［論文］

RIA におけるデザイナと開発者の完全分業を実現するフレームワーク

福田浩章†，山本喜一‡

要 旨

現在の情報化社会において，情報の発信，検索，オンライン取引など，インターネットを利用したウェブアプリケーションは必要不可欠であり，我々は目的に応じて適切なウェブアプリケーションを利用している。それらのウェブアプリケーション開発では，一般にデザイナが HTML や CSS，画像などを利用して画面を設計し，開発者がビジネスロジックを実装する。また，ソフトウェア開発者は実行するロジックの結果によって動的に HTML タグを構築するため，デザイナが作成する画面に手を加える必要がある。したがって，画面の変更が発生すると，デザイナだけでなく開発者の手も煩わすことになる。

そこで本論文では，デザイナと開発者の作業をソースレベルで完全に分離する Flex ベースのフレームワーク，Camel の提案および実装を行う。そして，実際に Camel を用いたアプリケーションを構築することによって有効性を示す。Camel によって，画面に変更が生じた場合でも開発者の手を煩わすことなく，変更前の動作を保障するアプリケーション開発が可能になる。

Abstract

In current information society, web applications using the Internet such as information transmittance, search and online trading have become vital foundation of our life. We usually use appropriate web applications to achieve our goal. In the development process of current web applications, designers design interfaces of an application by using HTML, CSS, image and developers not only implement business logics but also modify the interface for dynamic generation of HTML tags based on the executed logic. Therefore, not only designers but also developers have to work every time when the page design is changed.

This paper provides a Flex based framework called Camel which is able to separate developer's and designer's work completely. We also show the utility of this framework by implementing and executing an application. By using Camel framework, developers do not have to work when the page design of an application is changed and the behavior of the application is guaranteed.

A framework for realizing complete separation of developer's and designer's work in RIA

Hiroaki Fukuda†, Yoshikazu Yamamoto†
†慶應義塾大学大学院理工学研究科

Graduate School of Science and Technology, Keio University

［論文］2008 年 11 月 10 日受付

1 はじめに

インターネットの普及により，情報の発信，検索，オンライン取引など，ウェブアプリケーションの使用は一般化し，それを利用した多くのサービスがインターネット上に存在する。ウェブアプリケーションは一般に HTTP と HTML で構築されるため，デスクトップアプリケーションと比較すると表現力に乏しく，同期通信のオーバーヘッドによる遅延などにより，操作性がよいとは言えない[1]。これらの問題点

を解決するため，近年では Ajax(Asynchronous JavaScript and XML) を代表とした，いわゆる RIA(Rich Internet Application) が提案され，画面のデザインやユーザの操作性を向上する努力がなされている[2]。

そのため，画面を設計するデザイナと，ビジネスロジックを記述する開発者の協業が必要不可欠である。ウェブアプリケーション開発では，一般にデザイナが HTML や CSS，画像を使用して画面を作成した後，開発者がビジネスロジックの実装を行う。また，ロジックの実行結果に応じて動的に HTML タグを生成するため，開発者はデザイナが作成した HTML ベースの画面に手を加える必要がある。したがって，システム開発中，または開発後に画面の変更要求があると，デザイナの作業だけではなく，ロジックの記述，動作確認といった開発者の作業も必要になり，作業量が増加する。これはデザイナと開発者で画面部分の HTML を共有せざるを得ないという制約があるためである。

そこで本論文では，RIA 開発ツールである Flex[3] を利用し，デザイナと開発者の作業をソースレベルで完全に分離するためのフレームワークである Camel の提案と実装を行う。Flex では，画面を設計するための豊富なコンポーネントがあらかじめ用意されており，コンポーネントで発生するイベントを契機に処理が行われる。したがって，通常各コンポーネントで発生するイベントと，それを受け取るハンドラを画面のデザイン部分に直接記述する必要がある。そのため，Flex 標準の仕組みだけでは画面とロジックをソースレベルで分離することはできない。Camel では Flex のイベント処理に着目し，別々のソースファイルに記述されたコンポーネントとハンドラの依存関係を実行時に確立する。そのため，コンポーネントのイベントとハンドラの関係をソースファイルに直接記述する必要がない。その結果，ロジックの影響を考慮することなくデザインを変更することが可能であり，変更後の動作も保障できるため，作業効率の向上が期待できる。

以下 2 節では，現在のウェブアプリケーション開発工程を述べた後問題点を指摘し，RIA アプリケーションの概要を述べる。3 節では，本論文のアプローチと関連技術について述べ，4 節で具体的な実装について述べる。5 節では実用的なアプリケーション開発に Camel を適用して評価する。最後に 6 節で関連研究について述べ，7 節でまとめと今後の課題を述べる。

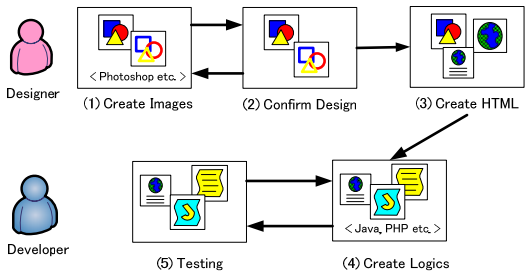


図 1　ウェブアプリケーションの一般的な開発工程

2 背景

本節では本研究の背景と関連技術について述べる。まず，現在のウェブアプリケーション開発工程を述べた後，問題点を指摘する。そして，RIA アプリケーションを構築する技術について述べ，最後に Flex フレームワークについて概説する。

2.1 アプリケーション開発プロセス

1 節で述べたように，ウェブアプリケーションは個々のユーザが情報の発信や検索を行うためだけではなく，企業にとっても自身の広告や新規ユーザの獲得など，ビジネスの運営に必要不可欠なツールとなっている。そのため，アプリケーションは機能だけではなく，配色や画像，Flash コンテンツ，動画といったコンポーネントの配置など，ユーザに強い印象を与える画面のデザインが重要になってきている。したがって，一般ユーザを対象にしたウェブアプリケーション開発では，ビジネスロジックを実装する開発者と，画面をデザインするデザイナの協業が必要となる。図 1 にウェブアプリケーションの一般的な開発工程を示し，番号で示す手順を述べる。

（1）デザイナは Photoshop などのツールを用いてウェブサイト全体を画像として作成する。

（2）アプリケーションの発注者がデザインを確認し，変更点があればデザイナに修正要求

を出す。

（３）一度デザインが完成すると，デザイナ(または HTML コーダ) は画像として作成されたウェブサイトからボタンやバナーなどの画像オブジェクトを切り出す。そしてそれらのオブジェクトと HTML, CSS を利用し，静的な HTML ページとしてウェブサイトを完成する。

（４）開発者がビジネスロジックを実装する。この時，データベースアクセスなどサーバ側のロジックに加え，動的な HTML タグの生成や，クライアント側での入力データ確認を行うため，デザイナが作成した HTML ページに手を加える必要がある。

