サーバサイドイメージレンダリングを用いたスマートデバイス対応 Web UI コンポーネントフレームワーク

Smart Device Enabled Web User Interface Component Framework using Server Side Image Rendering

モバイルファースト，クラウドファーストが提唱され，業務アプリケーションのスマートデバイスへの対応が必須な状況となっている。(株)日立ソリューションズ東日本はSynViz S2のガントチャートやPSI Visualizerに代表されるUser Interface(UI)コンポーネントを得意としているが，これらのコンポーネントは描画に必要な計算量が多く処理性能の低いスマートデバイスで画面表示を行うと十分な応答性能が得られない。また，画面表示に必要なデータ量も多く，低速なネットワーク環境下では初期表示までの待ち時間が長くなる問題もある。そこで，負荷の高い描画処理をサーバ上で実行しクライアントの負荷を低減する方式を利用した Web UI コンポーネントフレームワークを提案する。本手法によりクライアントの処理負荷が低減され，多くの計算量を必要とする画面をスマートデバイス上で表示可能となる。さらに，提案したコンポーネントフレームワークを利用することで開発工数を低減できることを Web アプリケーションの試作を通して確認した。

内海　宏律　Utsumi Hironori
菊地　大介　Kikuchi Daisuke
黄　双全　Huang Shuangquan
石倉　直弥　Ishikura Naoya
齋藤　邦夫　Saito Kunio
手塚　大　Tezuka Masaru

1．はじめに

スマートデバイスが広く普及する中，業務システムのスマートデバイス対応が求められている。スマートデバイス対応とするためには，既存のアプリケーションをWeb化し，端末のWebブラウザ上で利用する方法がある。この方法では，Model-View-Controller(MVC)アーキテクチャ[1)]を適用し図1のような構成をとることが多い。

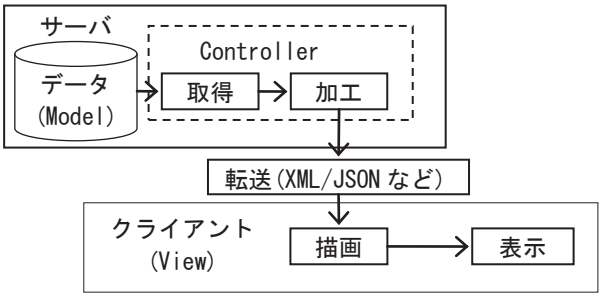

図1　MVCアーキテクチャを利用したWebシステムの構成

この方式はサーバ上に格納したデータモデルをクライアントのリクエストに応じてコントローラが取得と加工を行う。その結果はExtensible Markup Language (XML)[2)]などの文字列情報としてシリアライズ化され，クライアントに転送される。ビューとなるクライアントは受信したデータの表示位置を計算するなどして描画処理を行い，画面上に表示する。この方式はプロジェクト管理システムであるSynViz S2でも利用されている。クライアントの処理にはHTML5[3)]やCanvas[4)]など標準化された技術を用いることで，様々なデバイスのWebブラウザに対応できる。

(株)日立ソリューションズ東日本(以下，HSEと記す)ではSynViz S2のガントチャート[5)]画面や在庫可視化ソリューションであるPSI Visualizerの在庫可視化画面など，データの可視化技術を得意としている。これらの可視化画面は大規模なガントチャートや多数のグラフの同

