている被験者ほど高い得点を得ることができるのは，高い学習意欲および学習能力があることが関係していると考えられる．

6.2.3 学習時間に関する評価

学習時間と試験時間について，図 6.4 に示した結果から検討する．学習に時間をかけたグループ B については，ほとんどの被験者は学習と試験が同程度の長さである．得点が最も高い被験者 S2 のみ，80%を超える時間が試験に費やしており，時間配分に偏りがある．試験の実施中でも学習コンテンツを参照できることから，学習を早めに済ませて疑問点は試験を受けながら学習を補強して行った可能性がある．被験者 S2 はその傾向が強く，S5 についても S2 ほどではないものの，試験時間を長く取る傾向にが見られた．学習は一定の知識を得るプロセスであり，試験は得た知識を適用する作業を経験するプロセスでもある．それらを合計した時間が知識の獲得にかかった時間であると考えれば，偏りは被験者ごとの進め方の違いの現れであると言える．学習と試験の合計時間で，学習時間を評価するものとする．

グループ B の被験者については，合計の平均値は 2 時間程度であった．範囲も 100〜170 分となり，極端に短い被験者はもともと排除しているものの，極端に長い被験者もいないことから，学習と試験が簡単に済ませられないような十分な負荷を与えるものであった．また，一定時間内で完了できる性質のものである．2 時間の長短を比較するものがないので論じることはできないが，基本概念と宣言的な記述の学習時間の目安としては，HTML の知識があれば 2 時間程度である点がこの結果から得られる．現場のエンドユーザーが業務の片手間に進めることで，1，2ヶ月程度をかければ HTML の知識を始めとして INTER-Mediator での開発手法までの知識は得られると言える．

検証環境は用意はしていたが，被験者の利用頻度はグループ B で全部で 5 問あるのに対して平均して 2.3 問であり，検証せずに回答とした場合が多かった．また，得点と検証環境の利用度合いには関係性が見られなかった．試験後のアンケートで検証環境の使用方法が分からなかった点を指摘され，積極的に利用できる状態ではなかったのが低い利用率の原因である．検証環境の利用方法に対する適切なガイダンスができなかった点は反省点である．

6.2.4 問題分野ごとの正答率

一定以上の得点を得た 8 人に対して，45 問中，43 問について，フレームワークのどのような仕組みに対する問題なのかを表 6.1 のように種類分けした．各問題は最初の 3 種類のいずれか 1 つに属する．最後の 2 種類に含まれる各問題は最初の 3 種類にも属している．ここで「保守作業」は，ページファイルでのターゲット指定に関する問題と，コンテキストの中での検索条件と並べ替えに対する問題を集めた．コンテキストが作られ，一定の範囲でページが構築された上で，比較的小規模な改変をする場合に必要な知識として，ターゲットの指定と検索条件および並べ替えの指定の記述方法を「保守作業」と位置付けた．

種類ごとの正解率を図 6.5 に示した．例えば,「ターゲット指定」に関しては，75%以上の正解率が 2 人，50〜75%が 4 人，25〜50%が 0 人，0〜25%が 2 人であったことを示している．

表 6.1: 問題の種類と概要

分野	問題数	設問の概要
ターゲット指定	14	フィールドや属性などの指定
コンテキスト	19	データベース利用や検索条件など
繰り返し	10	複数のレコードの表示
リレーション	15	テーブル間の関連性
保守作業	20	保守時に利用されることが想定される記述

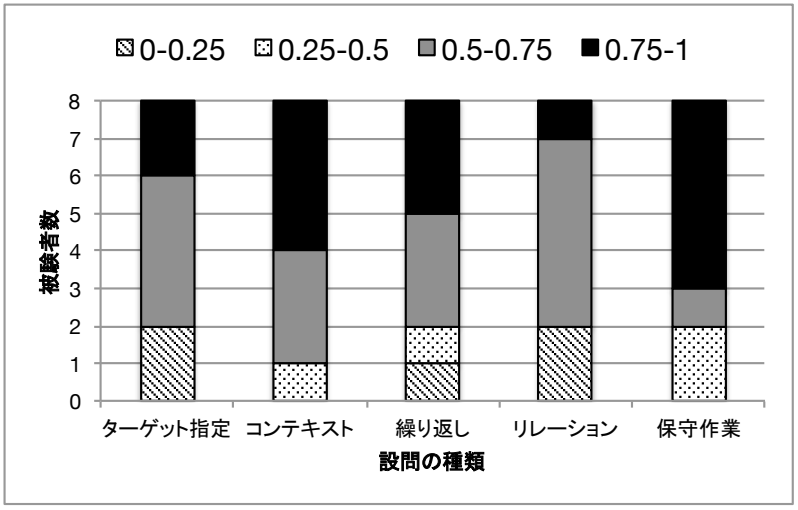

図 6.5: 問題の種類ごとの得点分布

すべての分野に対して，8 人中 6 人以上が半分以上正答を得ている．このことから，どの分野も同程度に学習ができたことを示し，分野に対する強い偏りは見られなかった．75%以上の回答の傾向を見る限りは，保守作業が最も人数が多く，習得しやすい傾向があるが大きな差はない．一方，リレーションシップについての問題は 75%以上の正答者が最も少なく，問題分野としてはどちらかと言えば困難な部類にあると言える．

分野ごとの違いに関して，リレーションシップについては概念上複雑なものだけに，正解しにくい傾向にあることは，実験の結果を引用しなくても言える点ではある．実験結果からは，むしろ問題分野ごとに大きな差がない点が重要である．実験結果を見る限りは，ある分野でほとんどが 0 点であり，別の分野でほぼ全員が正解しているといった状況ではない．いずれの分野でも得点の多くが 50%以上となっていて，学習に得た知識で一定以上の正解が得られるている点が見られる．ここで分離した分野は，INTER-Mediator 特有の概念の中でも必ず理解する必要があると考えられる重要なものである．誰も理解できなかったような分野がなかった．多少のばらつきはあるものの，基本概念として扱ったものはいずれも学習可能であったと言える．

6.2.5 被験者に対する事後のアンケート結果

自由記述形式によるアンケートを，試験を受けた後に被験者に対して行った．フレームワークの機能のうち，わかりやすい点や評価できる点については，6 人の被験者が，HTML のタグ内にフィールド名などの記述を加えるだけの簡単な作業でデータベースと連動できる点を挙げられた．被験者のプログラミング言語の利用については，6.2.2 項で解説をしている．以下，難しいと感じた点の回答結果をもとに，データベースに関する学習の必要性を議論する．

学習ができていないとみられるグループでは，難しいと感じたことに関して「全く知識の無い当方にとっては敷居が高く感じました」「データベースへの知識はないので，自身のスタート位置が低かったです」といった回答があった．また，中程度の得点の被験者からは,「日頃データベースを扱っていないので聞き慣れない言葉が多く理解するのがむずかしかった」「Microsoft Access を使ったことがあるので，理解できた部分もあります」という回答があった．一方，高得点の被験者からは，難しいと感じた内容としてはリレーションシップが挙げられた．

これらのアンケート回答結果より，学習ができなかった被験者の理由の 1 つとして，データベースそのものに対しての経験や知識がない，あるいは少ないことが挙げられる．学習コンテンツには，データベースの基本についての解説に 1 ページを割いているが，基本概念の説明だけであった．Web アプリケーション開発に必要なデータベースに関する知識を身につけるには概念の学習だけでは不十分であった可能性がある．実際にデータを入力するなどのデータベースの利用経験が必要であったと考えられる．

高得点層ではより深い知識が必要なリレーションシップの知識を困難な点と挙げているため，基本的なデータベースについての知識は背景にあったと考えられる．これらの回答より，本フレームワークを学習するにあたっての前提知識としては，HTML の記述だけでなく，データベースに関する知識も必要であることが言える．

今回の実験では，データベースの知識を確認あるいは揃えるということは行わなかったが，学習コンテンツでは基本概念の解説は行った．リレーションシップについても例を出した上で，解説をしている．しかしながら，日常の生活では，数字で振られた主キー値をもとに対応をとるようなことを実際に行うことは滅多になく，データベースを日常的に使っていない限りは理解の及ばない世界になってしまった可能性もある．もちろん，フィールドとのバインドのような，データベースを表形式に捉えて理解することで，求める機能を引き出せるように設定できるような内容もあり，データベース全てがわかりづらいということではない．リレーションシップのような回答を導くのに頭を捻る必要がある問題が，ここで議論してるアンケートの回答のような結果を出してしまう要因と考えられる．

被験者の HTML 記述に対する知識差はなくても，データベースに対する知識の差が，試験の得点に影響した可能性がある．学習においては，Web 技術だけでなくデータベースの知識も重視する必要があるが，特にリレーションシップのような分かりにくい概念を扱いやすくする，あるいは学習しやすい状況を作るということも必要である．

6.3　関連研究

学習可能性（Learnability，あるいは「学習のしやすさ」）は，ユーザビリティを実現するための 1 つの項目として挙げられている [134][27].

Nealsen は学習のしやすさを確保するためには，初心者ユーザーの学習曲線（時間に対する習熟度や効率性）が早期に高い割合に行くべきであると主張する．ビジネスマンを対象にしたユーザビリティに関する調査 [74] で高く評価された点の 1 つとして「全部を学ばなくてもプログラムで必要な作業ができる」という点があることを示している．INTER-Mediator に関しては，HTML の要素の属性にコンテキスト名やフィールド名を記述するだけでデータベースと連動するという点は，アンケートにより学習者にとってはわかりやすい点であった．そうした，全体を理解していなくても容易に理解できるコア機能がある点も，学習可能性の実現に寄与している．

Dix らはコンピュータシステムの Learnability に影響を及ぼす原則として，Predictability，Syntesizability，Familiarity，Genralizability，Consistency の 5 つを挙げている [27]．コンピュータシステム一般の知見ではあるが，Web アプリケーション開発のフレームワークの学習のしやすさについても同様な評価手法は当てはめることはできると考えられる．それぞれの原則が INTER-Mediator に当てはまるのかを表 6.2 にまとめた．

表 6.2: コンピュータシステムの Learnability に影響を及ぼす原則との対比

原則	意味	Web デザイナの被験者に対する効果
Predictability	操作などの道筋を立てやすい	どこに記述するのかは学習をしないと導き出せない
Syntesizability	操作などの結果を想定しやすい	HTML を記述して結果を得てきた経験がある
Familiarity	これまでの経験を活かしやすい	HTML を記述して結果を得てきた経験がある
Genralizability	知識を異なる状況にも適用しやすい	試験結果に大きくばらつきがあることから，困難な点もあると考えられる．
Consistency	同じような状況では似た結果になる	記述という点では統一されているが，動作がどうなるかは学習による習得が必要

本章の実験においては，Web デザイナを中心とした被験者なので，Familiarity つまりこれまでの HTML を記述して結果を得てきた経験があることがまず挙げられる．同様な理由で HTML のコードから結果を推測することを日常的に行っているため，Syntesizability もある状況であったと言える．Predictability，Genralizability，Consistency については，フレームワーク自体にそれらの性質があるかどうかに依存する．INTER-Mediator ではデータベースへのバインドを DIV タグでも INPUT タグでも利用できる点などがその根拠とは言える．しかしながら，リレーションシップの扱いのように理解が相対的に低くなる内容もあり，全般的に Predictability，Genralizability，Consistency があるかどうかという点では，学習の結果を的確に適用することが求められるため，試験の得点がばらついている結果を見る限りは必ずしも確保されているとは言えない．学習結果に良い影響を与えたのは Familiarity であり，普段扱っている HTML の中に属性として記述できる範囲での理解を助けたと考えられる．一方，リレーションシップのように日頃意識をしていなかったことは Familiarity が少なく，相対的に低い結果になったと言える．

Web アプリケーションに関するエンドユーザーのメンタルモデルに関する他の研究 [87] においても，リレーションシップや外部キーといった直接目にしないような仕組みを理解していないユーザーの方が多いと

いう結論が示されており，本論文での実験結果でも同様の傾向が見られた．

6.4 学習可能性に関する評価実験のまとめ

以上の実験結果より，被験者は 2 グループに分類できた．3 分の 1 の被験者は極めて低い得点で短時間で試験を終え，残りの被験者のほとんどは半分以上を正解し，学習や試験に対して時間をかけて取り組んでいた．後者のグループは学習の結果，問題への回答を得られたと言える．このことから，宣言的な記述を主体にした Web アプリケーションフレームワークの開発や保守のための知識は，学習をすることで，手続き的なプログラミングの経験が少ないユーザーでも，個人差はあるものの獲得が可能であることが学習後の試験結果より結論付けられた．また，問題を分野ごとに分類して正答率を見ても，どの分野も 8 人中 6 人が 50%以上の得点であり，いずれの分野も学習は可能であることが分かった．これらの結果より，Web デザイナが学習を進めることで，保守作業を担当することは，現実的であると言える．

学習および試験に費やした時間も 2 時間程度であり，開発環境を習得するコストとしては合理的な範囲と言える．本実験では，INTER-Mediator の宣言的な記述のうち主要な機能に絞ったが，機能のごく一部ではなく，フレームワーク特有の概念は含まれており，この範囲を学習すれば，その他の機能は，一般的な Web アプリケーションの機能との対比をすれば理解できる範囲である．多めに見積もって 5〜10 時間の学習をするとしても，1 日 1 時間であれば，1〜2 週間程度の期間で基本的な内容を学習でき，業務の片手間に学習をして比較的短期間で一定レベルまでマスターできると言える．

被験者のプログラミング経験は有利に働く場合もあるが，学習ができなかった被験者でもプログラミング経験者がいた．しかしながら，学習ができたグループ内では，JavaScript や PHP の経験がある被験者ほど得点が高く，プログラミング経験と学習成果が連動する傾向もあった．デザインだけでなくプログラミングといった周辺の知識まで持っている人は，もともと学習意欲が高かったり，能率よく学習できる素質があると考えられる．また，フレームワークの動作は，データの移動により表示や更新を実現するものである．そうしたコンピュータの処理を想起して問題の解決につなげるには，手続き的なプログラミングに特有の連続した処理の上にシステムが動作するという基本概念の理解が基礎になっているとも考えられる．プログラミング経験のある被験者であれば，未知のフレームワークの学習であっても，効率よく進めることができることも言える．