（５）開発者が単体または結合テストを行う。エラーやバグが発生した場合，手順(4) に戻り修正する。

通常は，手順(4) と(5) を繰り返し，運用に入る。現在のウェブアプリケーション開発では主に MVC (Model, View, Controller) アーキテクチャが採用されているため，サーバ側のビジネスロジックは画面設計と分離される。しかし，ロジックの実行結果を反映したモデルの状態を画面に反映させるため，デザイナが作成した HTML の画面に手を加え，動的に HTML タグを生成する必要がある。図 2 にデザイナが作成する HTML の具体例を示し，その HTML をもとに開発者が PHP でロジックを記述したソースコードを図 3 に示す。図 3 の下線部は開発者によって記述される部分であり，繰返し処理や動的に生成する HTML の表示を行っている。例えば図 2(ｉ)は図 3(ｉｉ)に対応しているが，図 3(ii)では表示するデータによって結合するセルの数を動的に切り替えるため，プログラムによって“colspan”属性の値を切りかえている。仮に図 3 のプログラムが完成した後，図 2(ｉ) 部分に画像を挿入するような変更が生じると，図 3(ｉｉ)で出力する文字列も変更する必要がある。すなわち，デザインの変更によるプログラムの修正が発生する。このように，デザインに変更が発生すると，開発者はもとの HTML に合わせるため，繰返し処理の中に単純な文字列や数値データだけでなく，HTML タグやオブジェクトの位置を微調整するための画像データ

```
<img src="image/spacer.gif" width="1" height="20"><br>
<table width="100%"  border="0" cellspacing="0" cellpadding="0">
 <tr>
  <td bgcolor="#999999">
   <table width="100%"  border="0" cellspacing="1" cellpadding="5">
    <tr bgcolor="#AAE3FF">
     <td height="25" colspan="2">
      <span class="L">CMS Development</span>
     </td>
    </tr>
    <tr bgcolor="#FFFFFF">
     <td colspan="2">
      Environment HTML, Javascript, VB<br>          (i)
     </td>
    </tr>
    <tr bgcolor="#FFFFFF">
     <td width="22%" bgcolor="#E6E6E6">SKILL</td>
     <td width="80%"><br>
       ・HTML、JavaScript、VB<br>
    </tr>
   </table></td>
  </tr>
</table>
```

図 2　HTML で作成されたデザイン

```
<?php
 $jobListArray = $model->getJobListArray();
 $salaryTypeArray = $model->getSalaryTypeArray();
?>
<?php  for($i = 0; $i < count($jobListArray); $i++) { ?>
 <img src="../image/spacer.gif" width="1" height="20"><br>
 <table width="100%"  border="0" cellspacing="0" cellpadding="0">
  <tr>
   <td bgcolor="#999999">
    <table width="100%"  border="0" cellspacing="1" cellpadding="5">
     <tr bgcolor="#AAE3FF">
      <td height="25" colspan="2">
       <span class="L"><?= $jobListArray[$i]->getTitle() ?> </span>
      </td>
     </tr>
     <tr bgcolor="#FFFFFF">
      <?= $jobListArray[$i]->getDesc()  ?>          (ii)
     </tr>
     <tr bgcolor="#FFFFFF">
      <td width="22%" bgcolor="#E6E6E6">SKILL</td>
      <td width="80%">
       <?= $jobListArray[$i]->getSkill()  ?>
      </td>
     </tr>
    </table></td>
   </tr>
 </table>
<?php } ?>
```

図 3 PHP のソースコード

を含める必要がある。図 2 のように，一般にデザイナが作成する HTML は複雑になるため，デザインを崩さないようにロジックを埋め込む作業は開発者にとって負担が大きい。特に，動的に HTML を出力している場合，デザイン変更に伴い出力するタグも変更されることが多く，すべての作業をやり直すことになる。この修正作業には時間も費用も費やすことになるが，初めからアプリケーション全体を過不足なく設計することは困難なため，実際の開発ではデザインを含めた仕様変更は頻繁に発生する。

最近のウェブアプリケーション開発では，

PHP やJSP 等のプログラムを直接HTML に埋め込むのではなく，タグライブラリやテンプレートエンジンを利用している。しかし，前述したように，それらを利用しても画面のデザインとロジックを完全に分離することはできないため，デザイナと開発者で画面のソースを共有せざるを得ない。これは，本来文書の論理構造を記述するために作成された HTML がインターネットとともに普及し，ブラウザだけで利用できるウェブアプリケーションに発展したことが一因である。

2.2 RIA 技術の概要

“Rich Internet Application”はデスクトップアプリケーションと，ウェブアプリケーション双方の欠点を補完し統合を図るため，Macromedia 社によって提案された[4]。HTTP と HTML を使用した従来型ウェブアプリケーションと比較すると，RIA は操作性に優れたインタフェースを提供する。また，RIA 自体はクライアントのリクエストを契機に初期データと共にロードされ，以後のデータは必要に応じて非同期にロードされるため，通信のオーバーヘッドを削減できるという効果がある。

RIA 開発を支援するため，さまざまな技術が提案され，利用されている。これらの技術は次の 4 つに分類することができる。

- スクリプトベース

クライアント側のロジックを Javascript などのスクリプト言語で実装し，画面を HTML と CSS で作成する方式。Ajax はこの分類で最も有名な技術である。

- プラグインベース

ブラウザにプラグインとしてインストールされたエンジンを利用し，操作性に優れたアプリケーションを実現する方式。Flash[5] がもっとも有名な技術であり，Flex, Laszlo[6] などがこの分類に属する。

- ブラウザベース

操作性に優れたコンポーネントをあらかじめブラウザに組み込むことによって RIA を実現する方式。個々のコンポーネントは XML をベースとしたマークアップ言語で記述することができる。

- ウェブを介したデスクトップアプリケーション

アプリケーションはウェブブラウザを介してダウンロードされ，ブラウザに依存せず実行される方式。Java Web Start[7] や Smart Client[8] がこの分類に属する。

1 節で述べたように，Camel は Flex フレームワークを利用しているため，Camel を利用して作成されたアプリケーションはブラウザにインストールされた Flash プラグイン上で動作する。

2.3 Flex フレームワーク

Flex は Adobe Systems Inc。で開発された RIA 開発環境である。Flex で作成されたアプリケーションは Flash プレイヤー上で動作するため，Flash プラグインがブラウザにインストールされていれば OS に依存せず動作する。また，Flex では Macromedia Flex Markup Language (MXML) と ActionScript 3。0 (AS3) という 2 つの言語を使用してアプリケーションを開発する。MXML は XML ベースの言語であり，TextInput，DataGrid，Button といったコンポーネントが MXML タグとして用意されているため，デザイナは主に MXML を利用して画面を設計する。一方，AS3 は開発者が Flash プレイヤー上で実行するクライアント側のロジックを記述するために使用される。また，Flex では HttpRequest，WebService，Flash Remoting といった RPC コンポーネントを備えており，必要に応じてサーバと連携して処理を進める。