時表示画面などを利用した複雑なデータ表示が可能な反面，データの表示位置の計算処理や多数のグラフオブジェクトの描画処理など，描画時に多くの計算量を必要とする。また，画面表示のために必要となるデータ量もガントチャート全体のデータが必要になるなど，単純なグラフコンポーネント(情報を表示したりユーザが操作したりするためのソフトウェア部品)と比較すると多くの情報が必要になる。これらのコンポーネントをスマートデバイスで表示した場合，端末の処理性能が低いため描画処理に時間がかかり画面がスムーズに応答しないなどの問題が発生する。Canvasの描画を最適化する手法[6]も提案されているが，画面全体が再描画対象となる場合には効果がないなどの問題がある。また，無線通信のような低速なネットワーク環境下では描画に必要なデータ全体を送る従来手法では初期表示までの待ち時間が長くなる。通信量削減のために必要なデータだけを差分転送する手法も存在するが，データの分別はデータモデルの構造や描画アルゴリズムに強く依存する。そのため汎用的な処理は困難となり，実現のためには描画処理と同等の処理が必要になる。これら問題を解決し，スマートデバイス上で可視化画面を表示するコンポーネントを開発するためのWeb UIコンポーネントフレームワーク(Webで用いられるUIコンポーネントを開発するための基盤)を提案する。さらにSynViz S2のガントチャートとPSI Visualizerの画面を外出先などでスマートデバイスから閲覧する想定でビューアを試作し評価を行った。

2．サーバサイドイメージレンダリングを用いたWeb UI コンポーネントフレームワークの提案

2.1 サーバサイドイメージレンダリングによる課題解決

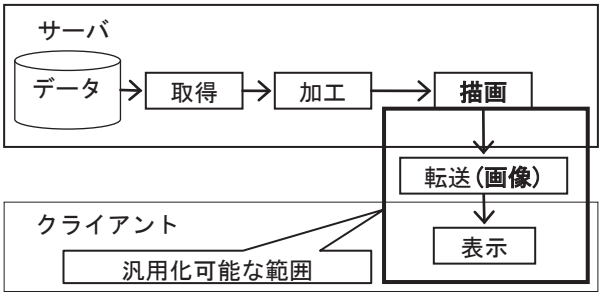

図2　サーバサイドイメージレンダリング方式

処理性能の低いクライアントサイドで負荷の高い処理を行っているためクライアントの応答性が低下していることに着目した。そこで，図2のように負荷の高い描画処理をサーバ上で実行するサーバサイドイメージレンダリング方式を提案する[7]。

提案方式では従来クライアントサイドで実行していた描画処理をサーバ上で実行する。描画結果は画像データとしてクライアントに転送し画面表示を行う。クライアントはWebブラウザ上に描画が完了した画像データを表示するだけとなり，高速なUIの応答が実現できる。また，描画結果は画像データとなるため，描画後の処理についてはデータモデルに依存せずに汎用的な処理が可能となる。そのため，従来手法では困難であったデータの分別についても，画面表示に必要となる範囲の画像を送信するという汎用的な処理として容易に実現可能となる。画面表示に必要となるデータ量は，高々Webブラウザの画面解像度と同等の画像データとなる。つまり，通常のWebページの閲覧時と同等のデータ転送量で初期画面の表示が可能となる。提案手法は，表1に示すように従来手法と比較し60%程度の時間で描画処理を完了でき，初期表示までに必要となるデータ転送量も表2のように10%程度まで削減できることを予備実験[8]で確認した。

表1　描画処理時間

Canvas 描画方式	238ms
サーバサイドレンダリング方式	138ms

表2　初期表示までに必要となるデータ転送量

XML データ	664kB
PNG 画像(初期表示領域のみ)	56kB

2.2 Web UIコンポーネントフレームワークの提案

サーバサイドイメージレンダリングを用いた場合，描画結果は画像データとなるため描画以降の処理はデータモデルに依存しない。この点に着目し，図2に示す「汎用化可能な範囲」を再利用可能なコンポーネントフレームワーク[9]として提案する。

クライアントのWebブラウザ上にガントチャートやグラフなどの可視化結果を表示するにあたり，以下の機能要件を設定した。

要件1：マルチデバイス，マルチプラットフォームで利用できる
要件2：画面スクロールやズームなどの操作ができる
要件3：ガントチャートのカレンダーやグラフの目盛りなどの領域は常時画面内に表示できるようにする
要件4：データの選択時などに選択状態の表示など，UIの操作に対するフィードバックができる

