情報処理学会論文誌　プログラミング　Vol. 2　No. 4　1–12 (Aug. 2009)

テンプレート・プログラミングモデルに基づく自動ウェブ・クライアント・サーバ分割

立堀道昭[†1]　鈴村豊太郎[†1]　小野寺民也[†1]

ウェブ・アプリケーション開発では，通常 HTML テンプレートを用いてウェブページの表現部を背後にあるビジネスロジックやオブジェクトから分離するのが事実上の標準プログラミングモデルとなりつつある．本稿では，このテンプレートに基づくプログラミングにおける典型的な慣習に着目し，従来の実装形態とは大きく異なる実装を施したサーバ側テンプレート・エンジンである FlyingTemplate について述べる．FlyingTemplate では，既存のウェブ・アプリケーションのテンプレート・エンジンを置き換えることにより，ウェブサーバの負荷を，自動的に，かつ従来の自動分散機構より安全にクライアントに分担させることができる．FlyingTemplate では，HTML 文書をまるまる生成する代わりに，テンプレートのパラメータ値とクライアント側で動作するブートストラップのコードのみを含む骨子文書を生成する．ブートストラップコードはクライアント側用のテンプレート・エンジンとウェブページのテンプレートをそれぞれサーバから取り寄せることにより，ウェブブラウザのキャッシュを有効利用できる．実験として，SPECweb2005 の Banking アプリケーションをそのまま，テンプレート・エンジンのみ FlyingTemplate で置き換えたところ，キャッシュがよく当たるケースでは，1.6 倍から 2 倍のスループット向上がみられた．

Automatic Web Client-Server Partitioning Based on the Template Programming Model

Michiaki Tatsubori,[†1] Toyotaro Suzumura[†1] and Tamiya Onodera[†1]

Web applications often use HTML template engines to separate the webpage presentation from its underlying business logic and objects. This is now the de facto standard programming model for Web application development. In this paper we proposes FlyingTemplate, which is a novel implementatoin of server-side template engines taking advantage of the nature of template-based Web programming convention. It can replace the template engine of an existing Web application for off-loading server loads on its clients automatically, and more safely than ordinary automatic distribution mechanisms. Instead of producing a fully-generated HTML page, FlyingTemplate produces a skeletal script which includes only the dynamic values of the template parameters and the bootstrap code that runs on a Web browser at the client side. It retrieves a client-side template engine and the payload templates, leveraging browser caches. In our experiment, we tested the SPECweb2005 banking application using Flying-Template without any other modifications and saw throughput improvements from 1.6x to 2.0x at its best mode.

1. はじめに

ウェブ・アプリケーションは，ビジネス・ロジックやビジネス・オブジェクトとウェブページの表現を分離するために，HTML テンプレートを用いたモデル・ビュー・コントローラー・アーキテクチャ[7),11)] に従って実装されることが多い．このアーキテクチャに従うべきであることは，経験を積んだウェブ・アプリケーション開発者にとってはいわば当然の作法であり[2),11)]，事実上の標準プログラミングモデルとなりつつある[3),6)]．

本稿では，テンプレートに基づくプログラミングモデルの性質を利用し，サーバのスループットを向上させるように既存のウェブ・アプリケーションを改善する，新しいサーバ側テンプレート・エンジンの実装を提案する．提案するテンプレート・エンジンは，HTML ページをまるまる生成する代わりに，クライアント側のウェブブラウザ上で動作する骨子スクリプトを生成する．このブートストラップ・コードは，サーバへの要求ごとに変化するテンプレートの引数値のみを含んでおり，テンプレート・データとクライアント側のテンプレート・エンジンを別に取り寄せる．これにより，ウェブブラウザは，比較的変化のないテンプレート・データをキャッシュすることができるようになる．同じテンプレートに基づいて動的に生成されるページ用の他の骨子スクリプトは，キャッシュされたテンプレートを再利用できる．

実現されるアーキテクチャにより，たいていの場合サーバは要求ごとに動的なデータが変化しても骨子スクリプトを用意するだけで済むため，サーバ側の負荷を減らすことができる．改善されたサーバは，基本的には，新たな利用者が現れるたびにクライアント側のテンプレート・エンジンを供給し，また，ユーザが新たなページを訪れるごとにテンプレート・データを供給するだけでよい．もちろん，クライアント側のキャッシュが（ウェブサイトが

†1 IBM 東京基礎研究所
IBM Research, Tokyo Research Laboratory

更新されたり，キャッシュがクリアされたりして）当たらない場合にもこれらの転送は必要になる．キャッシュに当たらない場合でも，テンプレート・エンジンやテンプレートはサーバのファイル・システム上の静的なファイルであり，ウェブサーバにとって，大きく複雑なページを動的に生成するよりは相当に軽い処理となる．

実装上の違いはほとんど透過であり，テンプレートの典型的な使い方をしているウェブ・アプリケーションは，テンプレート・エンジンのすげ替えだけで，変更なしに動作させることができる．実際，我々は PHP 言語[13] 用のテンプレート・ライブラリである Smarty[10] の API をエミュレートするようにしたため，性能実験で用いた SPECweb2005 のアプリケーションで利用されている元の Smarty ライブラリを置き換えることが可能であった．このような単純な置き換えは，少なくとも，SugarCRM のような他のいくつかのウェブ・アプリーケーションで問題なく行えた．

クライアント・サーバの自動分割を単純に適用すると，セキュリティ上の脆弱性の問題が生じるが，我々はこの問題をよく考慮して FlyingTemplate を設計している．FlyingTemplate では，ウェブ・アプリケーションの管理者が認知して制御可能な，簡潔なセキュリティ方針を守るように設計されており，セキュリティの問題のないようにクライアント上でコードが実行される．公に利用できるウェブ・アプリケーションでは，セキュリティはつねに重要である[16]．Hilda[15] にみられるような，典型的な自動クライアント・サーバ分割技術では，サーバ側のロジックの一部を信頼できないクライアント上で動作させることによるセキュリティ上の問題は考慮されていなかった．

本稿の主要な貢献をあげておく．

- 既存のウェブ・アーキテクチャに合った，自動クライアント・サーバ分割へのテンプレート・エンジンによるアプローチ
- 効率的なキャッシュの利用とセキュリティをともなった設計と実装
- 提案したテンプレート・エンジンを業界標準のベンチマークである SPECweb2005 のアプリケーションに適用した実験結果
- 提案のテンプレート・エンジンを用いることによって自動的に得られるパフォーマンスの向上という付加的な優位性を示すことによる，テンプレートに基づくウェブ・アプリケーション開発の潜在的な促進