このように，Flex アプリケーション開発では，デザイナが MXML，CSS，画像を利用して画面を作成し，開発者が AS3 でクライアント側のロジック，Java や PHP といったサーバ側の技術でサーバ側のロジックを実装していく。この開発プロセスはデザイナが HTML や CSS で画面を作成し，開発者が JavaScript などの技術を利用してウェブアプリケーションを構築していく開発プロセスと酷似しているため，デザイナや開発者はこれまでの開発で得た経験や知識を再利用することができる。

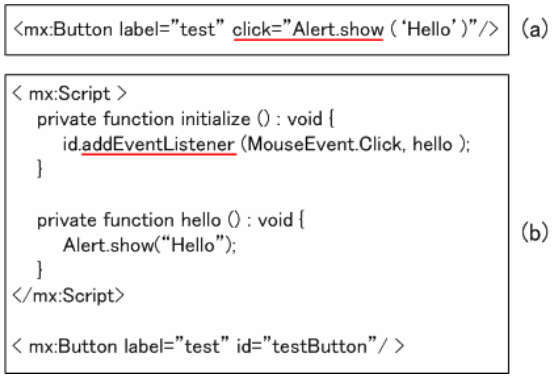


図 4　Flex プログラミング

3 アプローチ

本節では，本研究で注目した Flex の動作方式と，コンポーネント間の結合，そして依存性の確立について述べる。

3.1 Flex の動作方式

Flex アプリケーションはイベントドリブンで動作する。ユーザがコンポーネントを操作する，あるいは時間の経過やサイズの変更，オブジェクトの生成や削除を契機にイベントが発生し，イベントハンドラがそれぞれのイベントを処理する。これらのイベントハンドラは，AS3 を用いてメソッドとして定義される。また Flex では，イベントとイベントハンドラを関連付けるために，2 種類の方法が用意されている。

まず図 4(a) で示すように，すべての MXML タグはそれ自身が発行するイベントを属性名として保持し，イベントハンドラを値として記述することで関連付けを行う。

図 4(a) の例では，“Button” コンポーネントに “click” イベントが発生したとき，“Alert。show(‘Hello’)” が実行されることになる。次に図 4(b) で示すように，イベントを発行するコンポーネントの “addEventListener” メソッドを利用してイベントとイベントハンドラを関連付けを行う。“addEventListener” メソッドはイベントを発行するコンポーネントが必ず保持するメソッドであり，これを利用するには第一引数にイベント名，第二引数にイベントハンドラを指定する。図 4(b) の例では，“testButton” という ID が割り当てられた Button コンポーネントの “MouseEvent。Click” が発生したとき，“hello” メソッドが実行されることになる。なお，AS3 はオブジェクト指向言語であるため，イベントハンドラだけのクラスを MXML とは異なるファイルで作成し，図 4(b) のように関連付けることも可能である。そして，これらの処理は Flash プレイヤー上で行われるため，ネットワークの状態によらず動作する。

一方，データの送受信をサーバと行う場合，Flex では 2.3 節で述べた RPC コンポーネントを利用する。表示用コンポーネントと同様に，RPC コンポーネントもイベントドリブンで動作し，RPC の呼出しが完了すると “result” または “fault” イベントが発行される。そのため，開発者は呼出しを行う前にそれらイベントのイベントハンドラを関連付ける必要がある。

3.2 コンポーネント間の結合

2.1 節で述べたように，現行のウェブアプリケーション開発では拡張性や柔軟性を考慮し，MVC アーキテクチャを採用している。そのため，Flex アプリケーションでも MVC アーキテクチャの採用が望ましい。ウェブアプリケーションで MVC アーキテクチャを導入する場合，View である HTML は Model の状態によって変化する。つまり，View と Model は密結合である。一方，View は Controller へ HTTP リクエストを送信し，Controller は受け取ったリクエストに対して適切なビジネスロジックを実行して Model の状態を変更する。View と Controller は HTTP を介してデータをやり取りするため疎結合である。現行のウェブアプリケーションは View と Model が密結合であり，HTML ベースである以上，両者をソースファイルレベルで完全に分離することはできない。

一方 3.1 節で述べたように，Flex アプリケーションでは View である MXML に Controller であるイベントハンドラは一対一で対応するため密結合となる。また，View の各コンポーネントと Model の関連付けも一対一で対応するため密結合となる。Camel では，View，Model，Controller をそれぞれ独立して記述し，アプリケーション実行時にそれらの依

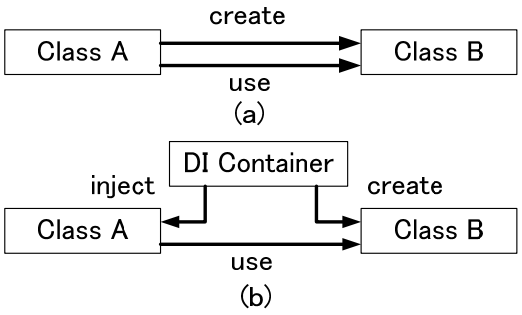


図 5　依存性の確立

存性を確立して密結合にする。この結果 View を作成するデザイナと，Model, や Controller を作成する開発者の役割をソースレベルで完全に分離することが可能になる。

3.3 依存性の確立

依存性の確立とは“DI する”とも言われ，近年導入されている軽量コンテナを実現するための新しい設計思想, Dependency Injection のことである。

図 5(a) では，クラス A はクラス B のオブジェクトを生成するコードと，生成したオブジェクトの機能を使用するコードを含んでいる。そのため，クラス A はクラス B が存在しなければ実行することができない。言い換えると，クラス A はクラス B に依存している。一方図 5(b) では，クラス A がクラス B のオブジェクトを生成するコードを直接クラス A の中に含めるのではなく，間接的な方法(メタレベル) で記述し，DI コンテナと呼ぶフレームワークに生成処理を委譲している。そのため，クラス A とクラス B がインタフェースレベルでの依存関係を持つ場合，実装クラスの生成を DI コンテナに委譲することで，両者を疎結合にすることができる。その結果，クラス A はそれ自身とクラス B が実装するインタフェースだけで実行やテストが可能になる。

Camel ではこの設計思想を応用し，図 4 で示した属性または “addEventListener” を用いたコンポーネントとイベントハンドラの関連付けを排除するとともに, Model 内変数と値との連結を行う。

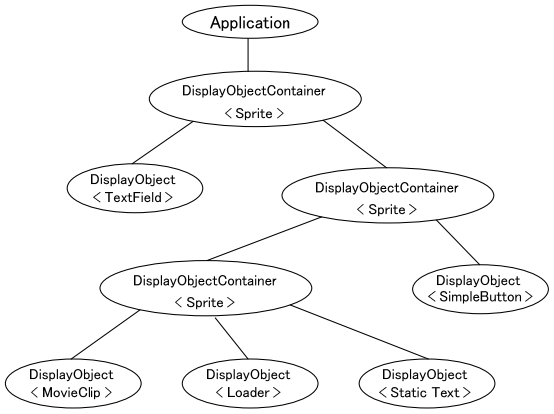


図 6　Display Object の木構造

4 設計と実装

Camel は Flash プレイヤーが内部で保持する Display Object の木構造と，AS3 のリフレクション機構を利用している。本節ではこれらについて述べた後，Camel のアーキテクチャを示す。そして，Camel を利用する場合の基本ルールについて述べる。