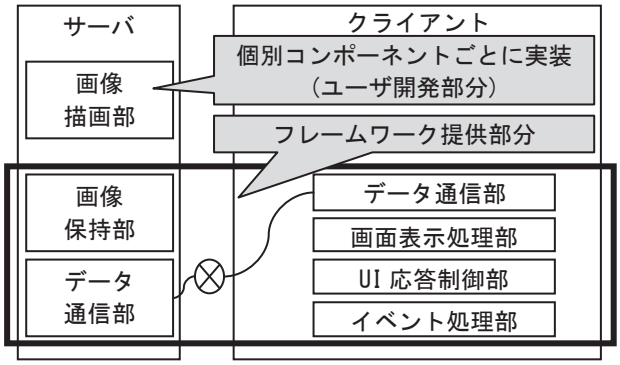

図3　Web UIコンポーネントフレームワークの構成

上記をふまえ，コンポーネントフレームワークの構成を図3に示すように設計した。サーバ上で描画した結果を画像保持部が受け取り，クライアントの画面表示に必要な部分だけをデータ通信部を介して転送し画面に表示する。提案するコンポーネントフレームワークのサポート環境は表3に示すとおりである。クライアントサイドでは標準化された技術を利用しているため，プラットフォームに依存せずに多くの環境で利用できる。

表3　コンポーネントフレームワークのサポート環境

サーバ	Windows Server 2012 R2 IIS 8.5, .NET Framework4.5
クライアント	HTML5 および JavaScript をサポートする Web ブラウザ （Internet Explorer, Google Chrome, Firefox など主要なブラウザに対応）
その他	WebSocket が利用可能な環境

要件1はHTML5など標準化された技術を利用することで実現可能であるが，WebブラウザごとにUIイベント[10]の仕様や実装が異なる。そこで，フレームワーク内のイベント処理部でプラットフォーム間の差異を吸収し，抽象化したUIイベントとして発行する機構を備える[7]。これによりフレームワークの利用者はデバイスの差異を意識する必要がなくなり，同一のイベントインタフェースを利用した開発を行うことができるようになる。要件2については，クライアントのUIに対する操作を応答制御部が受け取り，スクロールやズームなどの応答をフレームワーク内で完結させる構造とした。これにより，基本的なUIの応答について，アプリケーション側はコーディングレスで実現できる。要件3については，画面の上下左右に常時表示される「固定領域」を設定可能とし，様々な画面構成に柔軟に対応できる汎用性をもったフレームワークとした。要件4については，UIの操作結果をサーバに送信しサーバ上で画像更新を行った場合，通信速度が低速な環境では応答性が悪化する。そこで，画面上にクライアントサイドのプログラムで描画が可能なCanvasレイヤーを提供し，クライアントだけで操作に対するフィードバック表示を可能とした。

3．Web UI コンポーネントフレームワークによるアプリケーションの Web 化

本章では，提案したWeb UIコンポーネントフレームワークを適用し，既存のアプリケーションをスマートデバイス対応とする方法について検討する。

3.1 スタンドアロンアプリケーションの構成

ガントチャートやPSI Visualizerなどのように工程状況や在庫情報を入力とし，その可視化結果を出力とするようなスタンドアロンアプリケーションの内部処理を抽象化したものを図4に示す。

クライアント側にあるCSVやXMLなどの可視化対象データを，アプリケーションが提供するファイルダイアログなどのデータ入力インタフェースを介してアプリケーションに読み込む。アプリケーションはそのデータを入力とし，固有のロジック(処理エンジン)を用いてソートやデータの加工などの処理を行う。その結果は画面に表示するためにレンダラへと渡される。レンダラはグラフやガントチャートなどのような可視化結果として描画を実行する。最後にその結果をデバイスコンテキストなどの画面表示インタフェースを介して画面に表示するフローで処理が行われる。