開発に必要な知識として，フレームワーク固有の知識以外に，HTML について知識が必要になる．加えて，データベースに対する知識についても Web アプリケーション開発では前提として必要とされる知識であることもアンケート調査から分かった．実験に参加した被験者は Web 構築を業務としているため，HTML についての知識は十分あると言えるが，データベースに対する知識や経験はまちまちであり，経験が少ないほど得点が低い傾向が見られた．記述のための知識があっても，データベースに関する知識が十分ではない場合は，正解を導くことはできないことを示している．学習コンテンツでは基本概念は掲載したが，概念だけでなく実際のデータに対する処理を検討できるようになるには，例えば手軽に使えるデスクトップアプリケーションで実際にデータを扱った経験などが必要であることが言える．

第7章　INTER-Mediatorの適用範囲と評価

本章において，INTER-Mediator を利用して開発する上で効率的に開発ができるアプリケーションと，手続き的なプログラミングが必要なアプリケーション，あるいは考慮が必要なアプリケーションがどのようなものかを説明する．さらに，実際に開発を行った方々へのインタビューと，現在の利用実績を紹介する．

7.1　INTER-Mediatorに向く開発と向かない開発

この節では，INTER-Mediator を利用して Web アプリケーションを開発する上で，どのようなアプリケーションであれば適合度が高いのかを，フレームワークの持つ機能，業務でのコンピュータ利用の観点から議論する．また，開発者やユーザーの特性からも議論する．

7.1.1　機能面からの考察

INTER-Mediator を使用した Web アプリケーションは，宣言的な記述の範囲内で開発した場合，基本的には 2 階層のアプリケーションとなる．一般にはアプリケーションのアーキテクチャとしては 3 階層以上の階層構造を求められているものの，2 階層のアプリケーションで実現できるものも多い．そのようなアプリケーションを，なるべく少ない記述で実現することを目指したのが INTER-Mediator である．

画面に表示される要素と，データベースのあるレコードのあるフィールドとのバインドが機能の基本になっているため，MVC フレームワークであるような「コントローラー」を意識して構築するタイプのフレームワークではない．コントローラーに相当する機能としては，計算プロパティにより，計算結果の表示をするくらいはできるが，それを超える処理を組み込むような用途では，手続き的なプログラミングが必要になる．INTER-Mediator で宣言的な範囲内で開発できる Web アプリケーションは，Excel のワークシートを共用するような用途，あるいはノートなどの紙ベースで行う台帳を使って管理するような用途である．例えば，社内や組織内の機材や資産管理，あるいは会議室の予約といった用途であれば，基本的な機能は宣言的な記述だけでまかなえる．使い勝手を高めるためには，手続き的なプログラミングが必要になる場合も考えられるが，まずはデータの蓄積を行うところから使い始められる．

INTER-Mediator は宣言的な記述を中心に開発できるようになっている一方，手続き的なプログラミングを行わないと機能の組み込みができない面もある．ユーザーインタフェースの開発については，典型的な処理は宣言的な記述での開発が可能であるが，それを離れた処理は手続き的なプログラミングが必要になる．例えば，テキストフィールドと，データベースのあるレコードのあるフィールドとのバインドは宣言的な記述ができ，ユーザーインタフェースを利用した更新は，データベースへ反映される．このとき，ユーザーインタフェース側での更新結果を別のデータベースのフィールドに対しても反映させたいような場合，つまり，

1 つのユーザーインタフェースオブジェクトから複数のフィールドに適用したいような場合には，手続き的なプログラミングが必要になる．具体的にはページ上のオブジェクトの更新イベントを受けて，JavaScript でデータベースの処理を行う．コンテキストに対する更新等の API は用意しているので，単純なものでは，数行程度の記述では済むが，手続き的なプログラミングに加えて INTER-Mediator 独自の API の学習も必要となり，対処可能な開発者は限定される．

また，複雑なビジネスロジックは，通常はサーバー側に階層的にオブジェクトを用意するなどして対処する．INTER-Mediator はサーバーサイドで，アスペクト指向的な拡張点は持つものの，ここでも手続き的なプログラミングが必要になると同時に，永続的なオブジェクトを独立して扱う機能等が用意されておらず，PHP 言語の基本的なオブジェクト指向の仕組みをベースに複雑なロジックを作成しなければならない．もちろん，この点も改良すべき点の 1 つではあるが，本フレームワークの目的から優先順位は低いと判断し，執筆時点でも実装はしていない．また，サーバーサイドでバッチ処理的な仕組みを作る機能もサポートしていない．こうしたサーバー側にロジックを持たせる仕組みが必要な場合は，7.3.6 項に紹介するような，PHP 言語による MVC フレームワークである CakePHP，CodeIgniter，Yii と INTER-Mediator を統合して利用している開発者の成果を利用するのが，現状での解決策となる．

7.1.2　ブラウザ対応と SEO 対応について

INTER-Mediator は一定以上の機能を持つバージョンの JavaScript を想定しているため，Internet Explorer Ver.7 以前や，従来型の携帯電話等での利用はできない．一般的にはこれらのデバイスの利用率はすでに 10%は下回っている．StatCounter Global Stats[92] では，2013 年 3 月で Internet Explorer 7 以下とその他を含めて全世界で約 6.5%であった．ただし,「その他」の中でも利用可能な場合もある．2014 年 10 月の直近 3ヶ月の発表値では Internet Explorer 7 以下の比率は 0.27%と事実上 0 になっている．ただし，対象ユーザーによって比率は大きく変わり，サポートしていないブラウザでの利用が要求される場面もある．現実に 2014 年になってから，Internet Explorer 7 で利用できないことを「バグ」としてレポートした顧客があった．

ブラウザ対応については，原則として，HTML5 準拠のものとしている．現状の開発リソースでは，細かなバージョンごとの確認ができないため，テストは最新のブラウザでの作業が中心になる．つまり，デスクトップ版の Chrome，Firefox，Safari での確認が中心となり，ある程度の機能ができたところで，Internet Explorer の Ver.8 と Ver.11 でテストをするという流れになっている．Android や iOS 搭載デバイスでの確認も時々行うが，頻繁には行っていない．StatCounter Global Stats からの 2014 年 10 月の直近 3ヶ月の発表値をモバイルを含むすべてのプラットフォームで見た場合，常時テストを行っているデスクトップ版の 4 つのブラウザのシェア合計は約 68%である．一時期はメジャーな製品に絞られていた Web ブラウザではあるが，近年，さまざまな製品が新たに登場している．Internet Explorer の非互換性問題は Ver.8 のサポートをやめればほぼ終息すると考えられる．ブラウザ間の互換性の大きな問題は最新のブラウザの間ではほぼないと言えるものの，動作プラットフォームが違えば予想外の問題が発生する可能性は残る．以前は Internet Explorer への対応が開発上の大きな問題点であったが，現在は多品種化した Web ブラウザ製品のカバレッジを増やす方策を検討する時期に来ている．

INTER-Mediator は DOM モデルによるテンプレート処理を行っている．以前は Google を始めとする検

索エンジンでは HTML ファイルの場合，サーバーから得られたそのままの結果をもとに解析して索引生成をしていたため，DOM モデルに基づいて追加されたテキストは解析対象外であった．しかしながら，2014 年中頃には JavaScript の実行結果に対する解析が Google では行われていることが表明されており [52]，INTER-Mediator で開発したページの場合はテンプレート処理後の結果に対して解析が行われるようになっている[1]．サンプルで作成したページに対する解析依頼を行ってみたところ，データベースから取り出したテキストが，検索結果の一覧に表示されていた．しかしながら，データベースのすべての文字列が取り出されているわけではなく，ページに表示されたものだけである．検索エンジンによるページ内容の取り込みが行われても，データベース内のテキストを確実に解析対象にさせるには，ページ構成上の対策が必要である．例えば，リンクであれば自動的にリンク先を解析対象に含める模様で，この動作はスタティックな HTML ページの解析処理と同等である．一方，ボタンや onclick イベントを利用したページ移動の場合には，状況によってページ移動を行う場合と行わない場合があることが Google からの情報提供で明らかになっている [37]．結果的に，INTER-Mediator で開発したサイトのデータベース内にあるテキストが解析対象になるように，SEO に関わる手法を理解する必要がある．検索エンジンからの誘導を主目的としたような不特定多数のアクセスが要求されるような Web サイトを効率的に運用するには，開発において多大な努力が必要になる可能性がある．

業務ユーザーの場合には SEO とは関係ないことも多く，検索エンジンから検索できる必要性はないと言える．PC の業務利用において，数年サイクルでの買い替えが一般的とすれば，そのタイミングでブラウザの更新も自動的になされることが期待でき，古いクライアント環境で使えないという問題は時間が解決する側面もある．この点からも，本フレームワークは業務システムでの利用の方が問題点は健在しにくい．

7.1.3 利用形態からの考察

エンドユーザーコンピューティングの代表的な事例として Excel によるワークシート作成やスクリプト作成が挙げられており，ビジネス現場での Excel の利用は非常に多い．あらゆるデータを Excel にまとめて管理しているのが一般的なビジネスマンのパソコンの使い方とも言える．データの集積が個人で行われ，同一個人で活用される上では，ファイルに記録する方法での問題はない．しかしながら，ビジネス現場では，情報共有が求められる．簡単なメモからまとまったデータまで共有が求められ，時には統合することも必要になる．

共有が必要になったとき，Excel で作ったワークシートを複数の担当者にメールで送り，それらが別々に更新されるような場面が多々あり [111]，かえって作業効率が低下する場合もある．また，Excel で蓄積したデータの活用は容易でないこともある [69]．Excel を利用しながらもデータの再利用性を考慮されていない「Excel 方眼紙」あるいは「ネ申 Excel」といわれる利用に陥り，データの有効活用を阻害することも問題点として指摘されている [117]．

これらの問題を解消するためにデータベースを使うきっかけとして，INTER-Mediator の利用による業務の効率化が可能であると考える．特に，Web を使うモチベーションの 1 つは，参照あるいは入力するユー

[1] Google が JavaScript の結果を検索エンジン側で取り込んでいるかという問題に対する明確な回答がそれ以前はなく，実験的な手法による調査で限定的ながらも取り込みはされていることが確認され，ブログ等で公表されていた（例えば [31]）．

ザーが不特定多数という場合である．セミナーなどのイベントの参加申し込みや，アンケート回収といった用途であれば，入力専用ページを利用した仕組みを利用でき，必要なら投稿後にメール送信も，宣言的な記述が可能である．

イベントのスケジュールをスタッフ同志で検討するような場合，Excel のワークシートを添付したり，メールの文面でやりとりするのは，受け取った全員がデータ整理をしないけなくなるが，こうした用途の Web アプリケーションを作ることで，データを一元化することもできる．しかしながら，単なるデータリストではないため，実用性の確保には一定以上のスキルは必要ではある．また，1 文書を複数ユーザーでの編集するタイプのアプリケーションは，むしろ Wiki を使う方が手軽に用意できるという側面もあるため，INTER-Mediator を使ったメリットはあまり発揮できないと言える．

7.1.4　ユーザー特性からの考察

Web アプリケーションを現在開発をしているような人たちは，手続き的プログラミングを伴った開発が一般化しており，そうした状況での業務スキルを得ている．したがって，INTER-Mediator のような手続き的なプログラミングが不要という機能には興味を示さないどこか，むしろ拒否反応を示すような場合も見られ，例えば勉強会で説明しても「無意味」と断言してしまうようなエンジニアもいた．つまり，職業プログラマは簡単には受け容れないと言える．Web アプリケーション開発に関わっている人たちで，SE やコンサルティングを行う中で，ユーザーがなるべく参加する，あるいは保守はユーザーが主体的に行うといった方針で進める上では選択される余地はあると考えられる．

一方，Web デザインを行うような人たちの間では，第 6 章の被験者にもあるように，手軽に使えそうな点を評価する一方，データベースのリレーションシップのような複雑な知識の適用になると習得されにくい側面もある．記述部分の作り込みとなると難しい面はあるかもしれないが，記述の修正となると，Web デザインの人たちは触り慣れている HTML ファイルを中心に変更を進めれば良いので，普段の作業に近く，保守を行う要員になりうると考える．

INTER-Mediator は FileMaker コミュニティでの利用者が多い．Web 開発で定評のあるさまざまなフレームワークの多くが FileMaker のサポートをしていないかあるいは困難さがあるのに対し，INTER-Mediator は FileMaker Server を最初からサポートしている．また，2013 年に発売された FileMaker 13 で，WebDirect としてデスクトップアプリケーション上でのレイアウトとほぼ同一の機能を Web ブラウザで利用できる代わりにライセンスモデルを変更して，コネクション単位でのライセンスが必要になった．WebDirect のレイアウトの再現性はかなり高いものの，サーバーに高いスペックが必要になるなど，コスト的にはアプリケーションを購入するのと大差なくなることもあり，導入に踏み切れない事が多い．Web に期待することは，さまざまなデバイスに対応できることであるが，WebDirect は Android に非対応だったり，なぜか Firefox がサポートブラウザになっていないなど，制約がある．INTER-Mediator は FileMaker 5.5 に起源を持つ「カスタム Web」という機能をベースにしており，この機能は現在も利用でき，追加ライセンス等は不要である．

FileMaker での Web のニーズは，他のデータベースと大きく異なる．MySQL 等で開発する場合は，すべてのユーザーインタフェースを Web アプリケーションとして開発するが，FileMaker の場合，社内の業務は FileMaker Pro のアプリケーションから行い，公開情報や，社外とのやりとりに Web を使うといった形態を