4.1 Display Object Tree

図 6 に示すように，Flex では画面を構成するコンポーネントを“Application”クラスのオブジェクトをルートとした“ディスプレイリスト”と呼ばれる木に追加することで画面を表示している[9]。そして，これら画面用のコンポーネントは“DisplayObject”クラスを継承した“UIComponent”クラスを基底クラスとしている。また，非表示のクラスであっても，ディスプレイリストに追加することができる。したがって，このディスプレイリストを操作することによってオブジェクトの参照を実行時にすべて取得することができる。

また，ディスプレイリストにオブジェクトを追加するには，追加するオブジェクトを引数に指定して親となるオブジェクトの“addChild”メソッドを呼び出すか，親となる MXML にタグとして記述すればよい。なお，非表示のクラスでも MXML タグとして記述することができ，追加方式の違いによる差異はない。

4.2 リフレクション機構

AS3 は Java や Ruby と同様，強力なリフレクション機構を備えている。AS3 のリフレクション機構では, すべてのオブジェクトを XML

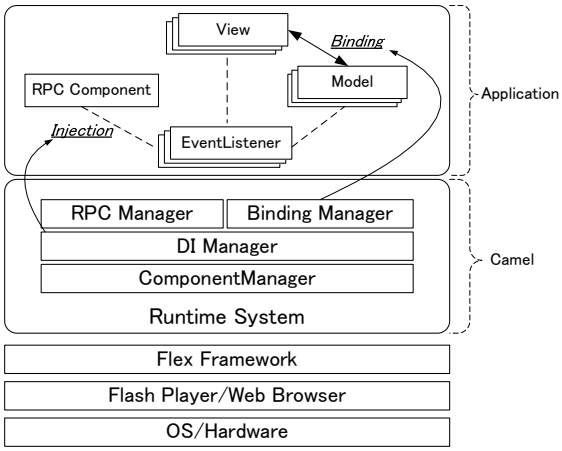


図 7　Camel のアーキテクチャ

形式に直列化することができる。そして，その中にプロパティやメソッド名，親クラスなどのメタ情報がすべて含まれている。また，XML の各要素は EX4 形式[10] で参照することができる。そのため，オブジェクトの参照が取得できれば，メソッド名や引数の情報だけで容易にメソッドを呼び出すことができる。

4.3 Camel のアーキテクチャ

図 7 に Camel のアーキテクチャを示す。Camel 自身は Flex フレームワークを利用しているため，Flex のライブラリという位置付けである。4.4 節で述べるように，Camel を利用する場合デザイナが作成する View に対し，開発者は内部コンポーネントのイベントを処理する“Listener”クラスを作成する必要がある。

Camel は主に次に示す 4 つのコンポーネントで構成されている。

（1）Component Manager

Component Manager は View や Model コンポーネントの参照を保持する。また，View や Listener，あるいはディスプレイリストでそれらの親に相当するコンポーネントの階層構造を管理する。

（2）RPCManager

RPCManager は Flex で用意された HTTPService, WebService, RemoteObject, Producer, Consumer という RPC 用のコンポーネント管理する。開発者はアプリケーションで使用する RPC 用コンポーネントを RPCManager に登録しておく必要がある。具体的な登録方法は 4.4 節で述べる。

（3）DI Manager

DI Manager はアプリケーション実行時にコンポーネント間で 3 種類の依存性を確立する。1 つ目は，View 内部のコンポーネントで発生するイベントと，対応する Listener が保持するイベントハンドラの関連付けである。2 つ目は Listener クラス内に定義された View と Model の参照変数と，実体オブジェクトの関連付けである。3 つ目は，RPC 用コンポーネントを利用する場合，Listener クラス内に定義された RPC コンポーネントの参照変数と，RPCManager が管理する RPC コンポーネントの関連付け，およびコールバック関数の関連付けである。具体的な記述法および関連付けについては 4.4 節で述べる。

（4）Binding Manager

Binding Manager は View の内部コンポーネントが保持する値と，Model のメンバとして定義された変数の値を連結する。Binding Manager は，View 内のコンポーネント ID と，Model 内の変数名が同一である場合，両者の値を連結する。図 8 に示す View と Model の関係では，Model 内の “dataGrid” は，View 内の “dataGrid” という ID が付加されたコンポーネントと連結される。また，String や Number といった変数の型は，コンポーネントの種類ごとにあらかじめ Camel で定義している(TextInput では String, DataGrid では ArrayCollection など)。

4.4 基本ルール

Camel では，Ruby on Rails[11] で導入された Convention over Configuration(設定より規約) の思想を受け，XML ベースの設定ファイルを用いるのではなく，名前をベースにした規

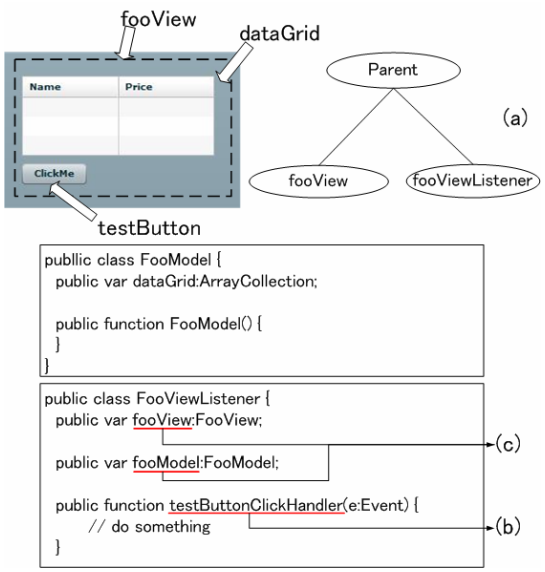


図 8　Model, View, Listener の関係

```
<ServiceRepository>
  <mx:WebService id="myserv" url="http://aaa.bbb/"/>
</ServiceRepository>
```

```
public class BarViewListener {
  public var myserv:WebService

   public function myserv_doLoginResultServiceHandler {
      // do success process
  }
  public function myserv_doLoginFaultServiceHandler {
      // do fault process
  }
}
```

図 9　RPC コンポーネントの使用

約を導入することによって依存性の確立を行う。

Camel を利用した開発では, 開発者は Model, View, Listener を 1 つの組として作成する必要がある。図 8 に Model, View, Listener の関係，図 9 に RPC コンポーネントの使用例を示し，Camel で導入する規約を次に述べる。

（1）すべてのコンポーネントは MXML タグで定義され，固有の ID を持つ。

（2）図 8 (a)に示すように，View，および対応する Listener は，ディスプレイリストの同一階層に配置され，同一の親を持つ。

（3）View の ID は“View”という文字列で終わる必要がある。また，図 8(a) のように，対応する Listener は View の ID に“Listener”という文字列を付加する(例えば，“fooView”に対応する Listener の ID は“fooViewListener”とする)。