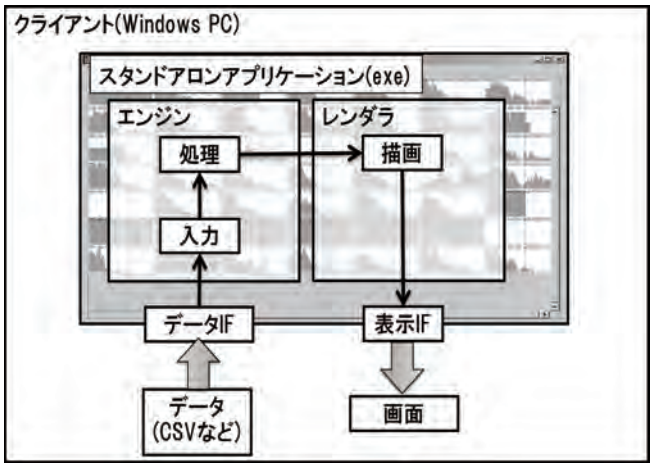

図4　スタンドアロンアプリケーションの処理フロー例

3.2 Webアプリケーションの構成

SynViz S2のようなWebアプリケーションの場合はサーバ上のDBを入力データとし，図4の左側をサーバサイド，右側の描画以降の処理をクライアントサイドで行っているだけで実質的な処理フローはスタンドアロンアプ

リケーションの場合と同一といえる。サーバとクライアントの間では，データをJSON形式などの文字列にシリアライズ化することで受け渡しを行っている。

3.3 コンポーネントフレームワークによるWeb化

スタンドアロンアプリケーションのWeb化を考えた場合，クライアントとサーバを利用したMVCモデルを適用し，サーバサイドにモデルおよびコントローラ，クライアントサイドにビューを配置した構成とするなどの方法が利用される。ただし，今回はフレームワークの適用にあたり，画面の描画をサーバサイドで行う構成とするためレンダラ部分もサーバ上で実行する構成とする。つまり，図4で挙げたアプリケーション固有のロジック部分(エンジン部分)およびレンダラ部分をサーバ上で実行する構成とし，図5のように「入力」から「描画」までをサーバサイドで実行する。アプリケーションの入力となるデータは，クライアント上にあるデータを利用する場合はネットワークを介してサーバにデータをアップロードし，そのデータをアプリケーションの入力とする。また，近年のクラウド化されたシステムではデータがクラウド上のデータベースなどに集約されていることも多い。その場合には，データベースと接続する部分を新規に作成し，それをアプリケーションの入力として利用する構成とする。アプリケーションの描画部分では，メモリ上に確保したビットマップ上に描画を行い，その結果をフレームワークに渡す構成とする。なお，画像の差分取得に対応するため，描画した画像は任意の大きさのブロックに分割した後にフレームワークへと渡す。ビットマップを受け取ったフレームワークはその後の処理を担い，クライアントに画像を送信し画面上に描画結果を表示するところまでを実行する。

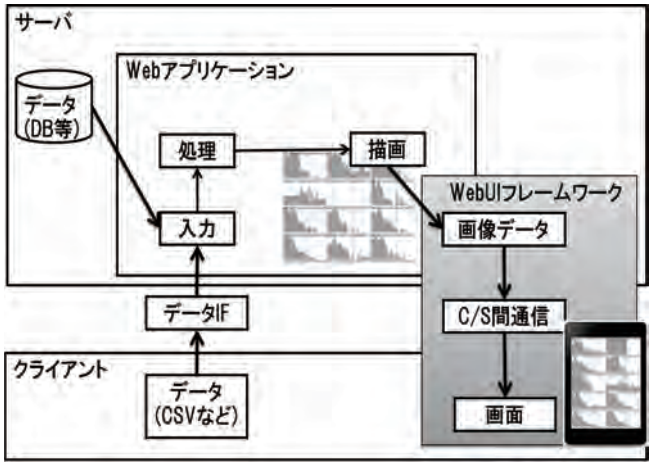

図5　コンポーネントフレームワークによるWeb化

この場合，クライアントとサーバの実行環境が双方ともに同一のプラットフォームであれば，図5に示した「入力」から「描画」までの処理は既存のソースコードを流用できる。つまり，アプリケーションの開発者は「入力」から「描画」に該当するソースコードをサーバ上に移植した後，本フレームワークを適用し入力データを受け取る部分と描画結果をフレームワークに渡す部分を新規に開発するだけでよい。画面のスクロールやズームなど，UIの操作に対する応答や画像の転送部分はすべてフレームワークが実行するため，データのビューアを想定したアプリケーションであればクライアントに関する作りこみは基本的に不要となる。また，フレームワークはマルチデバイス，マルチプラットフォームに対応しているため，クライアント端末やブラウザの種類に応じた条件分けなどの処理も基本的には不要である。