これらは，国際会議[12] で発表した内容に基づいているが，本稿では，プログラミングの側面に焦点を当てて全体的に再構成している．特に，FlyingTemplate が効率的に動作するために必要なテンプレート記述の制限や，既存のウェブ・アプリケーションに FlyingTemplate を安全に適用するための運用上の注意事項を新たに明らかにする．

本稿の残りは次のように構成されている．まず，2 章で，従来型のテンプレート・エンジンの代表としての参照テンプレート・エンジンのプログラミングモデルと実装について述べる．3 章で，我々の提案する新しいサーバ側テンプレートエンジンの設計である FlyingTemplate について説明する．4，5 章で，効率やセキュリティに影響する実装上の注意点を述べる．

2. テンプレート・プログラミングモデル

テンプレートに基づくウェブ・アプリケーション開発が主流である理由は，主に，ページの表現部，すなわち「View」を，プログラムのビジネス・ロジックやビジネス・データ，すなわち「Controller」と「Model」から分離できることにある．そのようなテンプレートは，ソフトウェア・ライブラリ[7),11)]，プログラミングやモデリング言語の機能[1),3)]，または，ウェブ・アプリケーション・フレームワーク[2),6)] として利用される．本稿では，これらの機能を提供するモジュールをテンプレート・エンジンと呼ぶ．テンプレート・エンジンを利用する利点は，ウェブサイトの見た目部分のカプセル化，明確な表示（ビュー）記述，グラフィック設計者とプログラマの分業のしやすさ，表示設計における部品再利用，見た目の変更の統制，実行系の維持管理性の向上，プロジェクト間での表示に関する成果物の共有，エンド・ユーザによる見た目のカスタマイズとセキュリティの両立，など多岐にわたる．

2.1 テンプレートの使用

一般的に，テンプレート・エンジンを利用するには次の 3 つのステップを踏む．

- 使用するテンプレートを指定する．
- パラメータに値を設定する．
- テンプレートのパラメータ部分に設定された値を埋め込んで実際の HTML 文書を生成する．

アプリケーション・プログラミング・インタフェースは，テンプレート・エンジンにより変わってくるが，基本的には，どのテンプレート・エンジンもこれらのステップを踏んで利用することになる．

本稿では，PHP[⋆1] 用の，Smarty ライブラリ[10] を，テンプレート・エンジンのリファレンス実装として採用する．このライブラリは，XOOPS や SugarCRM など，数多くの，製品相当品質のオープンソース・ウェブ・アプリケーションで利用されている．さらに，Smarty

⋆1 http://php.net/

```
check_login();
$smarty=new SmartyBank;

$summary=backend_get($_SESSION['userid']);

$smarty->assign('userid', $_SESSION['userid']);
$smarty->assign('summary', $summary);
$smarty->display('account_summary.tpl');
```

図 1　テンプレート・エンジンを用いている PHP スクリプト
Fig. 1　A PHP script using a template engine.

は，業界標準のウェブサーバ・ベンチマークである SPECweb2005 [*1] の PHP によるアプリケーション実装でも用いられている．Smarty の様々な配布版のうち，SPECweb2005 とともに配布されている 2.6.7 版を我々の実験では用いた．

図 1 のコード片は，Smarty を用いて書かれた単純な PHP スクリプトであるが，SPECweb-2005 から抽出して説明のために大幅に簡略化したものである．このコードでは，最初に，ユーザ定義の関数 `check_login()` を呼び出して，アプリケーション利用者のアクセス制御を行っている．そして，テンプレート・エンジン・オブジェクトを `SmartyBank` クラスから生成して変数 `$smarty` に代入している．次に，ユーザ定義関数 `backend_get()` を読んで必要な利用者情報をバックエンドのデータベースから取り出している．`$_SESSION` は既定のグローバル変数であり，利用者のセッション情報を保持する連想配列になっている．ここで，`userid` は，利用者識別番号と紐付けられたキーである．取り出した結果は，変数 `$summary` に格納される．最後に，利用者識別番号とバックエンドから得られた利用者情報を，テンプレートを用いて最終的な出力結果として書き出している．PHP 実行系は，その出力結果をフロントエンドのウェブサーバに渡して HTTP 応答に含めることになる．**図 2** は，結果として得られるブラウザ上の表示である．

この例では，Smarty テンプレート・エンジンの使用過程は，2 つのメソッド `assign()` と `display()` の呼び出しからなっている．メソッド `assign()` は，新たな値をパラメータに設定するために呼ばれ，メソッド `display()` は，テンプレートの指定に加え，テンプレートを設定された値で埋めた結果を生成するステップを兼ねている．図 1 の例では，テンプレート・パラメータ ’`userid`’ には利用者識別番号が割り当てられ，別のパラメータ ’`summary`’ には，バックエンド・データベースから得られた情報をもとに構成した結果が割り当てられている．最終的な結果を生成する段階で，テンプレート・ファイルの名前 ’`account_summary.tpl`’ が指定されている．

⋆1 http://www.spec.org/web2005/

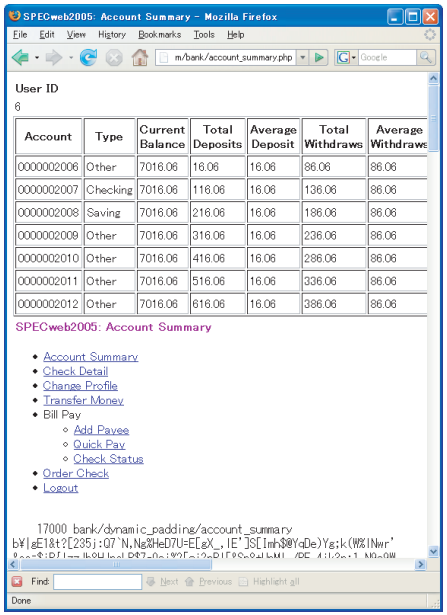

図 2　エンドユーザへの最終的な表示
Fig. 2　A final end-user view.