取ることができる．社内の場合は利用者数からライセンス数は確定する．一方，社外は不特定となりライセンス数が影響する使い方はしたくないと考えるのが一般的だ．このような形態のアプリケーション開発において，システムの一部分を Web で利用するために，FileMaker 開発者が取り組める可能性がある点で，INTER-Mediator は FileMaker コミュニティでの利用者に注目されている．FileMaker 開発者はデータベースの知識はあっても，HTML やあるいは手続き的なプログラミングの知識を持っていないケースも多い．そうした人たちが，Web 開発を取り組める範囲内で十分な開発ができるようにすることが，INTER-Mediator の存在意義を高めることになるとも言える．

7.1.5 INTER-Mediator の開発における困難さ

INTER-Mediator は結果的には少ない記述で開発ができるものの，これまでさまざまな利用者から聞かれる点として「難しい」ということがある．利用者が感じる困難さの本質的な理由には，INTER-Mediator 自体の概念に関することと，Web およびベータベースの知識といった複合知識の融合の 2 点が挙げられる．その他には，サーバーやデータベースのインストールにさまざな知識が必要になり簡単ではないといった直接的ではない理由もあるが，フレームワークに対する直接的な理由のみをここで論じる．

一般的には Web アプリケーションは手続き的なプログラミングを行い，フレームワークを使う場合にはそのフレームワークで想定された手順を踏むことになる．フレームワークごとに独特な手順がある一方，Ruby on Rails[48] 以降は，「設定より規約（Convention over Configuration）」として，決められたディレクトリにクラス定義のファイルを置くなどの規約を主体にした開発が多くのフレームワークの基本路線になっている．シンプルかつ柔軟性の高い規約が求められ，その流れで開発の大枠が決まる．INTER-Mediator での開発は，そうした規約よりも断片的な宣言的な記述であり，規約に基づく手順があるような Web アプリケーションフレームワーク一般の利用法と異なっている．

また，一方，Web アプリケーションをスクラッチから手続き的なプログラミングを行う場合，データベースへの接続を開き，クエリーを発行し，得られたデータを分解してページ要素として追加するといったボトムアップな流れを行う．INTER-Mediator はこうした作業を不要にしている一方で，1 つ 1 つを組み込むことによる解決を中心に行ってきたプログラマにとっては，なにもしないことがかえって混乱を招く結果になっていることが考えられる．

そして，INTER-Mediator は，繰り返しやリレーションシップの実現に独特の手法を用いている．この点は新しいことだけに，学習が必要である．既知の特定の仕組みに大枠は似ていても細部で違っているような場合では，学習者にはストレスがかかる．こうした習得に関わるハードルがある場合には，「難しい」という感想が出てきてしまう．

Web アプリケーションは複合知識の融合である．HTML や CSS はユーザーインタフェースが主体であるが，見栄えに関わる「デザイン」の領域を含む．そして，リレーショナルデータベースはシンプルな概念ながら，実用度を高めるためには的確な設計が必要である．さらに，JavaScript やサーバーサイドでの手続き的なプログラミングが必要になる．しかしながら，開発者 1 人 1 人の理解は一定ではない．

INTER-Mediator 自体にさまざまな機能があるが，CSS で実現できるような仕組みは INTER-Mediator には含めていない．例えば，テーブルを 2 つ左右に並べたいとしたとき，HTML/CSS に詳しい人であれば，

テーブルの中に左右に並ぶ 2 つのセルを配置し，それぞれのセル内にテーブルを配置するだとか，あるいは CSS の float 属性を使って横に並ぶようにすることは，一般的な知識として知っている．しかしながら，データベースの知識はあっても HTML/CSS の知識がない場合に「できない」「やり方がわからない」ということになる．逆に，データベースの知識があれば，集計結果をビューで取り出すという基本的なことも，デザイン分野の人にはまったく思いつかないかもしれない．

しかしながら，INTER-Mediator での開発の流れが一般的なフレームワークで異なるところが，開発者を混乱させる原因になる．ある機能を実装するとき，ここまでは宣言的な記述でできるものの，ある状況では JavaScript での手続き的なプログラムを書いて対応するという決定をしたとする．マニュアル等でその点を解説をしたとしても，開発時に想定した切り分けが伝わっていないことがむしろ一般的である．開発者は，宣言的な記述ではできないと考えてしまう．さらに手続き的なプログラムを作るとしても，そこで独特な概念を理解しつつプログラムを組み立てないといけない．これらの障壁が困難さにつながる．

これらの点を踏まえると，困難さがあることとそれを克服することは，INTER-Mediator 特有の事情をいかに効率良く学習してもらえるかという点に尽きると考える．開発者の知識レベルはさまざまだが，大きく分けて HTML を中心とした「Web デザイン」と「データベース」の 2 つが主要な領域である．これらの領域の人たちに何を学習して，何を理解すればいいのかを効率的に伝達することで，困難さは一定範囲では克服できると考える．

7.2　フレームワークの利用者による評価

INTER-Mediator を実際に開発に利用した方々に，開発フレームワークとしての評価してもらうために，インタビューを行った．インタビューを行った方々は表 7.1 に示した．本節では，インタビュー結果を元にしたユーザー評価を述べる．インタビュー対象者のすべてはデータベースとして FileMaker を利用している．A, B, C の 3 名が，INTER-Mediator を利用して開発を行っており，その中の A, B が立場の上ではエンドユーザー開発者である．D 以外の 3 名については 2014 年 9 月〜10 月にかけて対面でのインタビューを実施した．

表 7.1: インタビュー対象者のプロフィール

対象者	現在の業務
A さん	芸術系大学の教員，学内スタッフの業績管理を Web で入力や公開できるシステムを開発中
B さん	以前はデザイナーだったが FileMaker を中心とした社内システムの管理を行う．INTER-Mediator によってシステムの一部を開発を試みたが，いったん中止をした
C さん	FileMaker によるソリューション開発を業務とする．顧客より CDML ベースの Web アプリケーションの改定を依頼され INTER-Mediator で構築した
D さん	FileMaker データベースを INTER-Mediator で公開できるコンバータを開発．INTER-Mediator のコミッターでもある（メールによるインタビュー）

7.2.1 INTER-Mediator に注目したきっかけ

INTER-Mediator を選択した理由として，FileMaker Server に対応している点はすべての対象者が指摘する．加えて,「FileMaker Server のカスタム Web[35][36] での拡張を考えたが，PHP だけでなく手続き的なプログラミングの経験がなかったこともあり手がつけられなかった」といった点がある．また，手続き的なプログラミングよりも宣言的な記述を主体にしていることなどから「敷居が低そうでとっつきやすかった」といった印象を持った点も挙げられた.「ラピッドプロトタイプが可能そうに見えた」といった意見もあった.

FileMaker を選択する積極的な理由は，システム開発の「手軽さ」の一言に尽きる．しかしながら，FileMaker で溜め込んだデータの Web からの利用となると，手続的なプログラミングが必要になることや，高いスペックを要求するサーバーが必要となり，FileMaker 単体で実現された手軽さが薄らぐという印象を持つ．INTER-Mediator の利点は，FileMaker 利用者にとっては FileMaker に抱く印象を Web 開発でも実現することを印象付けていると言える．

INTER-Mediator は現在開発されているオープンソースソフトウェアであり，開発が活発に行われていることや，開発者自身にコンタクトを取れることが挙げられた．さらに突っ込んだ意見として,「開発が以前に止まってしまったものは今使えるかは分からない．しかしながら，現在進行中であれば問題点があったときの対応も可能だろうし，場合によっては業務依頼として修正や助言を求めることもできる」といった意見もあった.

INTER-Mediator は GitHub をレポジトリとして利用しており，プルリクエストなど開発の進行をイベントとして見えるような仕組みが使える．また，東京での開催が中心になるが，勉強会などのコミュニティ活動を継続して行っており，これらの複合的な理由により，活動が活発で利用に値するオープンソースであると決定付ける要因を持っていたと言える．

7.2.2 INTER-Mediator が評価できるところ

実際に INTER-Mediator を使用して開発を行った上での評価できる点については,「プログラムを書かないでもいい点は評価できる．プログラムをたくさん書かないといけないとすれば，本質的ではないところでパワーを消費する．短い記述で完結するためスピード感があり，Web アプリケーションをすぐに作りたい人向けには良い.」や,「組み立てキットのような感覚で使える」といった意見があった．手続き的なプログラムよりも宣言的な記述を主体にしている INTER-Mediator の特徴が評価された．エンドユーザー開発の現場で支持される理由は，結果に早く到達できるといった点である.

また，FileMaker のカスタム Web では PHP のプログラムを記述する必要がどうしてもあるため，より多くの学習をする必要が出てくるが,「プログラミングがまったく必要なく，設定の記述のみでデータベースアクセスができるようになっている」や,「カスタム Web を即席にすぐに使えるようになっている」いった意見もあった．FileMaker を選択した上で，Web 機能のうちのカスタム Web を使う必要があったとしても，ミドルウエア的なライブラリである FX.php[47] や製品に標準のライブラリを使い，かなり低いレイヤーからのプログラミングが必要になる．INTER-Mediator は開発を効率化するフレームワークとして，プログラミングをしなくてもデータベースアクセスの基本機能が利用できる点が実現されており，評価されている.

一方，INTER-Mediator の内部を知るコミッターは，評価できる点として，インストールが容易であること，エンクロージャーおよびリピーターにより HTML タグを拡張せずに生の HTML ファイルで表示用のファイルを記述できること，さらにはページ上での編集結果がクライアント間でリアルタイムに共有できる点を挙げている．独自拡張のない HTML の利用を好ましい機能として評価しており，それを実現する仕組みとしてエンクロージャー/リピーターを識別するテンプレートの動作への評価につながっている．

7.2.3　INTER-Mediator を利用する上での問題点

開発をした上での問題点として,「丁寧かつ詳細に説明されたドキュメントやケーススタディがない」など，ドキュメントの不備を指摘する．固有の機能に関して調べても分からないことが出てきたとしても,「疑問が発生しても解決しないことも多かった.」といった状況になる．現実に，ドキュメント類については確かに不足はあり，開発されてもドキュメント化されていない部分があることも事実である．加えて，INTER-Mediator を使うノウハウ的な情報が開発コミュニティ以外からはほとんど発信されておらず，メジャーなオープンソースのように，利用者がさまざまな視点で情報をブログ等で共有する状況にまでなっていない．この点は「開発コミュニティがまだまだ育っていないという印象がある．開発者の受け皿となるコミュニティが必要である.」という意見に集約されていると考えられる．開発側からの積極的な情報提供はもちろん必要だが，多くの利用者がノウハウを公開するようなコミュニティを醸成する必要がある．

一連のインタビューで，現状のサイト等で得られなかった情報を以下の箇条書きにまとめた．バインドの手法などの基本的な機能については，わからなかかったこととしては挙げられなかった．むしろ，INTER-Mediator 以外のソフトウェアとの相性や利用方法といった情報がない点に不満を感じている．現状の開発コミュニティからこうした情報を提供できれば理想的ではあるが，この種の情報はコミュニティが成熟しないと豊富に流通するような状況にはならないのではないかと考えられる．

- 他のフレームワークのとの連携手法
- INTER-Mediator が持つ認証機能と他のフレームワークを併用した時の認証機能の関係
- Bootstrap，Foundation などの CSS 系フレームワークの利用
- FileMaker に関する特殊性や注意点についてのドキュメント
- データベースエンジンごとに選択したことでのメリットやデメリット
- ボタンの位置やスタイルのカスタマイズ
- CSS の設定を利用したカスタマイズの方法

7.2.4　スクリプトの記述に関する意見

定義ファイルはキーと値の組を指定することが必要であり，PHP のプログラムを書く必要はなく，定義ファイルの記述自体は宣言的であることはすでに主張した通りである．しかしながら，定義ファイル自体の編集を，エディタで直接行う利用者も多い．開発コミュニティからは，PHP のファイルを直接編集する場合は，PHPStorm などの構文チェック機能を持った統合開発環境の利用を強く勧めている．

定義ファイルの記述については,「プログラムを知らなくてもそれなりに書けそうだと感じた」とは言う

ものの，カンマの有無でうまく動かないで試行錯誤したということもあり，結果的に少しではあるが配列の記述に関しての PHP の学習をする必要性があったと話す．なお，この時点ではまだ定義ファイルエディタの実装を行っていなかった．定義ファイルの編集作業に関しても，ドキュメントなどのなんらかのガイダンスをしっかりと行う必要性があるとも言える．

FileMaker のスクリプトと JavaScript によるプログラムの違いについても意見を伺った．FileMaker のスクリプトは，アプリケーション内のユーザーインタフェースを通じてステップを選択して追加し，パラメータはダイアログボックスで選択選択入力するなど，使い勝手は良い．FileMaker の開発を行っている対象者の 1 名は FileMaker のスクリプト作成経験があるものの，Web サイト開発や JavaScript のプログラム経験はなく，INTER-Mediator の開発に際して本格的に Web 開発を使い始めた．これらのスクリプトの違いについて，「FileMaker のスクリプトは，絵を描くように構築でき感覚的に見渡すことができるが，JavaScript は文字が並ぶことで追うのが大変になり，なかなか理解が進まなかった.」と話す．FileMaker のスクリプトのような開発ツールのサポートがあるような場合には，手続き的なプログラミングを習得していなくてもスクリプトの開発はできることを示している．INTER-Mediator は現在は開発ツール自体を持たない開発環境であるが，スクリプトが作りやすくなる点もメリットであり，開発ツールによってより開発が促進される点は今後の課題である．

7.2.5 INTER-Mediator の学習について

インタビューをした方々の多くは 1〜2ヶ月，業務の合間などに学習をすることで，概ね基本的なことは理解できたと話す.「Web サイトやコンテンツ開発の経験があり，すでに HTML/CSS の知識はあるので，特になにか別の知識を学習しなくてもテキストを読み進めれた」と話すように，HTML/CSS の知識と，データベース知識（この場合は FileMaker）があれば，INTER-Mediator に取り組むことができると言える．