Webアプリケーションの場合は，従来クライアントで実行していた「描画」をサーバサイドで実行することとなる。そのうえで，描画結果を画像データとしてクライアントに送信し表示するが，この部分は先に述べたとおりWeb UIコンポーネントフレームワークが実行する。つまり，データの表示(ビューア)相当の機能であれば，データの入力から処理までの部分はすでにサーバ上に構築されているものをそのまま流用し，「描画」部分をサーバ上に移設するだけで対応ができる。

以上より，提案するWeb UIコンポーネントフレームワークを適用することで，既存のアプリケーションを容易にスマートデバイス対応にできる。

４．コンポーネントフレームワークの評価

4.1 評価アプリケーションの概要

提案したフレームワークを実際のアプリケーションの Web 化に適用した想定で評価を行った。題材には Web アプリケーションである SynViz S2 のガントチャート画面とスタンドアロンアプリケーションである PSI Visualizer を利用し，スマートデバイス上からデータを閲覧するビューアとして利用可能な Web アプリケーションを試作し評価を行った。それぞれのアプリケーションでは提案したコンポーネントフレームワークを利用し，各可視化コンポーネントを作成した。PSI Visualizer のデータビューアは図 6 に示す 3 種類の画面で在庫の状況を可視化する仕様とした。

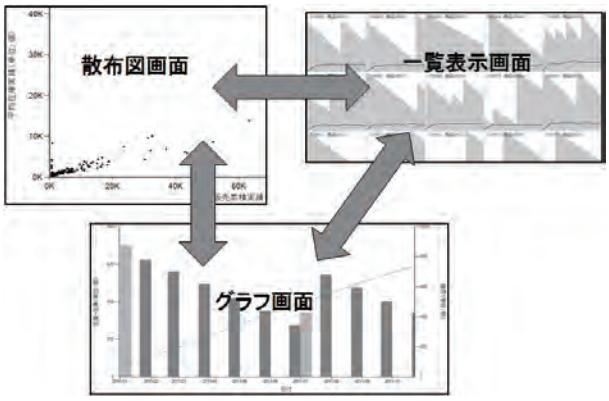

図6　PSI Visualizerデータビューアの画面

ガントチャートビューアでは図 7 のように SynViz S2 上に存在するプロジェクト一覧を表示し，任意のプロジェクトのガントチャートを表示する仕様とした。なお，それぞれのコンポーネントはタッチ操作での画面のスクロールとズームに対応する仕様とした。

図7　ガントチャートビューアの画面

4.2 表示対象画面の解像度に対する性能評価

提案したコンポーネントフレームワークがサーバ上で描画される画像の解像度の影響を受けずに，一定の時間で画面表示ができることを確認する。評価には入力データ数とサーバで描画される画像の解像度が比例関係にある PSI Visualizer の「一覧表示画面」を利用した。性能測定は表 4 に示す環境で実行した。処理時間は画像をコンポーネントフレームワークが受け取り内部に格納する画像処理時間，クライアントとサーバそれぞれの内部処理時間，ネットワークの通信時間に分けて計測した。

表4　性能測定環境

サーバ	Windows Server 2012 R2 Intel core i7 4770 3.4GHz / 4GB
クライアント	Android 4.4.4 APQ8064 Quad Core 1.5GHz / 2GB Firefox 表示領域の解像度 970×423 ピクセル
ネットワーク	Wi-Fi 接続(IEEE802.11n)