（4）Listener のイベントハンドラとなるメソッド名は，“コンポーネント ID”+“イベント名”＋Handler とし，イベント名の先頭は大文字とする。例えば図 8(b) では，“testButton”という ID が付加された Button の click イベントを処理するイベントハンドラを，“testButtonClickHandler”と定義している。

（5）Listener は対応する View と Model の参照変数を内部に保持する。さらに, View を参照する変数名は(3)で示した View の ID (fooView) と同一にする必要がある。また，図 8(c) で示すように，Model を参照する変数名 (fooModel) は，View を参照する変数名 (fooView) の接尾語である“View”を“Model”に置き換えて定義する。

（6）Model は View の内部コンポーネントと連結する変数を保持し，それらの変数名は対応するコンポーネントの ID と同一にする。また，4.3 節で述べたように，変数の型についてはコンポーネントの種類ごとにあらかじめ Camel で定義している。

（7）RPC コンポーネントを使用する場合，Listener 内部に使用する RPC コンポーネントの型で変数を定義し，2 種類のイベントハンドラを定義する。変数名は RPC Manager に登録する ID と同一である必要がある。また，イベントハンドラは完了を意味する Result ハンドラ, 失敗を意味する Fault ハンドラを“コンポーネント ID”+“_”+“メソッド名”＋Result/Fault + ServiceHandler というメソッド名で定義する。例えば WebService コンポーネントを利用する場合, 図 9 で示すように開発者は Camel が提供する“ServiceRepository”タグの子要素として WebService コンポーネントを定義する。このとき,“id”属性で指定した値 (myserv)

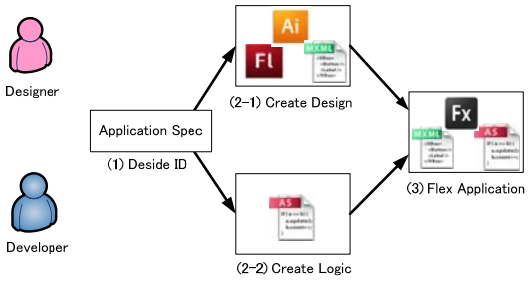


図 10　Camel を用いた開発プロセス

が RPC Manager に登録される。そして，WebService コンポーネントの“doLogin”サービスを呼び出す場合，Listener 内部に“myserv”変数を WebService コンポーネントとして定義し，“myserv_doLoginResultServiceHandler”，“myserv_doLoginFaultServiceHandler”をそれぞれ定義する。

5 適用例と評価

本節では，Camel を用いた開発プロセスについて述べたあと，実際にアプリケーションを構築・実行し，動作を確認するとともに定量的評価を行う。次に，実際に画面の変更を行う場合を例に，Camel の定性的評価を行う。

5.1 Camel を用いた開発プロセス

図 10 に Camel を用いた開発プロセスを示し，番号で示す処理を述べる。前提条件として，アプリケーションに配置するコンポーネントの種類や個数，イベントの処理方法に関する仕様は決定済みとする。

(1) デザイナと開発者でコンポーネント の ID を決定し，共有する。

(2-1) デザイナは画像や CSS，Flash で作成されたコンポーネントを用い，MXML ベースのデザインを作成する。この時，デザイナは手順(1)の決定にもとづき，配置するコンポーネントに ID をつける。この作業は MXML タグの ID プロパティに値を記入することなので，デザイナでも容易に行うことができる。

(2-2) 開発者は Camel の規約にもとづき，クラスやメソッドを実装する。この時，アプリケーションの動作を確認するため

表 1　測定環境

CPU Pentium4 3。2GHz	Pentium4 3。2GHz
Memory	2GB
OS	Windows XPPro + SP2
Browser	Internet Explore 7
プラグイン	Flash Player 9
開発環境	Flex3。0

表 2　CMS の起動時間

	起動時間（ms）	標準偏差（ms）
Camel なし	1,149。5	50。2
Camel あり	1,272	46

に何らかの画面が必要であるが，ここで必要な画面は手順(1)で決定したコンポーネントを含むものであればデザインを意識する必要はないため，開発者でも容易に作成することができる。

(3) 開発者は，デザイナが作成した画面を開発時に用いた画面と置き換える。アプリケーションの動作は手順(2-2)で保障されているため，動作テストは必要ない。

以上の作業によってアプリケーションが完成する。また，アプリケーション完成後，デザインの変更を行う場合には，デザイナが画面のソースを書き換え，変更前のものと置き換えるだけでよい。この時，画面のソースにはプログラムは一切記述されていないため，デザイン変更による動作テストを行う必要はない。ただし，使用するコンポーネントの追加，または処理するイベントの追加や修正が発生する場合には仕様変更とみなし，デザイナと開発者双方の作業が発生する。

5.2 アプリケーション

Camel の動作を確認するため，コンテンツマネージメントシステム(以下，CMS と呼ぶ) を実装した。CMS では，コンテンツの登録，確認，変更，削除，検索機能と，コンテンツのアップロード，ダウンロード機能を備えている。また，これら機能の実現には RPC コンポーネントの 1 つである RemoteObject を利用し，サー

バのデータベースと連携している。また，CMS は全 18 画面で構成され，中央に配置された ViewStack コンポーネントによって画面遷移を行う。その際，フェードイン/アウトの効果が加えられている。

CMS の実行例と 1 画面分のソースコードを図 11 に示す。図 11(a) で示すとおり，この画面には Label，Button，ComboBox，および TextInput など数種類のコンポーネントが合計 45 個配置されている。

5.3 定量的評価

Camel の基本性能を示すため，CMS の起動時間を測定する。起動時間は Flex の Application クラスが発行する“applicationComplete”イベントと，“flash。util。getTimer”メソッドを利用し，Flash プレイヤーが初期化されてからアプリケーションの初期化が完了するまでの時間を測定した。比較対象として, Camel を利用せずに同一アプリケーションの起動時間を測定した。Camel の導入はアプリケーションのルートタグを入れ替えるだけであり，起動時間に影響を与えることはない。表 1 に測定環境，表 2 に起動時間を示す。

Camel はアプリケーション起動時に 4 つのボタンと依存性の確立，および 20 のコンポーネントと値の連結を行う。また，Listener 内部の変数に Model や View，RemoteObject の代入を行い, Result/Fault 用ハンドラと依存性を確立する。その結果表 2 に示すように, Camel を利用しない場合, CMS の起動時間は平均 1,149。5 ms であり, 標準偏差は 50。2 ms であった。