2.2　テンプレート言語の表現力

テンプレート中で使われる言語要素は，HTML 文書に「穴」をあけるためにパラメータへの参照を埋め込む単純な表現だけではない．典型的なテンプレートを記述するために利用できる文法はもっと豊かである．たとえば，配列の各要素へのアクセスや，パラメータ値に基づく選択，他のテンプレートの参照などがある．さらに，1 つ以上のパラメータ値やリテラル値に基づいて計算を行ったり，中間結果を格納するためにテンプレート内だけの局所変数を使ったり，といったこともよくある．

テンプレート言語自体は，1 つのプログラミング言語にすぎないが，テンプレートは単純な使われ方をして初めて役に立つ場合が多く，複雑なロジックを埋め込んでしまうとその利

点は失われるという特徴がある．たとえば，PHP 自体も，HTML 文書中にコードを埋め込むので，見方によってはテンプレート言語であるといえよう．また，実際のところ，Smarty のような現実に使われているテンプレート・エンジンは，チューリング完全な言語を提供しており，関心事の分離のための特定の規則を守らなければ，その利点を容易に失ってしまうことが知られている[2),11)]．

適度に複雑だが，節度を守ったテンプレート言語要素の利用は SPECweb2005 のアプリケーションにみることができる．**図 3** は，Smarty のテンプレートで，SPECweb2005 の Banking アプリケーションの，口座概要（account summary）ページを表示するために利用されているものである（ただし，説明のために大幅に簡略化してある）．太字で示してある部分が，埋め込まれた「プログラミング」で，他の部分はそのまま表示される HTML テキストになっている．埋め込まれている {$userid} 部分により，テンプレート・パラメータの $userid を参照しており，パラメータ値がその場所へ書き出される．{foreach ..} と {/foreach} の組により，テンプレート・パラメータ $summary が指す配列の各要素について，挟まれた部分を繰り返す．各要素は，新しい局所変数である $acct に割り当てられている．$acct は連想配列であり，その要素は $acct[n] の形（n は連想キー）で参照されている．このテンプレートはまた，選択のための言語要素 if と elseif を使っており，オペレータ eq により真偽値を計算して選択肢を決定している．

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" ..>
<html>
 <head><title>SPECweb2005: Account Summary</title></head>
 <body bgcolor="white">

 <table  summary="SPECweb2005_User_Id">
  <tr><th>User ID</th></tr>
  <tr><td>{$userid}</td></tr>
 </table>

 <table summary="SPECweb2005_Acct_Summary" cellpadding=3 border=1>
  <tr><th>Account</th><th>Type</th><th>Current Balance</th>…</tr>
  {foreach item=acct from=$summary}
    <tr>
    <td>{$acct[0]}</td>
    <td>{if $acct[1] eq "1"}
          Checking
        {elseif $acct[1] eq "2"}
          Saving
        {else}
          Other
        {/if}</td>
    <td>{$acct[2]}</td>
    …… parameterized burdens for other columns ……
    </tr>
  {/foreach}
 </table>
 …… other burdens ……
 </body>
</html>
```

図 3　2 つのパラメータ $userid と $summary をとる，テンプレートの 1 例
Fig. 3　An example template taking two parameters $userid and $summary.

3. Flying Template

FlyingTemplate は，サーバ側で動作するテンプレート・エンジンの亜種である．サーバ側でテンプレートに実際の値を埋め込む作業を行う通常のサーバサイド・エンジンのようなふりをするが，実際には，クライアントにその作業を行わせる．本稿では，参照テンプレート・エンジンとして Smarty を，プログラミング言語として PHP を用いるが，FlyingTemplate の設計自体は他のテンプレート・エンジンやプログラミング言語にも適用できる．

3.1　目標と設計原則

FlyingTemplate の主要な設計目標は次にあげるとおりである．

実行効率　– FlyingTemplate は，少なくとも典型的な状況において，既存のテンプレート・エンジンより高い性能を示さなければならない．

標準の遵守　– ウェブの標準に従うように実装されなければならない．

実装の透過性　– 既存のアプリケーションは変更なしで正しく動作しなければならない．

サーバ・セキュリティ　– FlyingTemplate を導入することで，予期せぬセキュリティ上の脆弱性が生じてはならない．

これらの目標を達成するために，我々は FlyingTemplate を次の設計原則により開発した．

効果的なキャッシュ利用　– ウェブブラウザのキャッシュを有効活用することが，サーバの負荷を減らすうえで重要である．

透過性の適度な妥協　– 実装を完全に透過にする代わりに，テンプレートの典型的な利用にフォーカスし，大概のウェブ・アプリケーションで許されるような緩められた，簡潔なセキュリティ方針を導入する．

本章の残りの部分では，FlyingTemplate の設計とアーキテクチャについて，基本的な考え方を述べる．キャッシュの効果的な利用については別の論文[12)] で述べており，ここでは省略する．

3.2 骨子応答

FlyingTemplate は，すべてが記載された HTML 文書を生成する代わりに，クライアント側で動作するスクリプトを含んだ骨子応答を生成する．そのコードは，HTML 中の JavaScript であり，例を**図 4** に示す．このブートストラップのコードは，次の 2 つの構成要素からなる．

- クライアント側用テンプレート・エンジンのロード
- テンプレートの識別子とテンプレートのパラメータ値が埋め込まれたテンプレート・エンジンを呼び出す部分

例のコード中，filler.js は元のサーバから供給される JavaScript のスクリプトであり，そこで定義されている `fill_template()` を呼んで，テンプレートの適用を実施している．第 1 引数「account_summary.tpl」は，文字列リテラルで表されたテンプレート識別子であり，第 2 引数は，各テンプレート・パラメータの名前と値を対応付ける連想配列のリテラル表現である．

クライアント側のテンプレート・エンジンは内部で，最近のブラウザで提供されている XMLHttpRequest（XHR）と Dynamic HTML（DHTML）の 2 つの機能を利用している．これらの機能は，Ajax 方式のプログラミング[8] の基盤であり，Mozilla Firefox，Microsoft Internet Explorer，Opera，Apple Safari，Google Chrome といった広く普及しているすべてのブラウザで利用できる．XHR では，クライアント側のスクリプトで元のサーバに HTTP 接続を通じて非同期にアクセスすることができる[⋆1]．