性能測定結果を図 8 に示す。サーバ上の画像解像度によらずほぼ一定の時間で画像データを転送し画面表示が完了できることがわかる。これは，クライアントに画面表示に必要な画像だけを転送しているため，画面描画以降の処理はサーバ上の画像解像度に依存しないためである。また，データ数が 5000 個になった場合に性能劣化が見られるが，これはサーバのメモリが枯渇し，実行プラットフォームでメモリの解放処理が行われるようになった影響である。なお，データ数が 5000 個の場合にはサーバ上で約 30 万×970 ピクセルの画像が生成される。

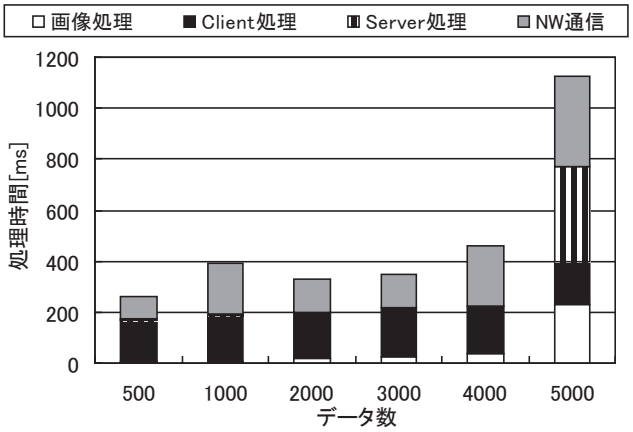

図8　データ数と処理時間の関係

4.3 生産性評価

提案したコンポーネントフレームワークを利用して既存のアプリケーションをスマートデバイス対応の Web アプリケーションとする際に必要な開発規模を測定した。

4.3.1 コンポーネントフレームワークの規模

提案するWeb UIコンポーネントフレームワークのステップ数は表5に示すとおりである。クライアントサイドはJavaScript[11]で実装されており，実行時にクライアントに転送される。ステップ数は10ks(キロステップ)程度であるため,その転送時間は大きな問題とはならない。

表5　コンポーネントフレームワークのステップ数

サーバサイド	3.2ks
クライアントサイド	10.6ks
合計	13.8ks

4.3.2 ガントチャートビューアの開発規模

ガントチャート画面の描画には,.NET環境でガントチャート画面を表示するコンポーネントであるガントX for .NET[12] (以下，ガントX)を利用した。SynViz S2に格納されたデータをガントXのデータ入力機能を利用しガントチャートオブジェクトを構築し，画面イメージを画像データとして描画する。ガントチャートビューアのステップ数は表6に示すとおりである。データ入力部や画像描画部は既存のガントXを流用している。また，画像の転送処理や画面のスクロール，ズームはコンポーネントフレームワークの機能をそのまま利用している。

表6　ガントチャートビューアのステップ数

流用	47.8ks
新規	2.5ks

4.3.3 PSI Visualizerの開発規模

グラフ画面の描画には既存の.NETコンポーネントを利用した。その他コントローラなどの部分は既存の.NETアプリケーションからの流用としている。新規開発分を含むステップ数を表7に示す。

表7　PSI Visualizerデータビューアのステップ数

流用	18.2ks
新規	4.6ks

4.3.4 コンポーネントフレームワークによる削減効果

コンポーネントフレームワークの機能のうち 8 割を利用したと仮定すると 13.8ks の 8 割である 11.0ks 分が再利用されている。つまり，それぞれのアプリケーションでは本来，新規開発分に加え上記ステップ数を開発する必要があった。コンポーネントフレームワークの適用による開発工数の削減効果を確認するため，本来必要であったステップ数に占めるコンポーネントフレームワークのステップ数を削減割合として表 8 に示す。提案したフレームワークは，スクロールなどの UI の操作に対する応答処理など汎用的な機能をサポートするため，ビューアアプリケーションでは開発工数を高い割合で削減できる。

表8　フレームワーク適用による開発工数削減効果

アプリケーション	本来必要なステップ数	新規開発ステップ数	削減割合
ガントチャート	13.5ks	2.5ks	81%
PSI Visualizer	15.6ks	4.6ks	70%