しかしながら，データベースの知識はあっても，HTML を始めとする Web 関連の知識も併せて学習するとなると，「学習する上ではとにかく用語に混乱された．その用語が，HTML のものなのか，JavaScript のものなのかも最初はわからず混乱する.」といった状況になる．そこに INTER-Mediator に関する事情も加わることになる．Web の世界は複合的な技術によって成り立っているだけに，この点は根源的な問題とも言えるが，単に JavaScript といっても多様な概念がある．例えば，手続き的なプログラミングをこれまで経験がない場合は，「メソッド，プロパティ，クラスといった用語が最初はわからなかった．また，ノードという考え方も理解するのが最初は困難だった.」というように，単に文法だけの学習では済まされないことになる．FileMaker のエンジニアにとってはこうした多様な Web に関する知識の習得が大きな障害になる．

7.2.6 INTER-Mediator に対して期待したいこと

INTER-Mediator に今後望むこととして，「現実の業務に近いサンプルが用意されていることである．そのサンプルをある程度修正すれば自分が望む形式になる…という状況になっているのが理想的である．そのサンプルは，たとえばブログといったような実用的なサンプルである必要があり，機能を見せるようなサンプルではない.」といった，半完成品，あるいは完成品のソースコードの入手を求める意見があった．実際

のビジネス現場で必要とされるような形式のアプリケーションがあれば，それを少し変更することで業務適用できる可能性がある．エンドユーザーの現場で求められる開発素材としては，そこまでの手軽さが求められている．

一方，データベースの教育に利用する可能性についての構想もある．「データベースの講義で Web アプリケーションの説明をしたものの，Web アプリを作る上ではデータベースに関係ない PHP の説明ばかりになり本来の説明ができない．可能であれば INTER-Mediator ベースで取り組みたいと考えている．INTER-Mediator だと HTML との親和性が高く，データベースの学習の一環で取り組む場合にはやりやすいのではないか．」という意見もある．INTER-Mediator をセットアップした状況であれば，HTML の属性に記述を加えることで，データベースのデータを表示したり変更することができるため，データベースがページ上に表示される意味合いを少ない作業で説明でき，学習効果の増大に寄与できる．

7.3　フレームワークの利用実績

INTER-Mediator を実際に Web システム等の開発に利用されて運用している実績を，名称や内容を公開できるサイトを中心にここで紹介する．また，具体的な利用者名や URL 等を公開できないものについては，概要のみを本節の末尾で紹介する．

7.3.1　ふち無しはがき印刷本舗（年賀状・暑中見舞いオーダー受付管理）

「ふち無しはがき印刷本舗」は，年賀状や暑中見舞いを中心としたはがき印刷を受け付けるサービス，いわゆるオンデマンド印刷のサービスを行っている（図 7.1）．本サービスの強みは，市販のはがきをカットすることなくふち無し印刷ができる点である．顧客ごとに異なるデザインで印刷を行うために，個別にファイルのやりとりや，デザインの確認，振込み確認などの作業が発生する．従来は FileMaker のデータベースを利用していたものの，通常のメールでやりとりをし，ファイルのやりとりの Web サービスを利用するなど，顧客とのやりとりを別々のサービスで行っていたため，やりとりの記録やファイルがどのオーダーのものなのかを特定するのに手間がかかるといった問題が発生した．しかも，業務は 1 年の一定期間に集中することもあり，混乱が発生しやすい状況であったとも言える．

同社の担当者は，以前に勤務していた会社での FileMaker 開発経験はあったものの，Web 開発の経験はなかった．また，同社のサイトは Jimdo を利用して運用しており，コンテンツ管理は Jimdo を利用することが決定していた．その上で，顧客のオーダーの進行管理を行うための動的なサイトを構築する業務を筆者が請け負った．まず，データベースはオーダーだけでなく，内部での印刷を中心としたさまざまな業務にも適用するニーズもあり，その改良や追加開発を自身の手で行いたいといったこともあって，FileMaker をデータベースとして運用するのが前提となった．オーダーの進行管理は受注側での情報だけでなく，発注する顧客が入力する必要もある．さらに，ファイルの入稿もある．これらの作業を顧客は Web でログインをすることで実現する必要があった．

まず，社内での情報共有のために FileMaker を利用する点は，大きな問題はない．顧客が利用する Web 側の機能については，認証が可能なこと，そしてレコード単位で個別のユーザーに対する認可が可能な点

図 7.1: トップページ

がすでに INTER-Mediator で実現していた．ファイルのアップロードについては発注時点では作りかけのコンポーネントがあったが，それを完成させると同時にプログレスバーの表示等にも対応できるよう，本開発と並行してコンポーネントを充実させていった．フレームワーク側に足りない機能を補充することを含めて，2013 年の 1 月に発注を受け，2013 年 5 月にカットオフとなり，その年の暑中見舞いの受注より利用した．その後，保守開発を継続している．

初めて発注を行うときに，ユーザーアカウントを作成する．本サイトでは，メールアドレスとパスワードでのログインができるようにしている．ログイン後に，印刷枚数や宛名印刷の有無などの最終的なオーダーを作成して，注文を確定する．その後，入金とその確認，データ入稿は並列的に行われ，これらが揃った上でデザインを作って校正を行い，印刷をして発注するといった進行の管理が発生する．発注した顧客は，ログイン後に参照できる図 7.2 のようなページでその進行状況を見ながら，入金あるいはファイルの送信などを行う．受注側ではスタッフが，入金確認やデザイン結果のファイルの登録などを，FileMaker Pro を用いて行う．このページ内には，顧客とスタッフが連絡を取り合う仕組みも組み込んだ．書き込み内容はメールでも送信される．受注側のスタッフは，特定のオーダーに対する顧客とのやりとりを FileMaker 上でまとめて時系列で FileMaker で確認でき，その場で返信できるようにした．こうして，スタッフ側はオーダーの全貌が FileMaker のレイアウト上で確認できるようになり，業務の効率化に寄与した．

発注前の利用者のために，図 7.3 のような費用見積もりのページも作成した．こちらは認証が不要なページである．動的な処理はサーバーからマスター情報を取り出すだけにしており，ユーザーが入力した情報はデータベースには残らない．また，本ページは単に見積もりだけでなく，発注に関する情報や価格リストも掲載し，本サービスの一般向けサイトに統合する必要がある．Jimdo は HTML での記述ができるので，ページ全

図 7.2: 印刷オーダーと進行の確認画面

体は Jimdo 側のコンテンツとして作成し，見積もり処理の部分だけ iframe タグを利用して INTER-Mediator を利用したサーバーより生成している．

顧客のオーダー時には印刷枚数などのいくつかのパラメーターが入力される．それを元に価格を計算する仕組みは複雑であり，例えば印刷枚数は，枚数に応じて単価が異なると同時に，発注時期に応じて割引が発生する．発注内容の別の明細項目に依存するものなどもあり，ロジックは複雑である．こうしたロジックは，本案件では FileMaker のスクリプトで実装することにした．ただし，見積もり時には FileMaker データベースへの入力はしないため，見積もり時にはクライアントサイドで実行される JavaScript でロジックを組み立てた．

開発結果より，同社の担当者は「INTER-Mediator を使うことで開発時間は短縮できることを聞いていたので，ぎりぎりまで内容の検討に時間をかけることができました．思った以上に短期間で Web アプリケーションが立ち上がり，またその後の修正も容易なので驚いています」と話す．

オンデマンド印刷サービスは数多くのサービスがあるものの，発注情報，ファイル入稿の手段，スタッフ

図 7.3: 印刷オーダーの見積もり画面

とのやりとりといったものが，顧客に対して統合的に提供されているような状況はあまり見られない点に西ヶ谷一志さんは懸念していた．他社サイトが使い勝手が悪くなっている事情は確定的にはわからないが，例えば，システムの構築後に機能を追加することがうまくいかず，受発注とは別にファイル送信サービスを併用するなどの措置を取っているといったことが原因として考えられる．しかしながら，INTER-Mediator を利用したサイトについては，他社のサイトを見て感じた懸念は払拭され，「使い勝手の上でも十分に競争力のあるサイトができて満足している」と高い評価をいただいている．

7.3.2 FMPress Publisher（FileMaker データベースを変換し Web アプリ化）

株式会社エミックは，FileMaker Server のホスティングサービス「FM Press[114]」を提供している．FileMaker でのシステム開発は十分に成果は出せるが，FileMaker データベースを Web ブラウザから利用する仕組みには決定版といったものがない状況でもあった．そこで，「実務担当者にとって役立つプロダクトおよびサービスを展開して，既存の自社サービスに付加価値を追加していきたかった」といった理由で FileMaker のデータベースを INTER-Mediator 上で稼働できるようにする手法に至り，データベースの設計情報をより Web 公開に必要なファイルを生成する「Publisher[115]」として公開した．

FileMaker のデータベースを Publisher で公開するためには，まず FileMaker のデータベースレポートの機能を利用して，設計情報を XML ファイルに書き出す．その書き出したデータを利用者向けに用意されたサ

イトにアップロードすると，自動的に各レイアウトに対応するページファイルと定義ファイルが生成されてダウンロードができるようになっている．データベースを運用している FileMaker Server にアップロードすればそれですぐに利用できる．コンバータは PHP で記述されたスクリプトで作成されており，JavaScript の追加ライブラリも含まれている．

図 7.4 に示す FileMaker のレイアウト画面をコンバートして Web ブラウザで参照したのが図 7.5 である．実用上，ほぼ問題がないくらいに FileMaker のレイアウトが再現されている．FileMaker の最近のバージョンではレイアウトテーマを CSS で記述するなど内部動作が CSS 基調であることが推測される．Publisher は FileMaker のレイアウトを CSS により再現していて，画面の見た目の違いは非常に少ないものになっている．FileMaker では任意のレコードセットと，その中の 1 つの選択レコードをテーブルオカレンスが保持し，それをレイアウトに表示するといった仕組みがある．INTER-Mediator にはそれに相当する機能はないが，Publisher では FileMaker との操作上の互換性を確保すべく，レコードセットの保持や選択レコードの扱いができるような Web アプリケーションを生成している．

INTER-Mediator を選択した理由として，FileMaker Server のサポート，そして FileMaker の古いバージョンとの互換性に必要な PHP 5.2 のサポート，さらに開発が継続的に行われてきている点を挙げる．さらに，他のフレームワークに比較して，デプロイメントが容易である点や，CRUD の処理に対応していること，ポータル（FileMaker での関連レコード展開）に相当する機能があったことも選択した理由に挙げられている．現在の FileMaker 13 は WebDirect という機能で FileMaker のデータベースをそのまま Web 公開できる機能を提供しているが，WebDirect が登場したとしても，Android での利用が可能な点など，Publisher の方にも有利な点があるとしている [112].

Publisher の開発は，INTER-Mediator のコミッターによって行われており，開発成果の還元もされている [116]．筆者も Publisher に対して全面的なサポートを行っており，2013 年 7 月に開催されたプレス向け発表会で同席をしている [113]．この発表会で Publisher の開発を公表した結果，いくつかのニュースサイトで記事が掲載されている [18][67][145].

Publisher では，FileMaker のすべての機能を再現できてはいない．計算フィールドの即時更新ができないことや，スクリプトのステップの部分的なサポートといったことが課題となっている．また，FileMaker の制限として設計情報に含まれない情報があり，しかもレイアウトのパートの区切り目の情報が得られない点は大きな課題になっている．同社はこれらの制限に少しずつでも対応していく予定である．

FileMaker データベースを INTER-Mediator で公開できる形式に変換はできるものの，現状では本格的に利用している目立った事例はまだ出てきておらず，テスト運用までの利用になっている．Publisher を使えば一定の機能を持ったアプリケーションがすでにできていることになり，見方を変えれば保守開発からスタートすることになる．現実に利用者側で行うことができそうな作業としては「レイアウト名やフィールド名の変更」「フィールドの配置場所の変更」「フィールドの追加」「ページファイルの変更・更新」を想定している．フィールドの追加を除いて，HTML の修正を始めとする宣言的な記述の修正や追加でできることである．FileMaker はテーブルのスキーマ定義を行うユーザーインタフェースがあり，一般的な SQL データベースに加えてスキーマ修正は容易であり，フィールド追加が手軽にできるのは FileMaker の 1 つの特殊性である．

図 7.4: 変換対象の FileMaker のデータベース

図 7.5: Publisher で変換した結果を Web ブラウザで参照（開発中のバージョンを利用）

7.3.3　naha_regi（Web レジスターシステム）

Web レジスターシステムの「naha_regi[?]」は，オープンソースで開発しているサーバーベースのレジスターシステムである．EC サイトでの通販を行っているような人たちが主な利用者として想定している．サーバーサイドでは PHP 上で稼働するフレームワーク Yui を利用し，ユーザーインタフェースの構築に INTER-Mediator を利用している．

商品や個数などを入力して，図 7.6 のような売り上げデータを伝票形式を作る「レジスターモード」機能があり，ここでの実装に INTER-Mediator が使われている．レジスターモードで個別の売り上げを入力する．商品は EC-CUBE との連動ができ，E コマースサイトとの併用も考慮されている．また，顧客リストを定義して，顧客ごとの購買履歴を残す機能もある．