```
<script src="/lib/filler.js"></script>
<script>
fill_template("account_summary.tpl",
[["userid","6"],
 ["summary",
  [["0000002006","00","7016.06","16.06","16.06","86.06","86.06"],
   ["0000002007","01","7016.06","116.06","16.06","136.06","86.06"],
   ["0000002008","02","7016.06","216.06","16.06","186.06","86.06"],
   ["0000002009","03","7016.06","316.06","16.06","236.06","86.06"],
   ["0000002010","04","7016.06","416.06","16.06","286.06","86.06"],
   ["0000002011","05","7016.06","516.06","16.06","336.06","86.06"],
   ["0000002012","06","7016.06","616.06","16.06","386.06","86.06"]]]]);
</script>
```

図 4　ブートストラップの JavaScript コードを含んだ骨子応答
Fig. 4　A skeletal result including bootstrap JavaScript code.

⋆1 The XMLHttpRequest Object, W3C Working Draft 15 April 2008, http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/

DHTML では，スクリプトでウェブページ中の HTML 要素を修正することにより，ページをロードした後にページの見た目や機能を追加・変更できる[9]．

ブートストラップのコードは，クライアント側用テンプレート・エンジンと，テンプレートを取り寄せる．**図 5** に示したように，FlyingTemplate による新しいアーキテクチャでは，従来のテンプレート・エンジンを備えたウェブサーバとクライアント間の相互作用と比較して，付加的な相互作用が必要となる．ブートストラップのコードはテンプレート・エンジンのスクリプトを DHTML の機能を利用して取り寄せてロードする．これは，新たに `<script>` タグを追加して，その `src` 属性をテンプレート・エンジンのスクリプトの URL を指すようにすることによってできる．さらに，XHR を用いてテンプレートを取り寄せる．最後に，ロードされたテンプレート・エンジンは，テンプレートとパラメータ値に基づいて HTML 文書を生成し，DHTML の機能を利用して生成結果を HTML の内容と挿入することにより，ブラウザの GUI として描画されるようにする．

3.3 サーバ側テンプレート・エンジン

FlyingTemplate のサーバ側の実装は，ウェブ・アプリケーションの既存のテンプレート・エンジンを一見模した動作をし，2.1 節で述べたようなテンプレート・エンジンと同じ使い勝手を提供する．Smarty[10] の場合は，図 1 で見せたような `assign()` と `display()` のアプリケーション・プログラミング・インタフェースを提供する必要があった．

ロジックの実装の肝は，**図 6** に示したような，クライアント側コードの生成部分である．ここでは，`assign()` の実装は省略しているが，その実装は単に与えられたキーと値の組をテンプレート・エンジン・オブジェクトの表に登録していくだけである．実際のところ，これは元の Smarty の実装となんら変わりはない．

コード生成のアルゴリズムもまったく簡潔である．図 4 で見せたようなブートストラップのコードを生成するにあたり，エンジンはまず，テンプレート識別子とパラメータ値のスクリプト上のリテラル表現を用意する．PHP では，`json_encode()` という組み込み関数を利用して PHP オブジェクトを，JSON 形式[5] と呼ばれる，JavaScript のリテラル表現に変換することができる．そのうえで，コード生成器は，`<script>` タグで囲んだテンプレート呼び出しコードに，準備したテンプレート識別子とパラメータ値を埋め込む．

図 6 では省略したが，後述するセキュリティ上の理由により，`display()` 内部では，検査と分岐のコードがあってもよい．検査においては，テンプレートが，限定された言語要素や表現形式のみを用いて骨子応答に効率良く安全に置き換えられるような形態になっているかどうかを調べる．そうでない場合には，従来と同じ通常のテンプレート処理を行う必

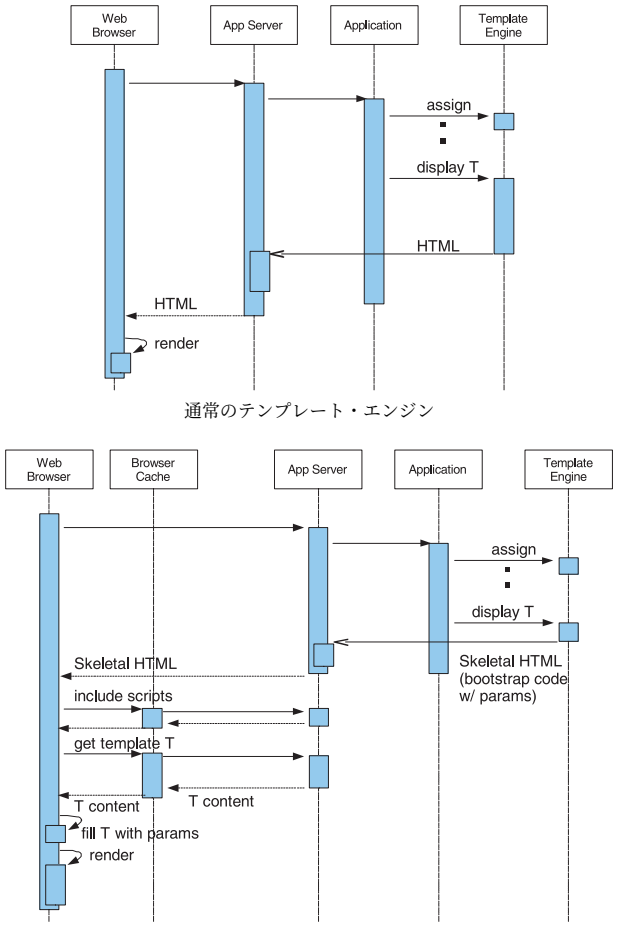

図 5　通常のテンプレート・エンジン（上）と FlyingTemplate によるテンプレート・エンジン（下）による相互作用の違い

Fig. 5　Interaction sequences with a normal template engine (top) and the FlyingTemplate template engine (bottom).

要がある．検査自体が必要かどうか，検査する場合にはどのようなテンプレートの利用形態が許されるかどうかは，元々提供されているテンプレート・エンジンの仕様に依存する．