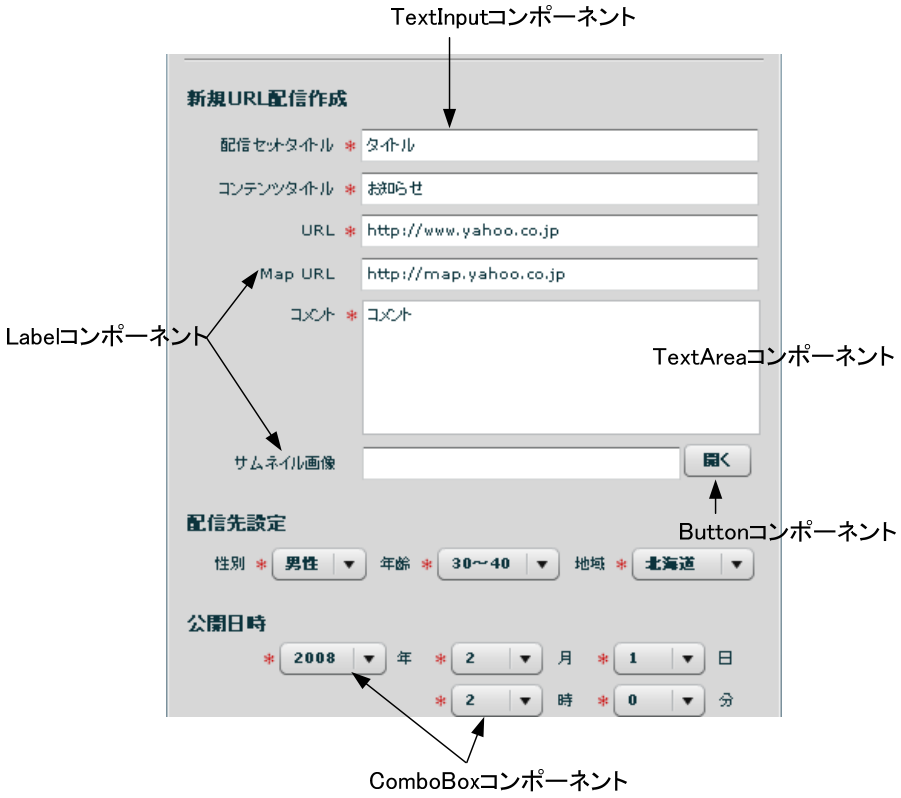
一方, Camel を利用する場合の起動時間は平均 1,272 ms であり，標準偏差は 46 ms であった。Camel を利用する場合，122。5 ms，約 10%程度起動に時間がかかったものの，一度起動したアプリケーションは通常の Flex アプリケーションとして動作するため，実用上問題ない。また, この差は View や Model のパース，依存性の確立に費やした時間であるため，表示コンポーネントの個数が増加するほど時間がかかる。したがって，配置するコンポーネントが多い場合，画面を複数に分割するなど，アプリケーションの設計を工夫する必要がある。

5.4 定性的評価

Camel は画面とロジックをソースレベルで完全に分離することにより，デザイナと開発者間の共有リソースを排除し，デザイン変更によるロジックへの影響を削減することを目的としている。そこで，CMS の 1 画面を変更し，デザインの変更によるロジックへの影響がないことを示すことで，Camel の有効性を確認する。

図 12 と図 13 に変更前と後の画面と，ソースコードを示す。この画面では，期間や地域を絞り込み, コンテンツを検索する機能を提供する。図 12 は Flex コンポーネントをそのまま配置しているが，図 13 ではコンポーネントの配置だけではなく，検索ボタンを Flash で作成したコンポーネントに置き換え，戻るボタンにも自作した画像を埋め込んでいる。検索ボタン上の Flash コンポーネントは，マウスオーバで“検索する”という文字列が浮かび上がるアニメーションが埋め込まれている。また, データグリッドにはマウスダウンで拡大し，マウスアップでもとに戻る効果を加えている。

図 13 のように，レイアウトの変更には Flex が提供する HBox や VBox，Spacer を利用し，コンポーネントの配置を微調整している。また，アニメーションなどの RIA らしい効果は，Button や DataGrid タグの属性を追加して行う。このように，Flex ではコンポーネントに埋め込む素材や効果を，タグの属性として指定して装飾を行う。そのため，デザイナは属性の追加や削除，位置を調整するタグを駆使する必要がある。このとき，既にイベントハンドラの埋込みや，プログラムの埋込みが行われていると，それらを変更しないように配慮する必要があるためデザインに集中できない。また，多くの場合画面の作成には FlexBuilder のデザインビューを利用し，GUI を通してコンポーネントの追加や変更，位置の調整を行うため，デザインの変更には一度コンポーネントを削除し，再配置することも多い。この時，コンポーネントにイベントハンドラが追加されていると，コンポーネントの削除とともにハンドラも削除され，開発者が再び埋め込む必要が出てくる。仮にデザイ



(a)

```
<mx:FormItem x="32" y="90" label="配信セットタイトル" required="true">
    <mx:TextInput width="260" id="usSetTitleInput"/>
</mx:FormItem>

<mx:FormItem x="30" y="119" label="コンテンツタイトル" required="true">
    <mx:TextInput width="260" id="usContentsTitleInput"/>
</mx:FormItem>

<mx:FormItem x="84" y="147" label="URL" required="true">
    <mx:TextInput width="260" id="usUrlInput" text="http://"/>
</mx:FormItem>

<mx:FormItem x="57" y="176" label="Map URL" required="false">
    <mx:TextInput width="260" id="usMapUrlInput" text="http://"/>
</mx:FormItem>

<mx:FormItem x="77" y="204" label="コメント" required="true">
    <mx:TextArea width="260" height="90" id="usCommentArea"/>
</mx:FormItem>

<mx:FormItem x="40" y="302" label="サムネイル画像" required="false">
    <mx:TextInput width="208" id="usThumbNailInput"/>
</mx:FormItem>

<mx:FormItem x="28" y="369" label="性別" required="true">
    <mx:ComboBox id="usGenderBox" width="62"></mx:ComboBox>
</mx:FormItem>

<mx:FormItem x="136" y="369" label="年齢" required="true">
    <mx:ComboBox id="usAgeBox" width="80"></mx:ComboBox>
</mx:FormItem>

<mx:FormItem x="263" y="369" label="地域" required="true">
    <mx:ComboBox id="usAreaBox" width="80"></mx:ComboBox>
</mx:FormItem>
```

(b)