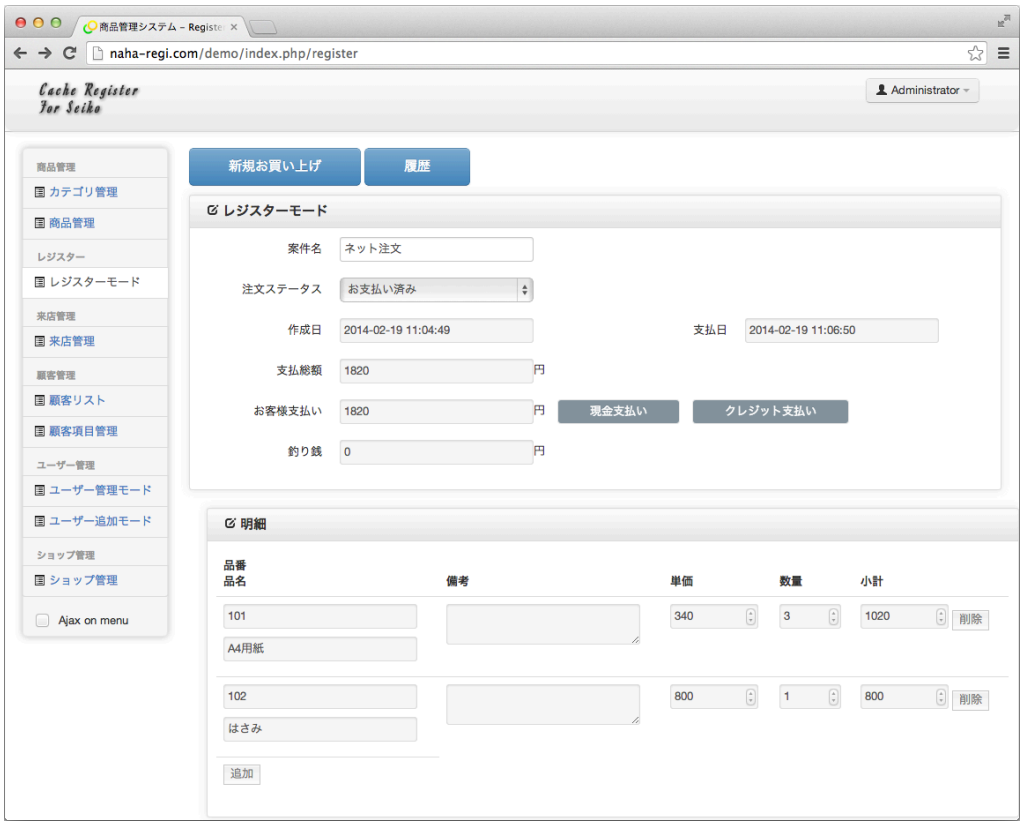

図 7.6: naha_regi の入力画面

7.3.4　筆者自身が利用するサイトでの開発事例

まず，第 6 章で紹介したユーザー実験においてのコンテンツの表示，試験の実施，そして試験結果の参照ができる Web アプリケーションを INTER-Mediator で開発して利用した．学習コンテンツ自体は単にフィールドの内容を画面に表示しているだけである．ページを表示したときの時刻を JavaScript の変数で記録し，読み終わった時点で被験者にボタンを押してもらう箇所では入力専用のレイアウトを作り，参照時間を計算する基礎データを残している．試験問題の穴埋めの箇所は，それぞれフィールドを割り当てており，あらか

じめ各被験者のレコードを作っておくことにより，試験解答のページは特定レコードのフィールドのデータを編集するページとして作成した．

実験実施サイト以外に，図 7.7 に示す筆者自身の出版物の一部や，自費出版物の販売用サイトを INTER-Mediator で構築した．コンテンツ自身はログインをして利用できるようになっている．コンテンツには大きく分けて，HTML で生成して Web ブラウザで参照するものと，PDF あるいは ePub でダウンロードして参照するものの 2 種類がある．データベースには MySQL を利用し，プロバイダの VPS 上で運用している．

図 7.7: コンテンツサイトのトップページ

ログインをすると，自分が参照する権利のあるコンテンツが図 7.8 のように一覧される．この図ではすべてのコンテンツに権利がある状態であるが，権利がなくかつ販売しているコンテンツについては，購入情報も表示される．購入は PayPal への購入ボタンが含んだスタティックなページだが，購入した利用者に対してはその情報をページ生成時に非表示にするようにした．なお，PayPal での購入の結果，図 7.8 に戻ってアクセス権を付与するレコードが作成されるようになっているが，そのとき，このページを認証済みな状態になっていることを想定している．PayPal に移動してから時間が経過してから決済をしているのか，認証されていない状況でこのページに戻る場合もある．そのため，後から手作業でアクセス権を付与する作業がときどき発生している．

HTML で参照するコンテンツは，チャプターごとに HTML の文字列をデータベースのフィールドに記録している．1 つのフィールドの文字列としては長いものになるが，フィールドの値を要素にバインドして参照できるようにしている点では，一般的な使用方法である．一方，ePub や PDF のコンテンツについては，コンテンツへのアクセス権を付与するレコードに対応付けて認証を確認した上で，サーバーサイドでファイルを生成している．PDF は利用者の名前とメールアドレスを全ページのフッタに挿入するサーバーサイドのプログラムを実行している．ePub についてはアーカイブしていない状態のディレクトリをサーバーに用意し，利用者名とメールアドレスをページ内に埋め込んだ上でアーカイブしてダウンロードするようにしている．

ユーザー管理のための機能も Web アプリケーションで用意した．このページは特定のグループのユーザー

図 7.8: 購入コンテンツの一覧ページ

のみがログインでき，購読者はログインはできない．ここではパスワードの手作業によるリセットと，アクセス権の手作業による付与が作業として発生するので用意をした．このページの原型となるユーザーやグループ管理のための Web ページは，INTER-Mediator のサンプルコードとしても配布している．

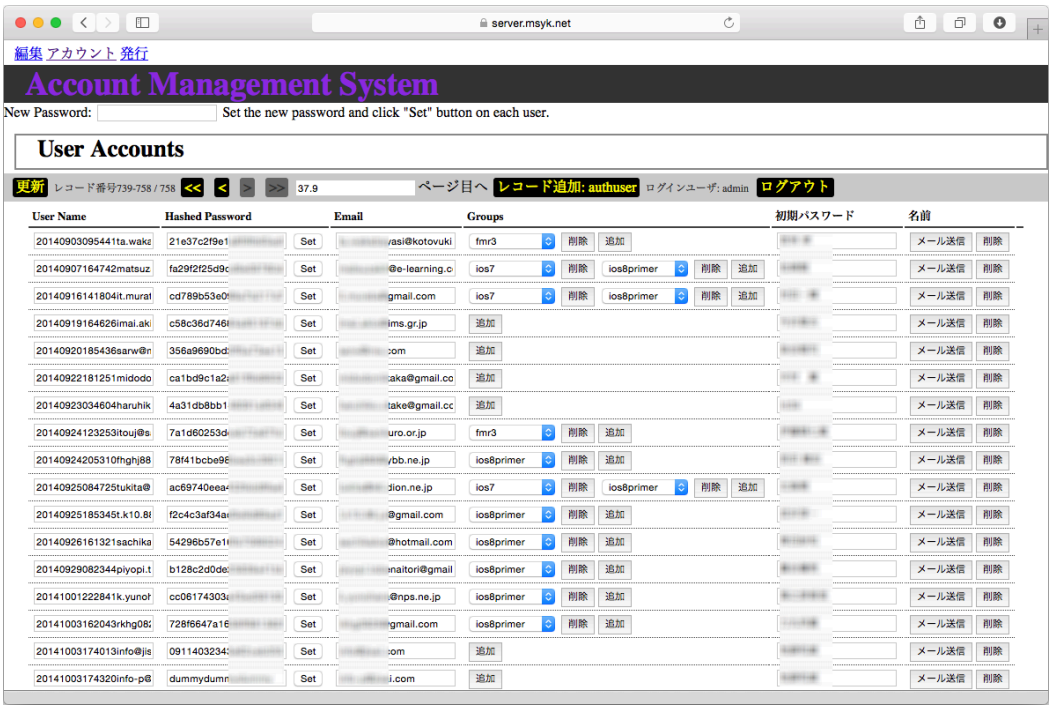

図 7.9: 登録ユーザーの管理ページ

7.3.5　その他の開発・運用実績

職員が 200 人近くいる病院の看護師の勤怠管理のためのシステムを 2012 年に受注して開発をした．多くの開発は FileMaker で行ったが，一部に FileMaker では作りにくい要件があったため，その部分を INTER-

Mediator による Web のシステムで構築した．勤怠管理は主に，先々のスタッフの常勤，夜勤，あるいは休日といった勤務割り当てを行うものである．縦軸にスタッフ，横軸に日付が並らび，その間のマス目に勤務形態をが表示され，その一覧を見ながら管理職員がスタッフの勤務計画を行う．FileMaker で作りにくい理由は，こうした表計算的な編集画面が要求されたからである．FileMaker のレイアウトは縦方向にレコードを並べるのは容易に可能だが，横方向に不定数のレコードを並べることが困難になる．HTML であれば，そうした構成も容易であり，マス目に相当する要素とレコードのフィールドをバインドすれば良い．このとき，スタッフと日付から，何行目，何列目かは判定できるので，要素の座標値を計算フィールドで求め，要素の CSS の position の値を absolute にして表計算的な配置を行った．また，本件は,「ファンクションキーでの割り当て変更」が要求にあったこともあって，FileMaker 単体でのファンクションキーの制御が完全にできないことも含めて Web での開発を行った．

会員企業に対して情報提供を行うためのサイトを 2013 年に構築した．FileMaker の CDML で作られたサイトを最新版の FileMaker に対応させるため，Web については全面的な改修が必要となり INTER-Mediator が選択された．Web の部分は筆者が開発したが，顧客との間には開発会社が入っており，開発会社からの依頼があったときに，認証機能を持ち，ユーザーごとの認可が実装できる点から INTER-Mediator の利用を提案したところ，受け入れられた．システム上の複雑な点は，提供側が，特定のカテゴリの会員企業あるいは特定の企業に対してのみ文書あるいはそのセットを公開するといった仕組みの構築である．提供側のスタッフが FileMaker での作業をするといった要求があったため，FileMaker のオブジェクトフィールド（ファイル等を記録できるいわゆる BLOB フィールド）と Web の連動を組み込む必要があった．また，CDML で構築されたサイトは言語そのものが低機能なために余計なプロセスや記述が多々発生する．それらを解析して適切な動作に持ち込む一種のリファクタリング的な開発であった点も困難であった点である．

海外出張申請承認システムを INTER-Mediator で開発して導入した企業もある．本案件も CDML で作られたサイトを新しい FileMaker でも稼働させるという目的のものであり，ページ構成はまったく変更しないで作成するといった要件があった．サイトの動作は，社員が海外出張の申請を行うと管理職が承認あるいは追加の情報を求めるなどといった，簡単なワークフローである．ページ構成を変えれば容易に実装できるところも従来と同一の構成にするために JavaScript でのプログラムを記述したり，スタイルシート属性をプログラムで制御して，画面が切り替わっているように見せるといった実装をしないといけない場面も発生した．

7.3.6 他のフレームワークとの統合

INTER-Mediator は，PHP 言語で開発された MVC スタイルの Web アプリケーションフレームワークとの統合実績がある．いずれも，モデル，コントローラーは MVC フレームワークの仕組みを活用し，ビューの合成において INTER-Mediator を利用している．

「IMCake[146]」は，PHP 言語で開発されたフレームワークである CakePHP[13] と INTER-Mediator を統合したものである．INTER-Mediator のテンプレートの仕組みをサーバーサイドで実現しているため，生成されたページの HTML にはデータベースのフィールドが追加された状態でクライアントに送信される．検索エンジンによっては JavaScript の実行後の結果を取得できない場合があるが，そのような検索エンジンに対してデータベースの内容を取得させたいページを作成したい場合に利用できる．なお，サーバーサイド

でのコントローラーやモデル等は，CakePHP 上での手続き的なプログラミングが必要になるため，開発には CakePHP に関する知識も必要となる．

他には，CodeIgniter[17] と Yii[104]（いずれも PHP ベースのフレームワーク）と統合を実現している [109][110]．Yii との統合結果をベースにして，7.3.3 項に紹介したレジスターシステムの「naha_regi」が開発されている．

7.4　今後の発展に向けての課題

現状の INTER-Mediator は，Web アプリケーションを開発するための基本的な機能は揃っていると考える．バグを潰したり，レアケースでの不具合の解消はもちろん随時続ける．一方，インタビューでも明らかになったように，直接的にはドキュメントの不足があり，早急に改善する必要がある．

エンドユーザー開発で利用価値を高めるためには，宣言的な記述で一定の機能を利用できる仕組みは随時追加する必要があると考える．ただ，闇雲に機能を追加するというのもかえって複雑になるだけである．今後の展開を検討するにあたり，O'Reilly が主催するフロントエンドツールのカンファレンス「fluent2014[78]」での中で「フレームワークはコミュニティのベストプラクティスと共に進化する」といった発言が Ember.js の開発者よりあった [59]．Web 開発者がやりたいことを「プラクティス」として抽出し，フレームワークの路線として強くサポートする必要があると解釈できる．プラクティスは「よく利用される手法」や「典型的な解決策」とも言える．一方で「パターン」と表現しないのは，パターンとしてまとめられていることのような抽象度を求めていないことの現れである．INTER-Mediator でマスター/ディテール形式のユーザーインタフェースを宣言的な記述だけでサポートするのは,「マスター/ディテール形式」が Web あるいは一般的な業務システムで使われている典型的なプラクティスである．こうしたプラクティスの抽出と効果的な実現が可能な実装を求めることで，より利用価値を高める必要がある．

第8章　まとめ

Web アプリケーションフレームワークの INTER-Mediator は，独自拡張のない HTML で記述するページファイルと，データベースとのやりとりを宣言的な手法で記述する定義ファイルを記述することで，データベースの内容を読み書き可能な Web アプリケーションが構築できる．開発作業や開発後の改変作業を宣言的な記述によって行える．一般的なフレームワークでは手続き的なプログラムの作成や修正が必要になり，言語でのプログラミングを学習する必要がある．こうした手法は柔軟性は高いものの，スキル獲得には時間がかかる．これに対して宣言的な手法であれば，学習コストが低くなり，エンジニアでないような自身の業務を持つエンドユーザーや Web デザイナーでも取り組むことができる．本フレームワークを利用して開発したシステムに対してエンドユーザーで保守を行えば，保守作業を専門会社に発注しなくても取り組めるため，保守のための予算が限られた現場でも保守開発を行う機会が増え，業務の変化に応じたシステム構築の継続を可能にする．