```
function display($template_id) {
  $js_template = "\"{$template_id}\"";
  $js_params = json_encode($this->params);
  echo '<script src="/lib/filler.js"></script>';  // template engine
  echo "<script>fill_template({$js_template},{$js_params});</script>";
}
```

図 6　サーバ側のコード片

Fig. 6　Server-side code fragments.

テンプレート・エンジンの実行時に検査を行うと大きなオーバヘッドになりかねないが，このような検査結果はキャッシュしておくことができる．通常，Smarty のような効率的なテンプレート・エンジンの実装では，テンプレートの構文解析などの結果を直接実行できるコードに変換しておくという最適化を行うが，テンプレートの検査はその一環として含めることができるため，実行時の解析コストはなくて済むようにできる．

4. テンプレート記述とセキュリティ

我々は，ウェブ環境におけるセキュリティをよく考慮して FlyingTemplate を設計している．公に利用できるウェブ・アプリケーションでは，セキュリティはつねに重要である[16] からだ．FlyingTemplate のような，ウェブ環境における自動クライアント・サーバ分割機構では，元々サーバ側で動作するように設計されたロジックの一部を信頼できないクライアント上で動作させるため，必ずセキュリティ上の問題をはらんでいる．本章では，FlyingTemplate がこの問題にどのように対処しているかを述べる．

4.1 セキュリティ方針

一般に，元々サーバ側で動作するように開発したウェブ・アプリケーションの一部をクライアント側で動作させるようにすると，セキュリティ上の脆弱性を生む可能性がある．特にサーバの観点からは，サーバ・データの秘匿性（confidentiality）と，サーバ・データ入力の完全性（integrity），アクセス制御上の重大な問題を生じる．例として，1 章で述べたウェブ・アプリケーションを考える．元々サーバ側で PHP コードとして動作していたロジックをクライアント側に移すと，改変されたウェブブラウザを用いるなどすれば，`check_login()` を飛ばして直接 `backend_get()` を呼び出すことにより，任意の利用者としてバックエンドのデータベースにアクセスできてしまう（アクセス制御の問題）かもしれない．仮に認証された利用者であったとしても，バックエンドのデータベースに直接アクセスして，元々はフィルタがかけられて表示されないようにしていたデータを読み取ってしまう（秘匿性の問

題）かもしれないし，データベース内のデータを改変してしまう（完全性の問題）かもしれない．

セキュリティも含め，元々のウェブ・アプリケーションのセマンティクスをどうやって完全に保持するのかという問題は，興味深い研究であるが，FlyingTemplate では，セキュリティについては若干妥協することにした．そして，完全に元のセキュリティを保つ代わりに，容易に受け入れられるような，簡潔で理解しやすい形でセキュリティ方針を譲歩してもらうことにした．譲歩したセキュリティ方針は，

- テンプレート・データに秘匿性は必要ない

というものである．

ビューをモデルとロジックから切り離すというテンプレート・プログラミングモデルの金科玉条により，このセキュリティ方針の譲歩は，ほとんどのウェブ・アプリケーションにとって，合理的に受け入れられるものであると我々は信じる．ビューの設計者は，ウェブ・アプリケーションのサーバ側のセキュリティに関わるべきではないからである．実際，多くの Smarty を用いたウェブ・アプリケーションを動作させているウェブ・サイトでは，テンプレート・データが誰でもアクセスできる場所においてあることがままある．

逆にいうと，FlyingTemplate を適用する場合，ウェブ・アプリケーションの管理者は，テンプレートにセキュリティ上の問題ある記述が含まれていないことに留意している必要がある．一般的には，テンプレート中に含まれるロジックの情報が外に漏れてはまずい場合がこれにあたる．たとえば，図 3 のテンプレートにおける `if` と `elseif` においては，パラメータの値によって表示を Checking，Saving，Other の中から 1 つ選び出している．パラメータの値により Checking が選ばれている場合には，本来，最終的な HTML からは Saving や Other について知る由もなかったはずである．FlyingTemplate を適用することにより，クライアントはこのテンプレート自体を入手できるようになるため，Saving や Other など，本来は得られなかった情報も得ることができるようになる．通常，テンプレートに重要な情報は含まれていないので問題はないが，ウェブ・アプリケーションの管理者は，これらの情報が開示されてしまうことに留意していなければならない．

4.2　テンプレート記述の制限

テンプレートの記述の内容によっては，上で述べたようなセキュリティ方針の妥協があっても，FlyingTemplate を安全に適用することが難しくなる場合がある．そのような言語要素は大きく分けて次の 3 つである．

- システム・リソースへのアクセス
- 外部サブテンプレートの参照

前者は，テンプレート・エンジンにより，ファイル入出力やユーザ定義関数の呼び出しなど，サーバのリソースにアクセスするようなインタフェースが提供され，それらをテンプレート内で利用している場合である．後者は，テンプレート内で別ファイルのテンプレートを参照してテンプレート全体の一部を構成する場合である．これら以外の，テンプレート内で閉じた演算に関しては，所詮クライアント側テンプレート・エンジンで再現できるため大きな問題とはならない．

システム・リソースへのアクセスをクライアント側のテンプレート処理中に許すためには，サーバ側でそれらのアクセスのための要求を受け付けるようにしなければならない．同様に，サブテンプレートを参照できるようにするためには，クライアント側のテンプレート処理中にさらにサブテンプレートをサーバから取り寄せられるようにしなければならない．こういった機構を実現するためには，サーバ側で慎重にアクセス制御をしなければならず，そのために実行性能の低下も生じることが容易に予測される．さらには，クライアント・サーバ間の通信遅延があるため，システム・リソースへのアクセス回数が多い場合には，クライアント側のページ表示に要する遅延が実用にならない程度に増大する恐れもある．

前述した問題を含むテンプレートに対しては，テンプレートの内容を解析し，問題がある場合には従来型のすべてサーバ側で行うテンプレート処理をするようにすればセキュリティの問題は解消される．該当するテンプレートについては実行性能の向上を得られなくはなるが，そうしたテンプレートの割合が少なければ，全体としての実行性能の向上は得られる．さらには，参照されるテンプレートが静的に決まっている場合には，テンプレートを解析して事前に参照されているテンプレートを内包してしまうなどの対処が可能である．一方で，静的な解析一般にみられる問題だが，関数名やファイル名を計算結果の文字列により与えるなどの処理が行われている場合には，問題となる言語要素の使用を検出できない．そのため，セキュリティ上の問題の可能性を排除しつつ動作するためには，解析不能なテンプレートについては，保守的に問題のあるテンプレートとして取り扱う必要がある．

