B-012

Web 標準技術による Web UI コンポーネントの実現に向けた 描画性能の見積による実現可能性の検討
Rendering Performance Estimation to Develop Web UI Component using Web Standards

内海 宏律† 太齋 真吾† 石倉 直弥†
Hironori Utsumi Shingo Dasai Naoya Ishikura

1. はじめに

主要な Web ブラウザでの Flash Player のサポート終了が発表され，既に Google Chrome などいくつかのブラウザにおいては Flash を利用した Web UI コンポーネントの実行が抑止されるようになった[1][2]。これに伴い，Flash を活用した Web UI コンポーネントは Flash Player に依存しない技術への移行が求められている。Web ブラウザの Web 標準への準拠状況も考慮すると HTML5[3]などの Web 標準技術を活用する方法があるが，Flex と比較し描画性能が遅い[4]ため，移行後の性能を早期に明確化する必要がある。そこで，Web 標準技術を用いたコンポーネントの実現可能性の判断に向け，画面上に表示されると見込まれるオブジェクトの数をもとに描画性能を見積もるツールを試作した。本ツールによる見積結果と既存の Flex コンポーネントから得られる実行速度をもとに，Web 標準技術を用いた Web UI コンポーネントの移行可能性を明確化する手法を提案する。

2. Web 標準技術への移行における課題

Flash を活用した業務向け Web UI コンポーネントの特徴として，HTML のみでは困難な大規模なデータの表示や Web ブラウザ上でのマウス操作による編集機能などのような高い操作性を実現している点がある。これらのコンポーネントでは，多数のオブジェクトが配置される中で，画面操作へのスムーズな応答が要求される。Flex と比較し描画性能が低い HTML5 への移行では，多くのオブジェクトが配置された際の描画性能を具体的に見積もることができないため，十分な応答速度が確保できるか判断できない点が課題である。また，Web 標準と呼ばれる中にも複数の描画技術が存在する[4]が，それらの性能特性が不明確であり，選択すべき描画技術の決定が困難である課題がある。

3. Web 上で利用可能な描画技術

Web 標準技術を用いた Web UI コンポーネントの実現可能性の検討に向け，描画技術の調査を行った。本章では Web ブラウザおよびサーバサイドで利用可能な描画技術に関する調査結果を説明する。

3.1 Web ブラウザにおける描画技術

主要な Web ブラウザにおいて JavaScript を利用した画面描画に利用できる技術のうち，既に標準化されているものを表 1 に示す。

Canvas 2D Context(Canvas)[5]と WebGL[6]は GDI[7]同様に即時モード[8]と呼ばれる描画モデルを持つ技術である。描画 API の呼び出し結果は図 1 のように逐次，メモリ上のビットマップに出力される。描画内容はブラウザの画面更新のタイミングに合わせ画面上に表示される。WebGL による描画は Graphics Processing Unit(GPU)によって処理されるため，対応する GPU の搭載が必須となり，クラウドなどで活用が進む仮想化環境での動作は保証されない。一方，Scalable Vector Graphics(SVG)[9]は HTML や Flex 同様の保持モード[8]と呼ばれる描画モデルを採用する。保持モードでは図 1 のようにメモリ上に描画のためのオブジェクトモデルを保持し，そのモデルをプログラムから操作する。その結果に対し，Web ブラウザや Flash Player などのプラットフォームが最終的なレイアウトを計算し，画面上に描画する。保持モードではオブジェクトの描画処理はプラットフォーム側によって行われ，JavaScript ではモデルの操作のみを行う。

表 1 Web において標準化された描画技術

	Canvas	SVG	WebGL
標準化団体	W3C	W3C	Khronos Group
描画モデル	即時モード	保持モード	即時モード
描画方法	描画 API の呼び出し	SVG DOM の操作	描画 API の呼び出し
要件	なし	なし	GPU が必要