保守作業が宣言的な記述の変更でできることを示すために，まず，Web アプリケーションの改変が発生する状況を「ページ要素」「データベース要求」「単一フィールド要求」「ユーザーインタフェースのカスタマイズ」「データベース応答」「スキーマ変更」と 6 つに分類し，本フレームワークで作られた Web アプリケーションをそれぞれの場合での改変作業を検討した.「ページ要素」「データベース要求」「単一フィールド要求」については一般的なフレームワークでは手続き的なプログラミングが必要になるのに対し，本フレームワークでは宣言的な手法で改変が行われることが確認できた．それ以外の分野は，手続き的なプログラミングが必要か，もしくはデータベース管理の知識が必要になり，エンジニアによって行われることが想定できる．すべての保守作業ではないものの，軽微な変更をエンドユーザーサイドで即座に対応できることで，経費節減だけでなく，業務への最適化のスピードアップにもつながる．

Web アプリケーションに必要なさまざまな機能を本フレームワークでは宣言的な記述で利用できる．そのことを実現する特徴的な実装の 1 つがバインディングである．テキストフィールドを構成する HTML 要素の属性にテーブル名とフィールド名を記述すると，ページを表示するとそのテキストにはデータベースのテーブルのフィールドにある値が表示される．ページ上で修正すると値をフィールドに書き戻す．加えて，複数のレコードを表示できるように，1 レコード分に相当するテーブルの行をレコードの数だけ複製する仕組みを採用した．1 レコードに対応した繰り返す要素群を「リピーター」，リピーターの親ノードを「エンクロージャー」として，例えばテーブルであれば TR タグ群がリピーターで TBODY タグがエンクロージャーとなる．テーブルの場合は自動的にエンクロージャーとリピーターのセットを識別可能なので，バインドの記述を行うだけで繰り返し範囲などの記述はしなくてもレコードの数に応じた繰り返し表示ができる．

本フレームワークを利用して開発するときの記述方法に関して，Web 業界で働くデザイナーを中心とした被験者 12 名に対する学習可能性の実験を行った．学習コンテンツを読んで学習し，その後に試験を行った．被験者は手続き的なプログラミングは日常的に行っていなかった．実験の結果，試験時間が極端に短

時間で試験もほとんど回答できなかった 4 名と，1〜2 時間の学習と試験を行い試験結果は 1 名を除いて半分以上の正解を得ている 2 つのグループに分かれた．前者は学習をあきらめたグループである．後者のグループでは得点が一様に分布していることから，学習した結果が試験に現れていると考えられ，学習は可能であるという結論が得られた．また，学習すべき基本概念に応じた分類ごとの正答率を求めたが，いずれの分類でも 8 人中 6 人は 50%以上の得点を得ており，分野ごとに大きな偏りはなかった．学習できた被験者とできなかった被験者のいずれもプログラミングの経験があったが，学習できた被験者群の中ではプログラミング経験があるほど高い得点になる傾向があり，プログラミングの経験が理解を助けるか，あるいは被験者の意欲の高さとの相関があることが考えられる．また，データベースに対する知識や経験にも影響することが，試験後のアンケート結果からも得られ，HTML の記述だけではなくデータベースの知識の重要性も確認できた．

本フレームワークを実際に使用して開発を行ったエンドユーザーや開発者にインタビューを行った．手続き的なプログラムが必要なフレームワークに比べて，手軽に手早くアプリケーションが作成できる点や，HTML に直接記述することの分かりやすさなどが高い評価を得られた．一方，一歩踏み込んだ情報が得られないなどの，情報提供やノウハウの共有という点での課題が浮かび上がった．さらに，本フレームワークは，実用システムでの利用も始まっている．年賀状などのオンデマンド印刷を行う会社では，発注側の顧客と印刷会社のスタッフとのやりとりのためのシステム，本フレームワークを利用して構築し，運用を続けている．また，FileMaker データベースを変換して Web アプリケーションとして稼働するシステムが，ホスティングサービスの会社によって同社の顧客に提供されている．

現在の業務システム開発では，さまざまなフレームワークが用いられているが，いずれも開発専門業者によって構築が必要になり，さらに保守作業も専門業者に頼ることになる．専門家による開発ではさまざまな機能を組み込める一方で，コストは高くなる傾向にあり，予算規模が小さな場合にはシステム化をあきらめたり，あるいは保守できない状況になってしまう．その結果，中小企業や部門の中でシステム化されない業務が発生すると考えられる．調査によれば，開発と保守にかかる費用は同程度である．本フレームワークを利用することで，開発を専門会社に発注するとしても，保守作業をエンドユーザー側で行えれば，直接的なコストは半減する．本フレームワークにより，外注を中心としたシステム開発の枠組みとは異なるワークフローを作り出し，直接経費の低減やエンドユーザーが直接保守を行うことでの効率化を実現し，システム化を促進して業務の効率化に貢献できる．

謝辞

本研究を進めるにあたり，トップエスイー在学中よりご指導いただいた国立情報学研究所アーキテクチャ科学研究系の鄭顕志助教，博士課程よりご指導いただいた電気通信大学大学院情報システム学研究科の石川冬樹客員准教授に感謝します．

INTER-Mediator の開発を進めるにあたって，コミッターとして開発にご協力をいただいた，株式会社エミックの松尾篤さん，春芳堂の伊藤清徳さん，株式会社エミックの鈴木健太郎さんの貢献に感謝します．また，開発に関して協力をいただいた，飯島興産株式会社の飯島基文さん，Office sevenF の今泉みゆきさん，小林幸彦さん，CM DATABANK の溝口敬章さん，森田デザインオフィスの森田博巳さん，株式会社ビッツの村上幸雄さん，のだ歯科医院の野田修さん，国際協力データサービスの湯浅力さんにも感謝します．

Facebook グループを中心としたコミュニティの皆さんや，システム開発で採用していただいた皆さんに感謝します．研究に直結した開発機会をいただいた合同会社オンデマンドの西ヶ谷一志さんに感謝します．

インタビューに応じていただいた，倉敷芸術大学メディア映像学科の馬場始三准教授，豊国印刷株式会社の多田信之さん，国際協力データサービスの湯浅力さん，株式会社エミックの松尾篤さんと鈴木健太郎さんに感謝します．

学習可能性を探るユーザー実験にご参加いただいた皆様に感謝します．

最後に，エンドユーザー開発に関わるさまざまな知見は FileMaker を利用した開発の現場から得られました．FileMaker に関係した皆さんを始め，いっしょに FileMaker ソリューションの開発や運用を行なった皆さんにも感謝します．

参考文献

[1] Abiteboul, S., Hull, R. and Vianu, V.(eds.): *Foundations of Databases: The Logical Level*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition (1995).

[2] Adobe Systems, Inc.: Adobe Flash Professional CC, Adobe Systems, Inc. (online), available from ⟨http://www.adobe.com/jp/products/flash.html⟩ (accessed 2014-11-23).

[3] ANA SKY WEB: ANA 座席指定について，全日本空輸株式会社（オンライン），入手先 ⟨http://www.ana.co.jp/dom/reservation/asr/⟩（参照 2014-11-23）.

[4] Apple: iOS 8, Apple Inc. (online), available from ⟨http://www.apple.com/ios/⟩ (accessed 2014-11-23).

[5] Apple: OS X Yosemite, Apple Inc. (online), available from ⟨http://www.apple.com/jp/osx/⟩ (accessed 2014-11-23).

[6] Bellare, M., Canetti, R. and Krawczyk, H.: Keying Hash Functions for Message Authentication, *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, London, UK, UK, Springer-Verlag, pp. 1–15 (online), available from ⟨http://dl.acm.org/citation.cfm?id=646761.706031⟩ (1996).

[7] Benny, nemesv and answerers: Recursive template with knockout js, Stackoverflow (online), available from ⟨http://stackoverflow.com/questions/15525216/recursive-template-with-knockout-js⟩ (accessed 2014-11-23).

[8] Benson, E.: Mockup Driven Web Development, *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, Republic and Canton of Geneva, Switzerland, International World Wide Web Conferences Steering Committee, pp. 337–342 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2487788.2487939⟩ (2013).

[9] Bergkvist, A., Burnett, D. C., Jennings, C. and Narayanan, A.: WebRTC 1.0: Real-time Communication Between Browsers, W3C (online), available from ⟨http://www.w3.org/TR/webrtc/⟩ (accessed 2014-11-23).

[10] Brambilla, M., Ceri, S., Comai, S., Dario, M., Fraternali, P. and Manolescu, I.: Declarative Specification of Web Applications Exploiting Web Services and Workflows, *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, New York, NY, USA, ACM, pp. 909–910 (online), 10.1145/1007568.1007688, (2004).

[11] Brat Tech LLC, Google and community: AngularJS, Google (online), available from ⟨https://angularjs.org/⟩ (accessed 2014-11-23).

[12] Burnett, M., Cook, C. and Rothermel, G.: End-user Software Engineering, *Commun. ACM*, Vol. 47, No. 9, pp. 53–58 (online), 10.1145/1015864.1015889, (2004).

[13] Cake Software Foundation, Inc.: CakePHP, Cake Software Foundation, Inc. (online), available from ⟨http://cakephp.org/⟩ (accessed 2014-11-23).

[14] Carroll, J. B.: The Carroll Model: A 25-Year Retrospective and Prospective View, *Educational Researcher*, Vol. 18, No. 1, pp. 26–31 (online), 10.3102/0013189X018001026, (1989).

[15] Ceri, S., Daniel, F., Matera, M. and Facca, F. M.: Model-driven Development of Context-aware Web Applications, *ACM Trans. Internet Technol.*, Vol. 7, No. 1 (online), 10.1145/1189740.1189742, (2007).

[16] Ceri, S., Fraternali, P. and Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites, *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Netowrking*, Amsterdam, The Netherlands, The Netherlands, North-Holland Publishing Co., pp. 137–157 (online), available from ⟨http://dl.acm.org/citation.cfm?id=347319.346270⟩ (2000).

[17] CodeIgniter Project: CodeIgniter, British Columbia Institute of Technology (online), available from ⟨http://www.codeigniter.com/⟩ (accessed 2014-11-23).

[18] CodeZine: エミック、FileMaker Pro ベースの Web アプリを高速開発できるサービス「FMPress」のベータ版を発表，翔泳社（オンライン），入手先 ⟨http://codezine.jp/article/detail/7268⟩（参照 2014-11-23）.

[19] Coplien, J. O. and Reenskaug, T. M. H.: The Data, Context and Interaction Paradigm, *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, New York, NY, USA, ACM, pp. 227–228 (online), 10.1145/2384716.2384782, (2012).

[20] Corporation., M.: Microsoft Visual Studio LightSwitch, Microsoft Corporation. (online), available from ⟨http://www.microsoft.com/japan/visualstudio/lightswitch⟩ (accessed 2014-11-23).

[21] Corporation., M.: Office Access, Microsoft Corporation. (online), available from ⟨http://office.microsoft.com/ja-jp/access/⟩ (accessed 2014-11-23).

[22] Corporation., M.: Windows, Microsoft Corporation. (online), available from ⟨http://windows.microsoft.com/ja-jp/windows/home⟩ (accessed 2014-11-23).

[23] Crumley, M. and Graf, R.: JavaScript Expression Evaluator, （個人名義）(online), available from ⟨http://silentmatt.com/javascript-expression-evaluator/⟩ (accessed 2014-11-23).

[24] Dey, A. K., Abowd, G. D. and Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications, *Hum.-Comput. Interact.*, Vol. 16, No. 2, pp. 97–166 (online), 10.1207/S15327051HCI16234_02, (2001).

[25] DiNucci, D.: Fragmented Future, *Print*, pp. 32,221–222 (online), available from ⟨http://darcyd.com/fragmented_future.pdf⟩ (1999).

[26] Dittrich, Y., Lindeberg, O. and Lundberg, L.: End-User Development as Adaptive Maintenance, *End User Development* (Lieberman, H., PaternÃš, F. and Wulf, V., eds.), Human-Computer Interaction Series, Vol. 9, Springer Netherlands, pp. 295–313 (online), 10.1007/1-4020-5386-X_14, (2006).

[27] Dix, A., Finlay, J., Abowd, G. and Beale, R.: 7.2 Principles to Support Usability, *Human-computer Interaction*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, pp. 260–275 (1997).

[28] Ecma International: Ecma International, Ecma International (online), available from ⟨http://www.ecma-

international.org/⟩ (accessed 2014-11-23).

[29] Ecma International: TC39 - ECMAScript, Ecma International (online), available from ⟨http://www.ecmascript.org/⟩ (accessed 2014-11-23).

[30] Ecma International: Standard ECMA-262 5.1 Edition ECMAScript Language Specification, Ecma International (online), available from ⟨http://www.ecma-international.org/ecma-262/5.1/⟩ (accessed 2014-11-23).

[31] Edgar, M.: Can Google Really Access Content in JavaScript? Really?, The YouMoz Blog (online), available from ⟨http://moz.com/ugc/can-google-really-access-content-in-javascript-really⟩ (accessed 2015-1-23).

[32] Ellis, C. A. and Gibbs, S. J.: Concurrency Control in Groupware Systems, *SIGMOD Rec.*, Vol. 18, No. 2, pp. 399–407 (online), 10.1145/66926.66963, (1989).

[33] FileMaker, I.: FileMaker Server 製品概要，FileMaker, Inc.（オンライン），入手先 ⟨http://www.filemaker.co.jp/products/filemaker-server/⟩（参照 2014-11-23）.

[34] FileMaker, I.: FileMaker プラットフォーム，FileMaker, Inc.（オンライン），入手先 ⟨http://www.filemaker.co.jp/products/overview.html⟩（参照 2014-11-23）.

[35] FileMaker, Inc.: FileMaker Server 13 カスタム Web 公開 with PHP，FileMaker, Inc.（オンライン），入手先 ⟨https://fmhelp.filemaker.com/docs/13/ja/fms13_cwp_php.pdf⟩（参照 2014-11-23）.