5. 性 能 評 価

我々は SPECweb2005 ベンチマークで提供されているアプリケーションの 1 つである Banking シナリオに FlyingTemplate を適用することによってその有効性を評価した．本章では，性能向上実験の結果を示す．

5.1 実験環境

Fedora Core 7（カーネル 2.6.23）OS 上にウェブサーバとして Lighttpd 1.4.19 を用い，3.4 GHz の Xeon プロセッサと 2 GB の RAM を備えた IBM IntelliStation M Pro 上で動作させた．Lighttpd は，親プロセス 1 つと，作業プロセス 2 つ，そして FastCGI プロセス 8 つとなるように設定した．負荷生成のためには，ギガビット・イーサネットで接続された 2.0 GHz のプロセッサを備えた IBM IntelliStation M Pro を擬似クライアントとして構成した．PHP ランタイムとして，繰り返して構文解析をする余分なオーバヘッドをなくすために APC（Alternative PHP Cache）をつけた PHP 5.2.6（php.net）を用いた．サーバ・スループットを測るために，Apache の JMeter 2.3.2 を用いた．これらの環境は，我々の経験[14] に基づいて注意深く調整したものである．

5.2 SPECweb2005 アプリケーションの性能

我々のアプローチの妥当性を示すため，ベンチマークの業界標準団体の SPEC によって策定された SPECweb2005 を用いて実験を行った．SPECweb2005 では，3 層アーキテクチャがとられており，ウェブサーバ，PHP ランタイムと，データベースの振舞いを装う BESIM シミュレータからなる．ここで用いる Banking アプリケーションは SPECweb2005 で提供されている 3 つのアプリケーションのうちの 1 つであり，SSL 接続されたオンライン銀行のウェブ・アプリケーションを模している．PHP スクリプトにより動的に生成されるページは，口座情報のチェックや別口座への送金など 12 のページがある．**表 1** は，このオンライン銀行シナリオにおける各ページを表示するために転送されたデータ量の平均を示したもので，元の構成と FlyingTemplate を適用したものを比較している．FlyingTemplate により，転送データが大幅に削減されていることがみてとれる．

表 1　各ページごとに転送されたデータ量の平均（バイト）
Table 1　Average data size transferred for each page.

ページ	従来手法	FlyingTemplate
account_summary	18,195	361
check_detail_html	12,731	271
profile	34,295	327
change_profile	24,566	289
transfer	30,393	255
post_transfer	15,067	283
place_check_order	26,438	288
order_check	25,678	312
bill_pay_status_output	23,584	378
quick_pay	17,821	294
bill_pay	35,400	270
post_payee	20,136	208
add_payee	17,781	327

図 7 は，元の構成と FlyingTemplate による，サーバのスループット性能を比較している．縦軸は単位秒あたりの平均ページ数であり，横軸にそれぞれのページの種類を並べている．グラフでは，1，2，4 とクライアント構成を変化させて測定したものを掲載しているが，1 クライアントあたり，Java のマルチスレッドにより 24 同時接続がなされる．グラフに示されているように，1 クライアントの場合で，FlyingTemplate は元の構成を最小で 59%（post_transfer ページ）最大で 104%（profile ページ）上回った．

5.3 プロファイル結果

図 8 は，SPECweb2005 Banking シナリオの口座概要（account summary）について，ページ単一要求（ページ）処理に要した CPU 時間の相対的な長さを示している．図 8 は，ウェブ・アプリケーションの実行に関わる主要なプロセスである PHP 実行系とウェブサーバの各プロセスの処理にかかった時間を示している．**図 9** と**図 10** はさらに，それぞれのプロセス内のモジュール中の処理にかかった時間を示している．図に示されているように，

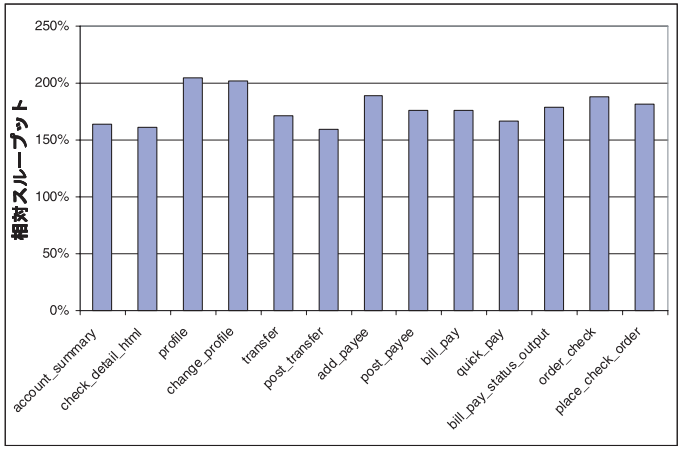

図 7　SPECweb2005 Banking シナリオにおける FlyingTemplate を用いた場合のスループット（対既存のテンプレート・エンジン）
Fig. 7　Throughput improvement with FlyingTemplate for a SPECweb2005 Banking scenario (compared to an existing template engine).

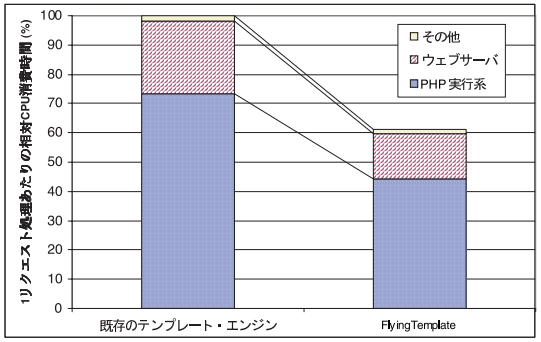

図 8　単一要求処理に要する相対 CPU 時間
Fig. 8　Relative CPU time for processing a single request.

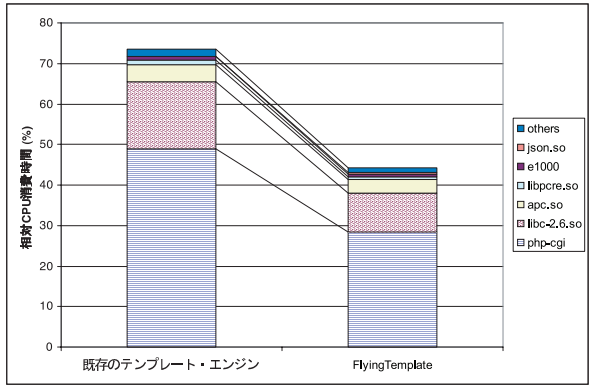

図 9　PHP 実行系プロセス内の各モジュール中の処理に要する相対 CPU 時間
Fig. 9　Relative CPU time for processing each module in PHP runtime processes.

各処理部にかかる CPU 消費の比は大体同じである．FlyingTemplate で新たに必要となった JSON の処理にかかった CPU 時間は 1 パーセント以下と，非常に小さかった．このプロファイルを得るためには，oprofile と呼ばれるプロファイル・ツールを用いた．

5.4　キャッシュ・ミスの影響

提案手法は，テンプレート・エンジンのすべての動作がクライアント側で行われ，必要となるファイルが多くの場合ブラウザのキャッシュにあることを仮定している．これらのファイルには，クライアント側テンプレート・エンジンのスクリプト自体とテンプレートの 2 種類があり，それぞれのキャッシュ・ヒット率は異なる性質を持つ．テンプレート・エンジンは通常，1 つのサイト内の複数のページやテンプレートについて共通である．そのキャッシュ・ミスは，なんらかの理由で利用者がブラウザのキャッシュをクリアされてしまう稀な場合を除けば，特に，新規の利用者がそのウェブサイトを訪れた場合にのみ発生する．対照的に，テンプレートは，動的に生成されるページの種類ごとに異なるのが普通である．たとえば，SPECweb2005 の Banking シナリオにおける，預金概要ページは，同シナリオのプロファイル情報ページなどの他のページとは異なるテンプレートを用いている．したがって，テンプレートのキャッシュ・ミスは，テンプレート・エンジンよりは頻繁に発生し，また，テンプレート・エンジンの場合と同様に，ブラウザ・キャッシュのクリアなどによっても発生する．