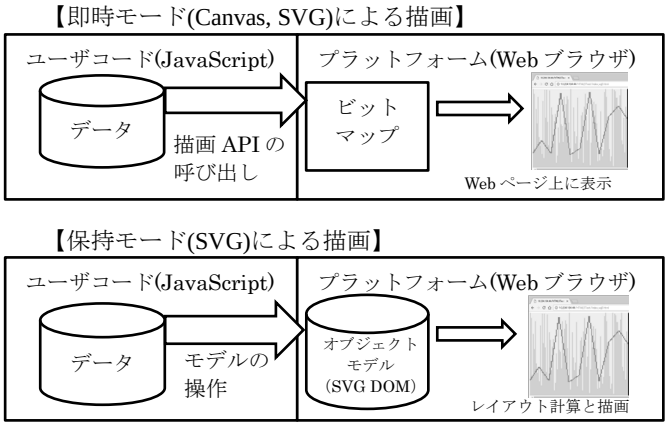

図 1 描画モデルごとの処理フロー

性能測定においては，プラットフォーム側の処理も含めた時間を計測する必要がある。そこで，性能測定においては Web ブラウザが画面を描画する直前に発生する requestAnimationFrame の発生間隔を測定することで，1 枚の画像の表示に必要な時間を計測する。

3.2 サーバサイドでの JavaScript を用いた描画技術

スマートデバイスなどのような処理性能の低いクライアントに対し，計算量の多い可視化コンポーネントを表示するためには，サーバサイドで描画処理を行いその結果をクライアントに転送，表示する手法が有効である[10]。Web

†（株）日立ソリューションズ東日本, Hitachi Solutions East Japan, Ltd.

標準技術を用いた場合，クライアントサイドは JavaScript で実装する必要がある。ソースコードの保守性などを考慮すると，サーバサイドでの描画処理もクライアントと同一のソースコードを用いて実現するのが望ましい。そこで，JavaScript の利用を前提としサーバサイドでの描画技術の適用可能性を調査した。

サーバサイドでの JavaScript の実行は，OSS である Node.js[11]を利用することで実現できる。SVG および WebGL については，サーバサイドで利用する際に，表 2 に示す課題が判明したため，評価対象から除外した。

表 2　サーバサイドで利用する場合の課題

描画技術	課題
SVG	1. SVG DOM をメモリ上に構築する方法がない 2. HTML DOM をエミュレーションする jsdom は Canvas の 35〜73 倍程度遅く実用的でない
WebGL	対応する GPU が必須であり，クラウドで広く活用されている仮想化環境上で動作しない

4. 描画性能の測定と移行時の性能見積方法

Web UI コンポーネントを移行する際の性能見積を目的とし，画面内に描画されるオブジェクトの種類と数に着目した性能測定を行う。そのうえで，既存の Flex コンポーネントを Web 標準技術へ移行する際の性能見積手法を提案する。