[36] FileMaker, Inc.: FileMaker Server 13 カスタム Web 公開 with XML，FileMaker, Inc.（オンライン），入手先 ⟨https://fmhelp.filemaker.com/docs/13/ja/fms13_cwp_xml.pdf⟩（参照 2014-11-23）.

[37] Fox, V.: Google I/O: New Advances In The Searchability of JavaScript and Flash, But Is It Enough?, Search Engine Land (online), available from ⟨http://searchengineland.com/google-io-new-advances-in-the-searchability-of-javascript-and-flash-but-is-it-enough-19881⟩ (accessed 2015-1-23).

[38] Galli, M., Soares, R. and Oeschger, I.: Inner-browsing extending the browser navigation paradigm, Mozilla Developer Network (online), available from ⟨https://developer.mozilla.org/en-US/docs/Inner-browsing_extending_the_browser_navigation_paradigm⟩ (accessed 2014-11-23).

[39] Garrett, J. J.: Ajax: A New Approach to Web Applications, Adaptive Path (online), available from ⟨http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/⟩ (accessed 2014-11-23).

[40] Google: Android, Google (online), available from ⟨https://android.com/⟩ (accessed 2014-11-23).

[41] Google: Chrome, Google (online), available from ⟨https://www.google.co.jp/chrome/browser/⟩ (accessed 2014-11-23).

[42] Google: Chromebook, Google (online), available from ⟨http://www.google.com/chrome/devices/⟩ (accessed 2014-11-23).

[43] Gossman, J.: Introduction to Model/View/ViewModel pattern for building WPF apps, Microsoft Corporation. (online), available from ⟨http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx⟩ (accessed 2014-11-23).

[44] Grossman, T., Fitzmaurice, G. and Attar, R.: A Survey of Software Learnability: Metrics, Methodologies and Guidelines, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, New York, NY, USA, ACM, pp. 649–658 (online), 10.1145/1518701.1518803, (2009).

[45] Hairgami_Master, jpmorin and answerers: How can I make recursive templates in AngularJS when using

nested objects?, Stackoverflow (online), available from ⟨http://stackoverflow.com/questions/15661289/how-can-i-make-recursive-templates-in-angularjs-when-using-nested-objects⟩ (accessed 2014-11-23).

[46] Hammer, M. and Mc Leod, D.: Database Description with SDM: A Semantic Database Model, *ACM Trans. Database Syst.*, Vol. 6, No. 3, pp. 351–386 (online), 10.1145/319587.319588, (1981).

[47] Hansen, C., Adams, C., Thorsen, G. G., Nii, M. et al.: FX.php, iViking.org (online), available from ⟨http://www.iviking.org/FX.php/⟩ (accessed 2014-11-23).

[48] Hansson, D. H. and Rails core team: Ruby on Rails, Rails core team (online), available from ⟨http://rubyonrails.org/⟩ (accessed 2014-11-23).

[49] Haverbeke, M. and community: CodeMirror, Open-source project (online), available from ⟨http://codemirror.net/⟩ (accessed 2014-11-23).

[50] Heinrich, M., Grüneberger, F. J., Springer, T. and Gaedke, M.: Exploiting Annotations for the Rapid Development of Collaborative Web Applications, *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, Republic and Canton of Geneva, Switzerland, International World Wide Web Conferences Steering Committee, pp. 551–560 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2488388.2488437⟩ (2013).

[51] Hemel, Z. and Visser, E.: Declaratively Programming the Mobile Web with Mobl, *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, New York, NY, USA, ACM, pp. 695–712 (online), 10.1145/2048066.2048121, (2011).

[52] Hendriks, E., Xu, M. and Nagayama, K.: Webmaster Central Blog: Understanding web pages better, Google (online), available from ⟨http://googlewebmastercentral.blogspot.com.es/2014/05/understanding-web-pages-better.html⟩ (accessed 2015-1-23).

[53] Hevery, M. and Abrons, A.: Declarative Web-applications Without Server: Demonstration of How a Fully Functional Web-application Can Be Built in an Hour with Only HTML, CSS & Javascript Library, *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, New York, NY, USA, ACM, pp. 801–802 (online), 10.1145/1639950.1640022, (2009).

[54] 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).

[55] INTER-Mediator Contributers: INTER-Mediator GitHub Repository, INTER-Mediator Directive Committee (online), available from ⟨https://github.com/INTER-Mediator/INTER-Mediator⟩ (accessed 2015-02-05).

[56] INTER-Mediator Directive Committee: INTER-Mediator, INTER-Mediator Directive Committee (online), available from ⟨http://inter-mediator.org/⟩ (accessed 2015-02-05).

[57] JetBrains: PhpStorm, JetBrains s.r.o. (online), available from ⟨https://www.jetbrains.com/phpstorm/⟩ (accessed 2014-11-23).

[58] Jobs, S.: Thouguts on Flash, Apple Inc. (online), available from ⟨http://www.apple.com/hotnews/thoughts-on-flash/⟩ (accessed 2014-11-23).

[59] Katz, Y. and Dale, T.: Keynote with Yehuda Katz and Tom Dale, *Fluent 2014*, O'Reilly Media, Inc., (online),

available from ⟨http://fluentconf.com/fluent2014/public/schedule/detail/34996⟩ (2014).

[60] Katz, Y., Dale, T. and community: Ember.js, Tilde Inc. (online), available from ⟨http://emberjs.com/⟩ (accessed 2014-11-23).

[61] Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M. and Wiedenbeck, S.: The State of the Art in End-user Software Engineering, *ACM Comput. Surv.*, Vol. 43, No. 3, pp. 21:1–21:44 (online), 10.1145/1922649.1922658, (2011).

[62] Ko, A. J., Myers, B. A. and Aung, H. H.: Six Learning Barriers in End-User Programming Systems, *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, Washington, DC, USA, IEEE Computer Society, pp. 199–206 (online), 10.1109/VLHCC.2004.47, (2004).

[63] Laine, M., Shestakov, D., Litvinova, E. and Vuorimaa, P.: Toward Unified Web Application Development, *IT Professional*, Vol. 13, No. 5, pp. 30–36 (online), 10.1109/MITP.2011.55, (2011).

[64] Laine, M., Shestakov, D. and Vuorimaa, P.: XFormsDB: an extensible web application framework built upon declarative W3C standards, *SIGAPP Appl. Comput. Rev.*, Vol. 12, No. 3, pp. 37–50 (online), 10.1145/2387358.2387361, (2012).

[65] Lizcano, D., Alonso, F., Soriano, J. and Lopez, G.: A Web-centred Approach to End-user Software Engineering, *ACM Trans. Softw. Eng. Methodol.*, Vol. 22, No. 4, pp. 36:1–36:29 (online), 10.1145/2522920.2522929, (2013).

[66] MacLean, A., Carter, K., Lövstrand, L. and Moran, T.: User-tailorable Systems: Pressing the Issues with Buttons, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, New York, NY, USA, ACM, pp. 175–182 (online), 10.1145/97243.97271, (1990).

[67] Mac お宝鑑定団 blog（羅針盤）：エミック、FileMaker データベース連動 Web アプリケーションを超高速開発できる「FMPress」を発表，Mac お宝鑑定団（オンライン），入手先 ⟨http://www.macotakara.jp/blog/category-57/entry-20548.html⟩（参照 2014-11-23）.

[68] McAfee, A.，Brynjolfsson, E.，有賀裕子：全米上場企業四〇年間の調査が明らかにする競争力と IT 投資の知られざる力学 (Feature Articles 組織 IQ の経営)，Diamond ハーバード・ビジネス・レビュー, Vol. 33, No. 9, pp. 60–72（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/40016192299/⟩ (2008).

[69] McCallum, Q. E.: *Bad Data Handbook: Cleaning Up The Data So You Can Get Back To Work*, O'Reilly Media, 1 edition (2012).

[70] Microsoft: Internet Explorer / Web を使って見る，Microsoft（オンライン），入手先 ⟨http://windows.microsoft.com/ja-jp/internet-explorer/browser-ie⟩（参照 2014-11-23）.

[71] mozilla.org: Firefox, mozilla.org (online), available from ⟨https://www.mozilla.org/ja/firefox/desktop/⟩ (accessed 2014-11-23).

[72] National Institute of Standards and Technology: FIPS 180-1, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-1, Technical report, US Department of Commerce (1995).

[73] National Institute of Standards and Technology: FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2, Technical report, US Department of Commerce (2002).

[74] Nielsen, J.: What do users really want?, *International Journal of Human-Computer Interaction*, Vol. 1, No. 2, pp. 137–147 (online), 10.1080/10447318909525962, (1989).

[75] Nulab Inc.: Cacoo, Nulab Inc. (online), available from ⟨https://cacoo.com/lang/ja/⟩ (accessed 2014-11-23).

[76] Oracle: The Java Tutorials, Lesson: Java Applets, Oracle (online), available from ⟨http://docs.oracle.com/javase/tutorial/deployment/applet/index.html⟩ (accessed 2014-11-23).

[77] O'Reilly, T.: What Is Web 2.0 Design Patterns and Business Models for the Next Generation of Software, O ' Reilly Media, Inc. (online), available from ⟨http://oreilly.com/web2/archive/what-is-web-20.html⟩ (accessed 2014-11-23).

[78] O'Reilly Media, Inc.: Fluent 2014, O'Reilly Media, Inc. (online), available from ⟨http://fluentconf.com/fluent2014⟩ (accessed 2014-11-23).

[79] Parr, T. J.: Enforcing strict model-view separation in template engines, *Proceedings of the 13th international conference on World Wide Web*, WWW '04, New York, NY, USA, ACM, pp. 224–233 (online), 10.1145/988672.988703, (2004).

[80] Paulson, L. D.: Building Rich Web Applications with Ajax, *Computer*, Vol. 38, No. 10, pp. 14–17 (online), 10.1109/MC.2005.330, (2005).

[81] Projects, T. C.: Chromium OS, The Chromium Projects (online), available from ⟨http://www.chromium.org/chromium-os⟩ (accessed 2014-11-23).

[82] Pusher Ltd: Pusher, Pusher Ltd (online), available from ⟨http://pusher.com/⟩ (accessed 2014-11-23).

[83] Reenskaug, T. and Coplien, J. O.: The DCI Architecture: A New Vision of Object-Oriented Programming, Artima, Inc. (online), available from ⟨http://www.artima.com/articles/dci_vision.html⟩ (accessed 2014-11-23).

[84] Ricca, F., Scanniello, G., Torchiano, M., Reggio, G. and Astesiano, E.: On the Effectiveness of Screen Mockups in Requirements Engineering: Results from an Internal Replication, *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, New York, NY, USA, ACM, pp. 17:1–17:10 (online), 10.1145/1852786.1852809, (2010).

[85] Rivero, J. M., Grigera, J., Rossi, G., Robles Luna, E., Montero, F. and Gaedke, M.: Mockup-Driven Development: Providing Agile Support for Model-Driven Web Engineering, *Inf. Softw. Technol.*, Vol. 56, No. 6, pp. 670–687 (online), 10.1016/j.infsof.2014.01.011, (2014).

[86] Rode, J. and Rosson, M. B.: Programming at Runtime: Requirements and Paradigms for Nonprogrammer Web Application Development, *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*, HCC '03, Washington, DC, USA, IEEE Computer Society, pp. 23–30 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1153917.1153962⟩ (2003).

[87] Rode, J., Rosson, M. and Perez-Quinones, M.: End-Users' Mental Models of Concepts Critical to Web Application Development, *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pp. 215–222 (online), 10.1109/VLHCC.2004.25, (2004).

[88] Rode, J., Rosson, M. and QuiËIJnones, M.: End User Development of Web Applications, *End User Development* (Lieberman, H., PaternÃš, F. and Wulf, V., eds.), Human-Computer Interaction Series, Vol. 9,

Springer Netherlands, pp. 161–182 (online), 10.1007/1-4020-5386-X_8, (2006).

[89] Sanderson, S. and community: KnockoutJS, knockoutjs.com (online), available from ⟨http://knockoutjs.com/⟩ (accessed 2014-11-23).

[90] Scaffidi, C., Shaw, M. and Myers, B.: Estimating the Numbers of End Users and End User Programmers, *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '05, Washington, DC, USA, IEEE Computer Society, pp. 207–214 (online), 10.1109/VLHCC.2005.34, (2005).

[91] SeleniumHQ: Selenium, SeleniumHQ (online), available from ⟨http://www.seleniumhq.org/⟩ (accessed 2014-11-23).

[92] StatCounter: StatCounter Global Stats, StatCounter (online), available from ⟨http://gs.statcounter.com/⟩ (accessed 2014-11-23).

[93] stevekrile: View / Add / Edit / Delete Design Pattern, KnockoutJS GitHub community (online), available from ⟨https://github.com/knockout/knockout/wiki/View-Add-Edit-Delete-Design-Pattern⟩ (accessed 2014-11-23).

[94] Stiemerling, O., Kahler, H. and Wulf, V.: How to Make Software Softer&Mdash;Designing Tailorable Applications, *Proceedings of the 2Nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '97, New York, NY, USA, ACM, pp. 365–376 (online), 10.1145/263552.263646, (1997).

[95] Sun, C. and Ellis, C.: Operational Transformation in Real-time Group Editors: Issues, Algorithms, and Achievements, *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, CSCW '98, New York, NY, USA, ACM, pp. 59–68 (online), 10.1145/289444.289469, (1998).

[96] Tharp, A. L.: The Impact of Fourth Generation Programming Languages, *SIGCSE Bull.*, Vol. 16, No. 2, pp. 37–44 (online), 10.1145/989341.989351, (1984).

[97] The PHP Group: PHP Data Objects, The PHP Group (online), available from ⟨http://php.net/manual/ja/book.pdo.php⟩ (accessed 2014-11-23).

[98] TinyMCE project: TinyMCE, Moxiecode Systems AB (online), available from ⟨http://www.tinymce.com/⟩ (accessed 2014-11-23).