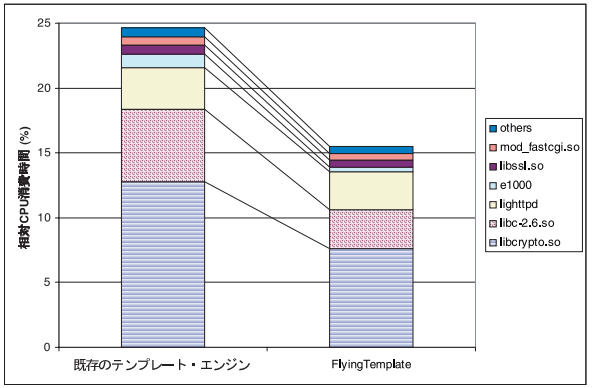

図 10　ウェブサーバ・プロセス内の各モジュール中の処理に要する相対 CPU 時間
Fig. 10　Relative CPU time for processing each module in Web server processes.

キャッシュの影響を調べるため，我々は，キャッシュ・ヒット率を仮想的に変化させてウェブ・アプリケーション・サーバのスループットの変化を測定した．**図 11** は，クライアント側テンプレート・エンジンのキャッシュ・ヒット率を変化させて調べたものである．テンプレート・エンジンのキャッシュ・ミスの際にはテンプレートのキャッシュもミスするのが通常であるため，テンプレート・キャッシュも同様に変化させた．**図 12** は，テンプレートのキャッシュ・ヒット率を変化させて調べたものである．この際は，テンプレート・エンジンのキャッシュがつねにヒットしている状況に固定した．

テンプレート・エンジンのキャッシュがミスする場合には，大きなパフォーマンスの低下

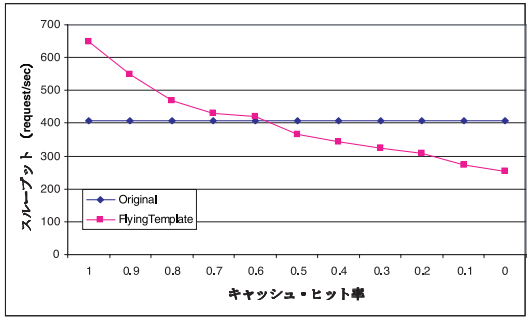

図 11　クライアント側テンプレート・エンジンのファイルのキャッシュ・ヒット率低下にともなうスループットの低下
Fig. 11　Throughput for various virtual cache hit ratios of client-side template engine script files.

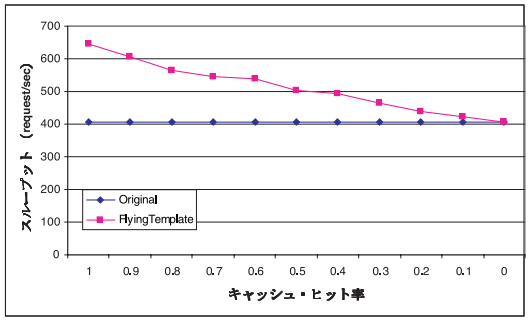

図 12　テンプレート・ファイルの様々なキャッシュ・ヒット率低下にともなうスループットの低下
Fig. 12　Throughput for various virtual cache hit ratios of template files.

がみられた．キャッシュ・ヒット率が 0.5 まで低下している場合には元のスループットよりも低下している．しかし，これは，2 ページに 1 ページの割合で新たなユーザが現れるような状況に相当し，通常起きるとは考えにくい．

一方で，テンプレートのキャッシュがミスする場合のパフォーマンスの低下は小さく，特に，キャッシュ・ヒット率が 0 の場合でも，元のスループットと同様のスループットが得られている．これは，FlyingTemplate では，PHP ランタイムを煩わせることなく，テンプレートのような大きなファイルをブラウザに配信できる効果によるものである．

6. 関 連 研 究

6.1　セキュリティ

分散コンピューティングの研究の観点からは，FlyingTemplate は，Hilda [15] のような，ウェブ環境向けの自動分散システムの 1 つと見なすことができる．しかしながら，元々サーバ側で動作するように実装された通常のプログラムを完全に自動的にクライアント側に分割して動作させるようにすると，ここまですでに述べてきたように，セキュリティ上の問題が生じる．

我々の知る限り，Swift [4] は唯一，ウェブ環境において強くセキュリティを意識して自動クライアント・サーバ分割システムを提案している．プログラマにセキュリティ上の注釈をつけることを強制することにより，Swift は，特別な，しかし統一されたプログラミング言語で書かれたプログラムが自動的に分割され，クライアント側の JavaScript とサーバ側の Java として動作するようにしている．セキュリティ上の注釈に従い，保安上重要なコードやデータはつねにサーバ側に保たれるようにしている．分割されたシステムが効率良く動作するかどうかは，プログラム全体の込み入った解析によっている．解析はいつも完全にできるわけではないので，最適な結果を得られるとは限らない．FlyingTemplate によるアプローチは，テンプレートに基づくプログラミングモデルのしきたりによった効率的な分割のヒューリスティクスを与えることにより，セキュリティ上の注釈作業やプログラム解析の必要性を大幅に減らしているものと見なすこともできる．

6.2　クライアント側のテンプレート

スタイルシートや XSLT は，HTML や XML 文書のテンプレートであると考えることもできる．特に，XSLT の表現力は強力で，トランスレータを書くことで，動的に生成された XML 要素から，複雑な HTML 文書を生成することができる．残念ながら XSLT は複雑すぎて，ビュー設計者の使用には向いていないが，Java や PHP のようなサーバ側のプログラミング言語と，XSLT を同時に使いこなせるプログラマは，自動的にクライアント・サーバ分割の恩恵を受けることができる．実のところ，我々は，別の版の FlyingTemplate では，XSLT スタイルシートと関連付けられた XML 文書を骨子スクリプト相当として生成するようなものも実装してみている．その際，元のテンプレートから XSLT ファイルを生成することはできたが，同時に，不完全な HTML 文書を生成するような XSLT を書くことは非常に難しいことが分かった．そして，そういった不完全な HTML 文書は，SPECweb2005 も含め，既存のウェブ・アプリケーションではよくあることである．

7. ま と め

本稿では，サーバ側で動作するテンプレート・エンジンでありながら，HTML 文書を生成する作業の多くをクライアントのブラウザに自動的に分担させる FlyingTemplate を提案した．このクライアント・サーバの自動分割が効率良く安全に行われるのは，テンプレートに基づくプログラミングモデルを仮定している点が大きい．実験では，SPECweb2005 の Banking アプリケーションにおいて，セキュリティ上の問題やアプリケーションの改変なく，FlyingTemplate を適用するだけで，最善の場合には 1.6 倍から 2.0 倍のスループット向上が得られることを示した．

参 考 文 献

1) Arnoldus, J., Bijpost, J. and van den Brand, M.: Repleo: A syntax-safe template engine, *Proc. GPCE 2007, 6th International Conference, Generative Programming and Component Engineering*, Salzburg, Austria, October 1–3, 2007, pp.25–32, ACM (2007).
2) Böttger, H., Møller, A. and Schwartzbach, M.I.: Contracts for Cooperation between Web Service Programmers and HTML Designers, *Journal of Web Engineering*, Vol.5, No.1, pp.65–89 (2006).
3) Ceri, S., Fraternali, P. and Bongio, A.: Web Modeling Language (WebML): A modeling language for designing web sites, *Computer Networks: The International Journal of Computer and Telecommunication Networking*, Vol.33, No.1-6, pp.137–157 (2000).