4.1 画面構成要素に基づく性能測定方法の提案

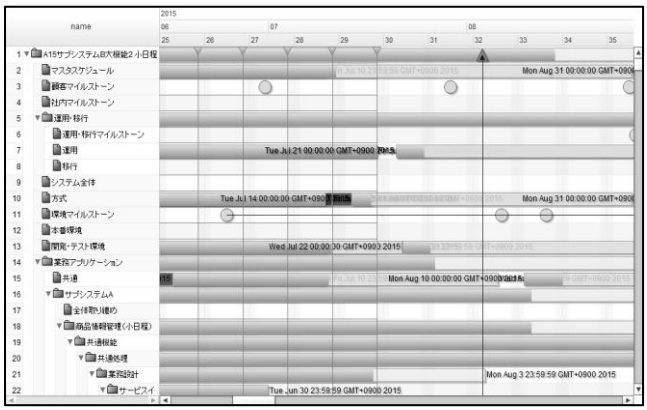

図 2　工程状況を可視化するガントチャート画面

表(グリッド)やグラフなどのような業務向け Web アプリケーションへの適用を想定したデータ可視化コンポーネントでは，画面に描画される内容は矩形や直線，円などの図形と文字列や画像の組み合わせから構成される。例えば，図 2 に示す工程管理で用いられるガントチャートと呼ばれるコンポーネントでは，縦方向に作業項目，横方向にカレンダーを配置して作業の状況を可視化する。画面を見ると直線や矩形，文字列，画像などの組み合わせで構成されていることが分かる。Web UI コンポーネントの描画性能を見積るためには，画面上に表示されると見込まれる図形の種類と個数を算出し，それと同様の内容を Web 標準技術を用いて描画した場合の描画性能を測定すればよい。

そこで，描画する図形の種類と個数を入力とし，表 1 に示した 3 つの描画技術を利用して同一の画面を描画し，描画時間を測定，比較するツールを作成した。同時に，サーバサイドで Canvas を用いて描画した場合の速度も測定する。本ツールにより，画面内に表示されるオブジェクトの種類と数を適切に指定することで，実アプリケーションを想定した描画性能の測定が可能になる。

4.2 移行時の性能見積方法の提案

移行においては描画性能の他に，コンポーネント固有のロジックなどの描画以外の処理時間も見積る必要がある。そこで，4.1 節の手法で実アプリケーションを想定した描画性能の見積を行った後，既存の Flex コンポーネントから得られる処理時間を利用した性能見積方法を提案する。

4.2.1　実アプリケーションを想定した描画性能の見積

Web 標準技術を用いたコンポーネントの性能見積を目的とし，実際の UI コンポーネントを想定した性能見積を行う。表 3 に示すように，表計算，ガントチャート，グラフ一覧表示の 3 種類の UI コンポーネントを想定した。ガントチャートでは，図 2 に示すような標準的なプロジェクトの工程状況の可視化をモデル化した「小日程」モデルと画面内がオブジェクトで埋め尽くされるような状況をモデル化した「大日程」モデルの 2 パターンの表示を想定した。それぞれの UI コンポーネントでは PC の 1280×1080px(ピクセル)のディスプレイで上部と左部にメニューがあるレイアウトでの表示および iPad3 での全画面表示を想定し，1000×600px の画面サイズとした。そのうえで画面内に描画されると想定されるオブジェクト数を算出し表 3 のようにパラメータ化した。

表 3　表示内容を想定した描画パラメータ

	表計算	ガント 小日程	ガント 大日程	グラフ 一覧
オブジェクトサイズ	72×18px	200×20px		100×100px
矩形	476	127	531	60
三角	0	21	174	0
円	0	30	84	0
直線	114	578	1293	6600
テキスト	504	161	967	180
画像	0	49	389	0

表計算コンポーネントでは，72×18px のセルが画面内に並べて表示され，各セルにテキストが入力され，9 つの系列を持つ折れ線グラフ形式のグラフオブジェクトが 1 つ表示されている状況をモデルとした。グラフ一覧表示コンポーネントでは，100×100px のグラフが画面内に並べて表示されると仮定した。画面内には 60 個のグラフが表示され，各グラフは 100 本の直線で値の推移を示し，10 本の直線から構成される折れ線およびラベルとして 3 つのテキストが表示される状況をモデル化した。ガントチャートコンポーネントでは，「小日程」モデルとして，図 2 の画面を元に画面内に含まれるオブジェクト数を算出した。また，大日程モデルでは，画面上が隙間なくオブジェクトで埋め尽くされるような表示を想定してパラメータ設定を行った。

4.2.2 既存の Flex コンポーネントの Web 標準技術への移行に向けた性能の見積

前項で測定可能な値は「描画処理」に必要な時間のみであり，オブジェクトの座標計算やコンポーネント固有の計算ロジック等の処理時間は含まれていない。UI コンポーネントでは描画処理以外にもこれらの計算が必要となるため，