図 11 CMS ツール


```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:VDividedBox width="500" height="100%" verticalGap="0" horizontalGap="0">
        <mx:HBox>
            <mx:Label text="配信コンテンツ修正、削除" fontSize="16" fontWeight="bold" />
            <mx:Spacer width="160" height="5" />
            <mx:Button label="メニューへ戻る" id="returnMenuBtn" />
        </mx:HBox>
        <mx:Spacer width="5" height="20" />
        <mx:HBox>
            <mx:DateField width="120" />
            <mx:Label text="～" />
            <mx:DateField width="120" />
            <mx:Button label="検索する" id="searchButton"/>
        </mx:HBox>
        <mx:Spacer width="5" height="20" />
        <mx:Label text="検索日" />
        <mx:Spacer width="5" height="5" />
        <mx:HBox>
            <mx:RadioButton groupName="tmpG" id="option1" label="登録日"/>
            <mx:RadioButton groupName="tmpG" id="option2" label="最新更新日"/>
            <mx:RadioButton groupName="tmpG" id="option3" label="終了日"/>
        </mx:HBox>

        <mx:Spacer width="5" height="20" />
        <mx:Label text="検索地域" />
        <mx:Spacer width="5" height="5" />
        <mx:HBox>
            <mx:CheckBox id="checkA" label="北海道" />
            <mx:CheckBox id="checkB" label="東北" />
            <mx:CheckBox id="checkC" label="関東" />
            <mx:CheckBox id="checkD" label="中部" />
            <mx:CheckBox id="checkE" label="中国" />
            <mx:CheckBox id="checkF" label="四国" />
            <mx:CheckBox id="checkG" label="九州、沖縄" />
        </mx:HBox>
        <mx:Spacer width="5" height="20" />
        <mx:Label text="2008年06月01日～2008年 06月12日" />
        <mx:Spacer width="5" height="5" />
        <mx:DataGrid width="100%" dataProvider="" doubleClickEnabled="true" id="contentsDataGrid" styleName="dg1">
            <mx:columns>
                <mx:DataGridColumn headerText="日付" dataField="date"/>
                <mx:DataGridColumn headerText="配信セットタイトル" dataField="settingTitle"/>
                <mx:DataGridColumn headerText="タイプ" dataField="type"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:VDividedBox>
</mx:Canvas>
```

図 12　変更前の画面とソースコード


```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Resize id="gridBig" widthFrom="280" widthTo="350" heightFrom="130" heightTo="160"/>
    <mx:Resize id="gridSmall" widthFrom="350" widthTo="280" heightFrom="160" heightTo="130"/>

    <mx:VDividedBox width="800" height="400" verticalGap="0" horizontalGap="0">
        <mx:HBox width="100%">
            <mx:Label text="配信コンテンツ修正、削除" fontSize="16" fontWeight="bold" />
            <mx:Spacer width="100%" height="5" />
            <mx:Button id="returnMenuBtn"
                upSkin="@Embed('img/upBtnImg.png')"
                downSkin="@Embed('img/downBtnImg.png')"
                overSkin="@Embed('img/overBtnImg.png')" width="20" height="20"/>
        </mx:HBox>
        <mx:Spacer width="5" height="20" />
        <mx:HBox width="100%" height="100%">
            <mx:VBox>
                <mx:HBox>
                    <mx:DateField width="120" />
                    <mx:Label text="～" />
                    <mx:DateField width="120" />
                </mx:HBox>
                <mx:Spacer width="5" height="20" />
                <mx:Label text="検索日" />
                <mx:Spacer width="5" height="5" />
                <mx:HBox>
                    <mx:RadioButton groupName="tmpG" id="option1" label="登録日"/>
                    <mx:RadioButton groupName="tmpG" id="option2" label="最新更新日"/>
                    <mx:RadioButton groupName="tmpG" id="option3" label="終了日"/>
                </mx:HBox>
                <mx:Spacer width="5" height="20" />
                <mx:Label text="検索地域" />
                <mx:Spacer width="5" height="5" />
                <mx:HBox>
                    <mx:CheckBox id="checkA" label="北海道"/>
                    <mx:CheckBox id="checkB" label="東北" />
                    <mx:CheckBox id="checkC" label="関東" />
                    <mx:CheckBox id="checkD" label="中部" />
                    <mx:CheckBox id="checkE" label="中国" />
                </mx:HBox>
                <mx:HBox>
                    <mx:CheckBox id="checkF" label="四国" />
                    <mx:CheckBox id="checkG" label="九州、沖縄" />
                </mx:HBox>
                <mx:Spacer width="5" height="20" />
                <mx:Image source="swf/menuSample.swf" id="searchButton"/>
            </mx:VBox>
            <mx:Spacer width="15" height="5" />
            <mx:VBox width="100%" height="100%">
                <mx:Label text="2008年06月01日～2008年 06月12日" />
                <mx:Spacer width="5" height="5" />
                <mx:DataGrid width="100%" height="100%" doubleClickEnabled="true" id="contentsDataGrid"
                    styleName="dg2" mouseDownEffect="gridBig" mouseUpEffect="gridSmall">
                    <mx:columns>
                        <mx:DataGridColumn headerText="日付" dataField="date"/>
                        <mx:DataGridColumn headerText="配信セットタイトル" dataField="settingTitle"/>
                        <mx:DataGridColumn headerText="タイプ" dataField="type"/>
                    </mx:columns>
                </mx:DataGrid>
            </mx:VBox>
        </mx:HBox>
    </mx:VDividedBox>
</mx:Canvas>
```


図 13　変更後の画面とソースコード

ナが注意深くデザインを変更したとしても，誤ってプログラムに手を加えてしまう可能性は否定できないため，デザインの変更後に開発者が確認を行う必要がある。

Camel を利用すれば，画面のソースにはプログラムやイベントハンドラを一切記述する必要が無いため，デザイナはいつでも自由にデザインを変更することができ，負担が軽減される。また，デザインの変更に際し，プログラムの改変が無いことは保証されるため，変更後の動作テストも不要であり，開発者の手を煩わすことは無い。これらの事実だけからも，Camel を用いる利点は明らかである。なお，コンポーネントの追加，または処理するイベントの変更が無い限り，イベントとイベントハンドラの関係は変わらないため，Controller の変更は必要ない。同様に，プログラムの実行結果を反映した Model の変更も不要である。言い換えると，コンポーネントの追加や処理するイベントの変更がある場合には仕様変更とみなし，デザイナと開発者双方の作業が発生する。

6 関連研究

現在，多数のウェブアプリケーションフレームワークが存在している。ZendFramework[12] や Mojavi[13] は MVC アーキテクチャをサポートした PHP 用のフレームワークである。これらのフレームワークでは Camel と同様に名前ベースのルールを設け，リクエストとビジネスロジックの関連付けを行う。また，Smarty[14] などのテンプレートエンジンと連携することによって直接ロジックを埋め込むことなく画面を記述できる。しかし，2.1 節で述べたように，PHP 自体が HTML を用いたウェブアプリケーションを前提としているため，これらのフレームワークでは View と Model を完全に分離して記述することはできない。

Struts[15] は Java ベースの MVC アーキテクチャをサポートしたフレームワークであり，JSTL[16] や Velocity[17] と連携してウェブアプリケーションを構築することができる。しかし Struts では，リクエストとロジックを関連付ける struts-config.xm という設定ファイルを記述する必要があるため，大規模なアプリケーションになるとその管理が難しく，煩雑になる。Spring[18] は DI コンテナの要素を含んだ Java ベースのフレームワークであり，それ自身でプレゼンテーション層，データベースアクセス層のライブラリを持つだけでなく，Struts や Hibernate といった他のフレームワークとも容易に連携することができる。しかし, Struts の struts-config.xm と同様，オブジェクトの生成や DI を行うために XML ベースの設定ファイルを記述する必要があるため，その管理が複雑かつ煩雑になる。それに対し，Seaser2[19] は規約ベースを導入した Java ベースの DI コンテナであり，例外的な処理を除けばほとんど設定ファイルを記述することなくウェブアプリケーションを開発することができる。しかし，これらのフレームワークや DI コンテナを導入したとしても，HTML ベースのアプリケーションでは View と Model を完全に分離して記述することはできない。