4) Chong, S., Liu, J., Myers, A.C., Qi, X., Vikram, K., Zheng, L. and Zheng, X.: Secure web application via automatic partitioning, *Proc. SOSP 2007, the 21st ACM Symposium on Operating Systems Principles 2007*, Stevenson, Washington, USA, October 14–17, 2007, pp.31–44, ACM (2007).
5) Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON) (19992006), RFC 4627.
6) Fraternali, P.: Tools and Approaches for Developing Data-Intensive Web Applications: A Survey, *ACM Comput. Surv.*, Vol.31, No.3, pp.227–263 (1999).
7) García, F.J., Castanedo, R.I. and Fuente, A.A.J.: A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications, *Proc. ICWE 2007, 7th International Conference on Web Engineering*, Como, Italy, July 16–20, 2007, LNCS 4607, pp.442–456, Springer (2007).
8) Garrett, J.J.: Ajax: A New Approach to Web Applications (2005). http://adaptivepath.com/ideas/essays/archives/000385.php
9) Goodman, D.: *Dynamic HTML: The Definitive Reference*, O'Reilly (2006).
10) New Digital Group, Inc.: Smarty: Template Engine. http://www.smarty.net/
11) Parr, T.J.: Enforcing strict model-view separation in template engines, *Proc. WWW 2004, the 13th International Conference on World Wide Web*, New York, NY, USA, May 17–20, 2004, pp.224–233, ACM (2004).
12) Tatsubori, M. and Suzumura, T.: HTML Templates that Fly: A Template Engine Approach to Automated Off-loading from Server to Client, *Proc. 18th International Conference on World Wide Web, WWW 2009*, Madrid, Spain, April 20–24, 2009, pp.951–960 (2009).
13) Tozawa, A., Tatsubori, M., Onodera, T. and Minamide, Y.: Copy-on-Write in the PHP Language, *Proc. POPL 2009, the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Savannah, Georgia, USA, January 21–23, 2009, pp.200–212, ACM (2009).
14) Trent, S., Tatsubori, M., Suzumura, T., Tozawa, A. and Onodera, T.: Performance Comparison of PHP and JSP as Server-Side Scripting Languages, *Proc. Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference*, Leuven, Belgium, December 1–5, 2008, pp.164–182, Springer (2008).
15) Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A.J., Gehrke, J. and Shanmugasundaram, J.: A unified platform for data driven web applications with automatic client-server partitioning, *Proc. WWW 2007, the 16th International Conference on World Wide Web, Banff*, Alberta, Canada, May 8–12, 2007, pp.341–350, ACM (2007).
16) Yu, D., Chander, A., Inamura, H. and Serikov, I.: Better abstractions for secure server-side scripting, *Proc. WWW 2008, the 17th International Conference on World Wide Web*, Beijing, China, April 21–25, 2008, pp.507–516, ACM (2008).

(平成 21 年 2 月 20 日受付)
(平成 21 年 5 月 10 日採録)

立堀　道昭（正会員）

1974 年生．2002 年筑波大学大学院工学研究科博士課程修了．同年日本アイ・ビー・エム（株）入社．以来，同社東京基礎研究所にて，プログラミング，分散システム，セキュリティの研究に従事．現在，同研究所主任研究員．工学博士．日本ソフトウェア科学会，ACM 各会員．

鈴村豊太郎（正会員）

1975 年生．2004 年東京工業大学大学院情報理工学研究科・数理計算科学専攻博士課程修了．同年日本アイ・ビー・エム（株）入社．以来，同社東京基礎研究所にて，XML 処理や PHP 言語処理等ソフトウェアシステムに関する性能最適化の研究に従事．現在，同研究所インフラストラクチャ・ソフトウェアグループ主任研究員．2009 年より東京工業大学大学院情報理工学研究科客員准教授．理学博士．情報処理学会 HPC 研究会運営委員．ACM Member．

小野寺民也（正会員）

1959 年生．1988 年東京大学大学院理学系研究科情報科学専門課程博士課程修了．同年日本アイ・ビー・エム（株）入社．以来，同社東京基礎研究所にて，オブジェクト指向言語の設計および実装の研究に従事．現在，同研究所シニア・テクニカル・スタッフ・メンバー．インフラストラクチャ・ソフトウェア担当．第 41 回（平成 2 年後期）全国大会学術奨励賞，平成 7 年度山下記念研究賞，平成 16 年度論文賞，平成 16 年度業績賞各受賞．理学博士．ACM Senior Member．日本ソフトウェア科学会会員．