移行の際には全体の処理時間を含んだ性能見積が必要である。そこで，ソースコードが存在する Flex コンポーネントを例に，Web 標準技術に準拠したコンポーネントに移植する際の性能見積方法について提案する。性能測定では，Flex で構築された既存のガントチャートコンポーネントを題材として性能見積を行い，移植可能性について検討する。

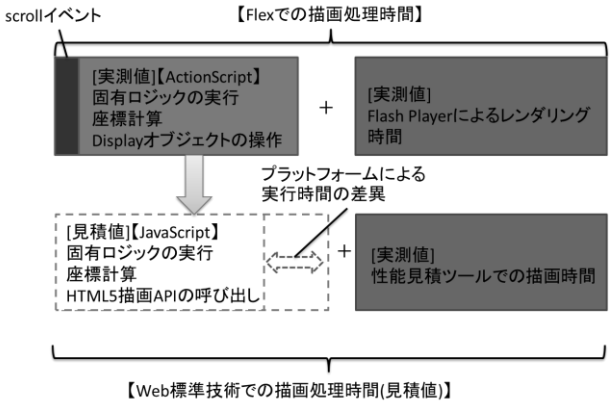

図 3　Flex コンポーネントの
実行時間に基づく性能見積

Flex コンポーネントは保持モードで描画が行われており，図 3 のように scroll イベントなどの画面更新が必要なイベントを起点としコンポーネント固有のロジックの実行や座標計算が行われ，オブジェクトモデルの操作が行われる。その後，Flash Player によるレンダリング処理が行われ，画面が更新される。Flex コンポーネントのソースコードが存在する場合，ソースコードの中に計測コードを埋め込むことで ActionScript によるこれらの処理に必要な時間と Flash Player によるレンダリング時間を個別に取得することができる。一方，同等の内容を含む画面を Web 標準技術を用いて描画する時間は 4.1 節の性能測定ツールにより実測値が得られる。つまり，図 3 のように ActionScript の処理にかかる時間を擬似的に JavaScript による処理時間とみなし，性能測定ツールにより得られた実測値を加算することで，Web 標準技術で構築したコンポーネントの処理時間を見積もることができる。ActionScript と JavaScript の実行時間に差異があることが想定できるが，近年の JavaScript 実行エンジンの高速化も考慮し，見積値の算出においては JavaScript の実行時間が ActionScript の実行時間と比較し 1.0 倍(等倍)～2.0 倍程度遅いと想定して係数がけする。

4.3 目標性能と測定環境

目標性能はユーザがシステムの反応が瞬時に行われていると感じる限界値[12]とされる 100ms と設定した。

表 4　性能測定環境

	PC	iPad3
OS	Windows 8.1 Pro	iOS 9.3.5
CPU	Intel Core i7 4770 3.4GHz	Apple A5X 1GHz
メモリ	8GB	1GB
GPU	Intel HD Graphics 4600	PowerVR SGX543MP4

測定は PC とスマートデバイスを想定し表 4 に示す環境で実施した。測定は各 100 回行い，平均と分散値を取得した。Node.js によるサーバサイドの性能測定も表 4 の PC の環境で行った。