[99] W3C: CSS Current Status, W3C (online), available from ⟨http://www.w3.org/standards/techs/css⟩ (accessed 2014-11-23).

[100] W3C: HTML Working Group, W3C (online), available from ⟨http://www.w3.org/html/wg/⟩ (accessed 2014-11-23).

[101] W3C DOM IG: Document Object Model (DOM), W3C (online), available from ⟨http://www.w3.org/DOM/⟩ (accessed 2014-11-23).

[102] Wikipedia: Netscape Navigator, Netscape Communications (online), available from ⟨http://en.wikipedia.org/wiki/Netscape_Navigator⟩ (accessed 2014-11-23).

[103] Wilson, C.: Compatibility and IE8, Microsoft Corporation. (online), available from ⟨http://blogs.msdn.com/b/ie/archive/2008/01/21/compatibility-and-ie8.aspx⟩ (accessed 2014-11-23).

[104] Xue, Q. and community: Yii Framework, Yii Supporters (online), available from ⟨http://www.yiiframework.com/⟩ (accessed 2014-11-23).

[105] Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J. and Shanmugasundaram, J.: A Unified Platform for Data Driven Web Applications with Automatic Client-server Partitioning, *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, New York, NY, USA, ACM, pp. 341–350 (online), 10.1145/1242572.1242619, (2007).

[106] Yang, F., Shanmugasundaram, J., Riedewald, M. and Gehrke, J.: Hilda: A High-Level Language for Data-DrivenWeb Applications, *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, Washington, DC, USA, IEEE Computer Society, pp. 32– (online), 10.1109/ICDE.2006.75, (2006).

[107] 浅川直輝：特集「軽い」内製 : システム開発の常識が変わる，日経コンピュータ，No. 825, pp. 44–51（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/40019536165/⟩ (2013).

[108] アドビシステムズ株式会社：Adobe ColdFusion 11 ファミリー，アドビシステムズ株式会社（オンライン），入手先 ⟨http://www.adobe.com/jp/products/coldfusion-family.html⟩（参照 2014-11-23）.

[109] 伊藤清徳：Inter-Mediator を CodeIgniter から使ってみる，（個人名義）（オンライン），入手先 ⟨http://agilmente.com/blog/2013/07/22/inter-mediator_with_codeigniter/⟩（参照 2014-11-23）.

[110] 伊藤清徳：Yii で Inter-Mediator を使ってみる，（個人名義）（オンライン），入手先 ⟨http://agilmente.com/blog/2013/12/25/yi_inter_mediator/⟩（参照 2014-11-23）.

[111] 梅田弘之：（ITPro 情報システム）プロジェクト・マネージャの「やってはいけない」［変更管理編］変更管理を Excel でやってはいけない，日経 BP 社（オンライン），入手先 ⟨http://itpro.nikkeibp.co.jp/article/COLUMN/20110921/369014/⟩（参照 2014-11-23）.

[112] 株式会社エミック：FileMaker WebDirect と FMPress Publisher の比較，株式会社エミック（オンライン），入手先 ⟨https://www.emic.co.jp/fmpress/publisher/comparison-with-webdirect/⟩（参照 2014-11-23）.

[113] 株式会社エミック：FileMaker データベース連動 Web アプリケーションを超高速開発できる FMPress のベータバージョンを発表，株式会社エミック（オンライン），入手先 ⟨https://www.emic.co.jp/press/20130711_fm/⟩（参照 2014-11-23）.

[114] 株式会社エミック:FMPress, 株式会社エミック (online), available from ⟨https://www.emic.co.jp/fmpress/⟩ (accessed 2014-11-23).

[115] 株式会社エミック：FMPress Publisher, 株式会社エミック (online), available from ⟨https://www.emic.co.jp/fmpress/publisher/⟩ (accessed 2014-11-23).

[116] 株式会社エミック：FMPress Publisher と INTER-Mediator，株式会社エミック（オンライン），入手先 ⟨https://www.emic.co.jp/fmpress/publisher/inter-mediator/⟩（参照 2014-11-23）.

[117] 奥村晴彦：「ネ申 Excel」問題，情報教育シンポジウム 2013 論文集，Vol. 2013, No. 2, pp. 93–98 (2013).

[118] エリックガンマ，ラルフジョンソン，リチャードヘルム，ジョンブリシディース：第 5 章振る舞いに関するパターン, Observer パターン，オブジェクト指向における再利用のためのデザインパターン，ソフトバンククリエイティブ，改訂 edition，pp. 313–324 (1999).

[119] グレープシティ株式会社：Forguncy, グレープシティ株式会社 (online), available from ⟨http://www.forguncy.com/⟩ (accessed 2014-11-23).

[120] 経済産業省 商務情報政策局情報経済課：情報処理実態調査，経済産業省（オンライン），入手先 ⟨http://www.meti.go.jp/statistics/zyo/zyouhou/index.html⟩（参照 2014-11-23）.

[121] 経済産業省中小企業庁事業環境部企画課調査室：中小企業・小規模事業者の数（2012 年 2 月時点）の集計結果を公表します，経済産業省中小企業庁（オンライン），入手先 ⟨http://www.meti.go.jp/press/2013/12/20131226006/20131226006.html⟩（参照 2014-11-23）.

[122] 経済産業省 ソフトウェアメトリクス高度化プロジェクト プロダクト品質メトリクス WG：システム/ソフトウェア製品の品質要求定義と品質評価のためのメトリクスに関する調査報告書 (2011).

[123] 斎藤昌義：システムインテグレーション崩壊〜これから SIer はどう生き残ればいいか?，技術評論社 (2014).

[124] サイボウズ株式会社：kintone，サイボウズ株式会社 (online)，available from ⟨https://kintone.cybozu.com/jp/⟩ (accessed 2014-11-23).

[125] 株式会社ジャストシステム：ノンプログラミング Web データベースソフト UnitBase，株式会社ジャストシステム（オンライン），入手先 ⟨http://www.justsystems.com/jp/products/unitbase/⟩（参照 2014-11-23）.

[126] 独立行政法人情報処理推進機構：中小企業等の IT 活用に関する実態調査（平成 24 年 9 月），独立行政法人情報処理推進機構（オンライン），入手先 ⟨http://www.ipa.go.jp/files/000023618.pdf⟩（参照 2014-11-23）.

[127] 株式会社セールスフォース・ドットコム：Salesforce, 株式会社セールスフォース・ドットコム (online), available from ⟨http://www.salesforce.com/jp/⟩ (accessed 2014-11-23).

[128] 株式会社セカンドファクトリー：事例紹介 JAL 国内線予約 Flash 版，株式会社セカンドファクトリー（オンライン），入手先 ⟨http://www.2ndfactory.com/showcase/jal.html⟩（参照 2014-11-23）.

[129] 武内利浩：IT 人財育成と IT コスト最適化〜アウトソースを活用した内製化アプローチ，アクセンチュア株式会社（オンライン），入手先 ⟨http://www.accenture.com/SiteCollectionDocuments/jp-ja/PDF/industry/finance/Accenture-fsa26-4.pdf⟩（参照 2014-11-23）.

[130] 田中克己：なぜ遅くて高い？企業システム開発の「不都合な真実」，日経 BizGate（オンライン），入手先 ⟨http://bizgate.nikkei.co.jp/article/73023716.html⟩（参照 2014-11-23）.

[131] 超高速開発コミュニティ：Extremely Rapid Application Development Community (xRAD), 高速開発コミュニティ (online)，available from ⟨https://www.x-rad.jp/⟩ (accessed 2014-11-23).

[132] 中嶋一樹：AngularJS ではじめる HTML5 開発—Part8 モーダルダイアログによる新規レコード作成フォーム,（個人名義）（オンライン），入手先 ⟨http://www.nkjmkzk.net/?p=5401⟩（参照 2014-11-23）.

[133] 財団法人名古屋市工業技術振興協会：中小企業向け情報システムの現状調査（平成 20 年 3 月），財団法人名古屋市工業技術振興協会（オンライン），入手先 ⟨http://nabinabi.biz/wp-content/uploads/hp_cyusyo_an.pdf⟩（参照 2014-11-23）.

[134] ヤコブニールセン：第 2 章 ユーザビリティとは?，ユーザビリティエンジニアリング原論- ユーザーのためのインタフェースデザイン (情報デザインシリーズ)，東京電機大学出版局，第 2 edition，pp. 19–38 (2002).

[135] 新居雅行，新妻利恵：FileMaker as a Relational Database リレーショナルデータベースを徹底活用，自費出版 (http://msyk.net/fmp/relbook3/) (2013).

[136] 新居雅行，鄭　顕志，石川冬樹：エンドユーザーによる保守作業を可能にする Web アプリケーションフレームワーク，コンピュータソフトウェア， Vol. 31, No. 1, pp. 60–74（オンライン），10.11309/jssst.31.1_60, (2014).

[137] 西岡靖之：コンテキスト編集をもちいたプログラムレスの業務アプリケーション開発，第 75 回全国大会講演論文集，Vol. 2013, No. 1, pp. 439–441 (2013).

[138] 日経 BP コンサルティング：中堅中小企業の IT 利活用調査（2012 年 1 月 23 日），日経 BP コンサルティング（オンライン），入手先 ⟨http://consult.nikkeibp.co.jp/consult/news/2012/0120it/⟩（参照 2014-11-23）.

[139] 日経 BP システム運用ナレッジ：企業情報システムの運用管理に関する実態調査 2013 現場が疲弊する理由 IT 関連コストの 75 %は稼働後に発生する，日経 BP 社（オンライン），入手先 ⟨http://itpro.nikkeibp.co.jp/article/COLUMN/20130702/488891/⟩（参照 2014-11-23）.

[140] 日経コンピュータ（編）：開発・改良の切り札 システム内製を極める，日経 BP 社 (2011).

[141] 社団法人日本情報システム・ユーザー協会：ソフトウェアメトリックス調査 2011（保守調査報告）(2011).

[142] 日本ユーザーメード医療 IT 研究会：J-SUMMITS, 日本ユーザーメード医療 IT 研究会 (online), available from ⟨http://j-summits.jp/⟩ (accessed 2014-11-23).

[143] フランクブッシュマン，ハンスローネルト，ミカエルスタル，レギーネムニエ，ペーターゾンメルラード：2.4 対話型システム Model-View-Controller（モデル-ビュー-コントローラ），ソフトウェアアーキテクチャ- ソフトウェア開発のためのパターン体系，近代科学社，pp. 120–139 (2000).

[144] フランクブッシュマン，ハンスローネルト，ミカエルスタル，レギーネムニエ，ペーターゾンメルラード：3.4 アクセス制御 Proxy（プロキシ），ソフトウェアアーキテクチャ - ソフトウェア開発のためのパターン体系，近代科学社，pp. 257–271 (2000).

[145] マイナビニュース：FileMaker データベース連動 Web アプリを高速開発できる「FMPress」ベータ版，株式会社マイナビ（オンライン），入手先 ⟨http://news.mynavi.jp/news/2013/07/12/266/index.html⟩（参照 2014-11-23）.

[146] 松尾　篤：IMCake,（開発者名義）(online), available from ⟨https://github.com/matsuo/IMCake⟩ (accessed 2014-11-23).

[147] 満田成紀，福安直樹：ウェブアプリケーションフレームワーク利用に潜む課題〜実プロジェクトデータを題材に〜，コンピュータソフトウェア， Vol. 27, No. 3, pp. 2–12（オンライン），10.11309/jssst.27.3_2, (2010).

[148] 谷島宣之：ソフトを他人に作らせる日本、自分で作る米国，日経 BP 社 (2013).

研究業績

[1] INTER-Mediator, http://inter-mediator.com/

[2] INTER-Mediator, GitHub Repository, https://github.com/INTER-Mediator/INTER-Mediator

[3] 新居 雅行, 鄭 顕志, 石川 冬樹; エンドユーザーによる保守作業を可能にする Web アプリケーションフレームワーク, コンピュータ　ソフトウェア 31(1), pp.60-74, 2014, 日本ソフトウェア科学会 [査読付き]

[4] Masayuki Nii, Kenji Tei, and Fuyuki Ishikawa; Framework Enabling End-Users to Maintain Web Applications, International MultiConference of Engineers and Computer Scientists 2015, ICICWS 2015, March 18 - 20, 2015, Hong Kong [査読付き]

[5] INTER-Mediator 基礎，INTER-Mediator 応用（オンライン学習コース），自費出版，2012 年 http://edu.msyk.net/

[6] 新居雅行; 低コストな開発・保守の実現を目指す Web アプリケーションフレームワーク, 電気通信大学産学官連携 DAY 博士後期課程学生研究発表会（2013/6/5）

[7] 新居雅行; 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，情報処理学会第 153 回データベースシステム研究発表会（2011/11/3）

[8] 新居雅行; Web アプリケーションでのページ遷移を伴わない認証認可処理の実装とセキュリティ，情報処理学会第 155 回データベースシステム研究発表会（2012/11/19）

[9] Masayuki Nii, INTER-Mediator: Simple and Extensible Web Application Framework, FileMaker Developer Conference 2012: CWP User Group Meeting

著者略歴

新居 雅行（にい まさゆき）: フリーランスのエンジニアとして，システム開発やコンサルティングに携わる．トレーナーやライターとしても活動する．1987 年，京都工芸繊維大学大学院修了．2012 年，国立情報学研究所トップエスイー修了．2015 年，電気通信大学大学院後期博士課程修了．1987〜1989 年，日経マグロウヒル社（現，日経 BP 社）で日経パソコン編集部記者．1989〜2002 年，株式会社ローカスでシニアアナリスト，後に情報メディア出版局編集長を勤める．2005〜2008 年，アップルコンピュータ株式会社（現，アップルジャパン株式会社）で認定トレーニング担当メンタートレーナー．1997 年より慶応義塾大学文学部非常勤講師．2015 年より国立情報学研究所特任研究員．主要な著書は「Macintosh アプリケーションプログラミング」「リレーションで極める FileMaker」「OS X システム管理」など．