5．おわりに

HSE では工程の進捗状況や在庫状況などの情報を可視化するコンポーネント技術を得意としている。これらのコンポーネントは複雑な表示が可能な反面，描画処理の負荷が高く処理性能が低いスマートデバイス上では十分な応答性が確保できない。そこで，負荷の高い処理をサーバサイドで実行する方式を提案した。さらにコンポーネントの開発に再利用できるようにコンポーネントフレームワーク化した。提案方式を利用すると表示する画面の解像度によらず一定の時間で画面表示が可能なことを示した。また，ビューアアプリケーションを試作し，開発ステップ数を削減できることも示した。

提案する Web UI コンポーネントフレームワークを適用することで，スマートデバイス上で高速に画面を表示するコンポーネントを容易に作成可能となる。今後は SynViz S2 のガントチャート画面など, HSE 製品への適用を進め，スマートデバイス対応コンポーネントを拡充することで顧客満足度向上に貢献していく予定である。

参考文献

1) Model View Controller, https://ja.wikipedia.org/wiki/Model_View_Controller, Accessed 2015/8.
2) Extensible Markup Language (XML) 1.1, http://www.w3.org/TR/xml11/, Accessed 2015/8.
3) HTML5, http://www.w3.org/TR/html5/, Accessed 2015/8.
4) HTML Canvas 2D Context | W3C, http://www.w3.org/TR/2dcontext, Accessed 2015/3.
5) ガントチャート，https://ja.wikipedia.org/wiki/ガントチャート, Accessed 2015/8.
6) HTML5 キャンバスのレイヤー化によるレンダリングの最適化, http://www.ibm.com/developerworks/jp/web/library/wa-canvashtml5layering/, Accessed 2015/8.
7) 内海 宏律, 黄 双全, 菊地 大介, 齋藤 邦夫, 手塚 大, “サーバサイドイメージレンダリングによる Web UI コンポーネントフレームワークの試作と評価”, FIT2015 第 14 回情報科学技術フォーラム 講演論文集, (2015).
8) 内海 宏律, 菊地 大介, 浦邊 信太郎, 齋藤 邦夫, 手塚 大, “サーバサイドイメージレンダリングによるウェブ UI コンポーネント・フレームワーク”, 情報処理学会第 77 回全国大会論文集分冊 3, pp.65-66, (2015).
9) ソフトウェアフレームワーク, https://ja.wikipedia.org/wiki/ソフトウェアフレームワーク, Accessed 2015/8.
10) UI Events, http://www.w3.org/TR/DOM-Level-3-Events/, Accessed 2015/8.
11) Standard ECMA-262, http://www.ecma-international.org/publications/standards/Ecma-262.htm, Accessed 2015/8.
12) 宍戸 政則, 門司 太郎, 齋藤 邦夫, "ガントチャート作成支援コンポーネント「ガント X」の開発", 日立 TO 技報 第 12 号, pp.23-26, (2006).

内海　宏律　　2006 年入社
研究開発部
Web システムおよびソフトウェアコンポーネントに関する研究開発
hironori.utsumi.zc@hitachi-solutions.com

菊地　大介　　2009 年入社
研究開発部
高機能 Web アプリケーション開発基盤技術の研究開発
daisuke.kikuchi.hz@hitachi-solutions.com

黄　双全　　2010 年入社
研究開発部
在庫管理や生産計画シミュレーション等意思決定支援技術の研究開発
shuangquan.huang.dc@hitachi-solutions.com

石倉　直弥　　2009 年入社
研究開発部
Web UI コンポーネントの研究開発

naoya.ishikura.kk@hitachi-solutions.com

齋藤　邦夫　　1992 年入社
研究開発部
自社パッケージ製品・ツールの研究・開発
kunio.saito.uh@hitachi-solutions.com

手塚　大　　1994 年入社
研究開発部
研究戦略の立案，研究開発の推進

masaru.tezuka.fd@hitachi-solutions.com