5. 測定結果による移行時の性能見積

5.1 実アプリケーションを想定した描画性能見積

表 5　実アプリケーションを想定した描画性能

環境	描画技術	計測値[ms] 表計算	ガント小日程	ガント大日程	グラフ一覧
PC IE11	Canvas	16.09 (4.78)	18.65 (9.13)	140.63 (67.45)	41.56 (40.95)
	SVG	15.57 (9.11)	15.28 (4.32)	112.86 (122.40)	26.47 (9.74)
	WebGL	15.18 (2.89)	14.49 (1.12)	14.99 (2.27)	15.27 (2.58)
PC Chrome	Canvas	17.63 (9.45)	17.45 (5.19)	56.83 (43.80)	29.67 (25.74)
	SVG	16.67 (0.22)	16.66 (0.48)	31.17 (22.94)	26.54 (13.89)
	WebGL	16.73 (2.71)	16.61 (1.34)	16.83 (5.64)	16.99 (3.47)
PC Firefox	Canvas	26.51 (8.33)	19.59 (6.08)	132.92 (153.55)	42.17 (22.05)
	SVG	16.76 (2.34)	16.74 (11.17)	18.35 (278.29)	17.03 (26.97)
	WebGL	16.89 (8.72)	16.74 (3.11)	16.87 (13.89)	16.58 (6.85)
iPad3 Safari	Canvas	151.73 (54.24)	223.30 (109.15)	1218.45 (59.31)	1199.11 (239.96)
	SVG	231.83 (27.52)	250.16 (2.93)	1702.26 (591.85)	457.69 (1570.99)
	WebGL	25.72 (58.10)	18.72 (42.94)	52.96 (72.12)	17.92 (30.75)
PC Node.js	Canvas	23.52 (3.07)	14.54 (34.65)	115.76 (43.60)	103.85 (28.63)

下段括弧内：分散値

実際の Web UI コンポーネントを想定した性能測定結果を表 5 に示す。性能目標である 100ms を超過したものについては背景をハイライトして示す。PC の Web ブラウザにおいてはガントチャート大日程モデル以外で，描画技術によらず性能目標の半分以下の所要時間で描画を完了できる見通しを得た。コンポーネント固有の処理時間が描画時間と同等の所要時間と仮定しても性能目標以下に収まるため，Web 標準技術への移行が可能であると見積もることができる。一方，ガントチャート大日程モデルでは Chrome 以外では目標性能が達成できない見通しを得た。画面全体を満遍なくオブジェクトが覆い尽くすような場合，性能問題が発生する可能性がある。iPad3 では，全てのアプリケーションモデルで WebGL 以外は大幅な性能未達となった。スマートデバイス対応においては，サーバサイドレンダリン

グの活用などのアーキテクチャを検討する必要があるといえる。また，WebGL は全てのパターンにおいて目標性能を達成した。ハードウェア要件および各環境での互換性に関する動作保障の問題を除けば，有効な選択肢となり得る。

5.2 既存の Flex コンポーネントの Web 標準技術への移行に向けた性能見積

Flex 版ガントチャートコンポーネントにおける処理時間を表 6 に示す。また，表 5 で取得した各描画技術の描画時間に対し，Flex のコード実行時間を係数がけのうえ加算することで算出した見積値を表 7 に示す。5.1 節同様に，描画時間および見積値が性能目標である 100ms を超過したものについては背景をハイライトして示す。なお，サーバサイドの見積値は Node.js と同一の JavaScript 実行エンジンを搭載する Chrome のコード実行時間を用いて算出した。

表 6　Flex による UI コンポーネントの実行時間

モデル	環境	計測値[ms]		
		コード実行 ActionScript	レンダリング Flash Player	合計
小日程	IE11	41.55 (120.65)	45.50 (69.85)	87.05 (161.25)
	Chrome	45.14 (55.38)	43.73 (81.60)	88.87 (125.69)
	Firefox	35.14 (75.26)	38.44 (80.21)	73.58 (156.80)
大日程	IE11	215.78 (1954.83)	292.92 (909.23)	508.70 (3475.59)
	Chrome	213.36 (544.77)	289.94 (649.34)	503.30 (1364.13)
	Firefox	180.65 (970.11)	258.32 (421.52)	438.97 (1598.55)

下段括弧内：分散値

表 7　Web 標準技術への移行時の描画時間見積値
（表 5 の計測値＋表 6 のコード実行時間×係数）

環境	描画技術	見積値[ms]			
		小日程モデル		大日程モデル	
		係数 1.0	2.0	1.0	2.0
IE11	Canvas	60.20	101.75	356.41	572.19
	SVG	56.83	98.38	328.64	544.42
	WebGL	56.04	97.59	230.77	446.55
Chrome	Canvas	61.95	107.09	270.19	483.55
	SVG	61.80	106.94	244.53	457.89
	WebGL	61.75	106.89	230.19	443.55
Firefox	Canvas	54.73	89.87	313.57	494.22
	SVG	51.88	87.02	199.00	379.65
	WebGL	51.88	87.02	197.52	378.17
Node.js	Canvas	59.68	104.82	329.12	542.48