一方，Flex はそれ自身が RIA 用のフレームワークであるが，MVC アーキテクチャはサポートしていない。そのため，開発者自身がフレームワークを実装する必要がある。Cairngorm[20] はアドビシステムが開発した Flex 用フレームワークであり, MVC アーキテクチャをサポートしている。しかし, DI コンテナの機能は含まないため，開発者自身がイベントとイベントハンドラの関連付けや値の連結を行う必要がある。そのため, View にイベントハンドラ等の記述を行う必要があり，画面とロジックを完全に独立したソースにすることができない。また，Flash ベースの DI コンテナとしては AOContainer[21] や di-as3[22]が存在する。AOContainer は設計と実装を分離するための軽量コンテナである。AOContainer では, 開発者が設計と実装を分離するクラス設計を行い，XML でオブジェクトを定義することによって，実装クラスを隠蔽したオブジェクトの生成を実現できる。また，オブジェクトの入れ子構造にも対応している。そのため，開発者が新規にクラスを作成し，オブジェクト定義を変更するだけで実装クラスの入れ替えが可能である。di-as3 も同様に, コンテナにオブジェクトを登録し，オブジェクトのライフサイクルをコンテナに委譲することにより，オブジェクトの生成処理とアプリケーションのロジックの分離を実現する軽量コンテナである。また，

AOContainer 同様, オブジェクトの入れ子構造にも対応しているため，オブジェクトとそれに関連するオブジェクトを一回の生成処理で取得することができる。これらの軽量コンテナは，ロジックを実装する開発者にとって非常に有用なものであるが，View で発生するイベントとイベントハンドラの DI はサポートしていない。そのため，View とロジックの分離は実現できない。

Camel はデザイナと開発者の作業を分離することを目的とした DI コンテナであり，起動時に依存性を確立するため，画面とロジックをソースレベルで完全に分離することができる。また，規約ベースの思想を導入しているため，設定ファイルを記述する必要もない。また, Camel は AOContainer や as-ds3 と競合するものではなく，併用が可能であるため，イベントとイベントハンドラの DI を Camel で行い，オブジェクトのライフサイクルを他のコンテナで管理することによって，開発やメンテナンス効率の向上が期待できる。

7 まとめと今後の課題

本論文では，Flex ベースの RIA アプリケーションフレームワーク, Camel の提案と実装を行った。そして，アプリケーションを実装して動作を確認し，起動時間を測定することによって有効性を示した。Camel を用いることによってデザイナと開発者の作業をソースレベルで完全に分離することができるため，画面に変更が生じても開発者の手を煩わすことなく修正できるだけでなく，修正後の動作を保証することができる。また, Camel では規約ベースの思想を導入し，依存性の確立を実現するため，複雑な設定ファイルの管理も必要ない。

今後の課題として，Camel に限らず，DI コンテナの要素を取り入れたフレームワークではコンパイルは通るものの，実行時エラーが多発する。Camel の基本ルールでは，コンポーネント ID やイベント名をもとに関連付けを行うため，タイプミスによる実行時エラーが発生する。Flex アプリケーションの開発は FlexBuilder を用いることが多く，FlexBuilder 自体が Eclipse のプラグインとして提供されているため，FlexBuilder を拡張し，入力の補完やタイプミスを検出するプラグインの提供を検討している。

参考文献

[1] M. Driver and R. Valdes and G. Phifer, “Rich Internet Applications Are the Next Evolution of the Web” Technical report, Gartner, May 2005.

[2] J. Duhl, “White paper: Rich Internet Applications”, Technical report, IDC, November 2003.

[3] Adobe Systems Inc., “Adobe Flex2” http://www.adobe.com/products/flex/.

[4] J. Allaire, “Macromedia Flash MX-A next-generation rich client”, ITechnical report, Macromedia, March 2002.

[5] Adobe Systems Inc., “Adobe Flash CS3 Professional” http://www.adobe.com/products/flash/, 2008.

[6] Laszlo Systems Inc., http://www.laszlosystems.com, 2008.

[7] Sun Microsystems Inc., “Java Web Start Technology”, http://java.sun.com/products/javawebstart/index.jsp, 2008.

[8] Microsoft Inc., “JSmart Client Developer Center”, http://msdn.microsoft.com/smartclient/, 2008.

[9] クジラ飛行机, “Adobe Flex 2 プロフェッショナルガイド” 毎日コミュニケーションズ, September 2007.

[10] Ecma International, “ECMA Script for XML Specification”, December 2005.

[11] 37signals, “Ruby on Rails”, http://www.rubyonrails.org/, 2008.

[12] Zend Technologies, “Zend Framework” http://framework.zend.com/, 2008.

[13] Major Computing., “Mojavi Framework” http://www.mojavi.org/, 2008.

[14] New Digital Group Inc., “Smarty”, http://smarty.php.net/, 2008.

[15] Apache Software Foundation, “Struts”, http://struts.apache.org/, 2008.

[16] Apache Software Foundation, “JSTL”, http://jakarta.apache.org/taglibs/,

2008.
[17] Apache Software Foundation, “Apache Velocity Project”, http://velocity.apache.org/, 2008.
[18] Spring Source, “Spring Framework” http://www.springframework.org/, 2008.
[19] The Seasar Foundation, “Saesar” http://www.seasar.org/, 2008.
[20] Adobe Labs, “Cairngorm Framework”, http://labs.adobe.com/wiki/index.php/Cairngorm, 2008.
[21] Spark Project, “AOContainer”, http://www.libspark.org/wiki/yossy/AOContainer, 2008.
[22] Google Code, “di-as3” http://code.google.com/p/di-as3/, 2008.

著者略歴

[1] 福田浩章

1998 年慶應義塾大学理工学部計測工学科卒業，計算機科学専攻修士，博士(工学) – 2007 年慶應義塾大学。現在，慶應義塾大学大学院理工学研究科特別研究助教。ユビキタスコンピューティング，ソフトウェア工学，リッチインターネットアプリケーションなどの研究に従事。情報処理学会，ソフトウェア科学会，情報システム学会 各会員。

[2] 山本喜一

1969 年慶應義塾大学工学部管理工学科卒業，管理工学専攻修士，工学博士(198 年慶應義塾大学)。現在，慶應義塾大学理工学部情報工学科教授。ソフトウェア科学，特にシステムの動的適合，ソフトウェア工学，ヒューマンインタフェースなどの研究に従事。ACM, IEEE，情報システム学会，日本ソフトウェア科学会, 電子情報通信学会, 日本シミュレーション学会 各会員。情報システム学会理事。