小日程モデルでは，ユーザコードの実行時間に対する係数を 2.0 とした場合でも目標性能+10ms に収まる値が得られた。また，Canvas と SVG では性能上の差は 4ms 未満であり体感可能な程度の大きな差はないことが分かった。目標値は満たしていないが，目標値と比較しても 10ms 程度の超過であり操作に若干のもたつきを感じる程度となると推測できる。そのため，全ての描画技術においてガントチャートは実現可能であるといえる。

大日程モデルでは全ての場合において目標性能を大幅に超過している。Flex 版ガントチャートコンポーネントの描画性能が許容できるのであれば，Web 標準技術によるコンポーネントへ移行しても問題は発生しないといえる。

サーバサイドにおいては，クライアントで描画する場合と同等の処理時間となっている。ネットワークのオーバヘッドが存在するが，処理性能の低いクライアントに対しても PC と同等の待ち時間で画面が表示できることが示された。以上より，ガントチャートを想定した Flex コンポーネントの Web 標準技術への移行については， Canvas または SVG を選択した場合であっても，描画性能面での著しい性能劣化は発生しないと考えられ，移行可能と判断できる。

6. おわりに

Flex と比較し描画性能が低いとされる HTML5 では，複数の描画技術が存在するが具体的な性能が不明であり，多数のオブジェクトの描画を必要とするコンポーネントの性能見積ができない課題があった。これらの課題に対し，汎用的に活用可能な性能測定ツールを試作し，画面内に表示されるオブジェクトの種類と数を元にした性能見積を可能とした。さらに，描画技術ごとに性能を比較可能とし選択すべき描画技術の決定を容易にした。また，Flex コンポーネントを Web 標準技術に移行した際の性能を見積もる方法を示し，描画性能面における Web 標準技術への移植可能性を早期に明確化することが可能となった。

参考文献

[1] Extending User Control of Flash with Click-to-Run, https://blogs.windows.com/msedgedev/2016/12/14/edge-flash-click-run/#TAWwIWpwTXPYOCih.97, Accessed 2017/3
[2] The Chromium Projects Flash Roadmap, https://sites.google.com/a/chromium.org/dev/flash-roadmap, Accessed 2017/3
[3] HTML5, http://www.w3.org/TR/html5/, Accessed 2017/3
[4] 山本 愛佑子, 渡辺 裕, "HTML5 Canvas および SVG における自動選択アルゴリズムを用いた描画パフォーマンスの最適化",オーディオビジュアル複合情報処理（AVM） 2014-AVM-84(6), pp.1-2, 2014
[5] W3C HTML Canvas 2D Context, http://www.w3.org/TR/2dcontext/, Accessed 2017/3
[6] WebGL Specification, https://www.khronos.org/registry/webgl/specs/latest/1.0/, Accessed 2017/3
[7] GDI, https://msdn.microsoft.com/ja-jp/library/windows/desktop/dd145203(v=vs.85).aspx, Accessed 2017/3
[8] SVG と Canvas: どちらを選ぶか, https://msdn.microsoft.com/ja-jp/library/gg193983(v=vs.85).aspx, Accessed 2017/3
[9] W3C Scalable Vector Graphics (SVG), http://www.w3.org/Graphics/SVG/, Accessed 2017/3
[10] 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", 情報科学技術フォーラム講演論文集 14(4), pp.77-84, 2015
[11] Node.js, https://nodejs.org/ja/, Accessed 2017/3
[12] ヤコブ・ニールセン，"ユーザビリティエンジニアリング原論"，東京電機大学出版局，p.107, 2002