Server-Sent Eventsを用いたWebSocketエミュレーションフレームワークの設計と実装

早川 智一[1,a)]　西角 知佳[2]　小泉 修一[1,b)]　疋田 輝雄[1,c)]

概要：WebSocket と Server-Sent Events はネットワークに関する HTML5 の新機能であり，一般に WebSocket の方が Server-Sent Events よりも用いられているようである．ところが，Server-Sent Events がほとんどの環境で動作する一方で，WebSocket はすべての環境で動作するわけではないという課題がある．そこで本論文では，Server-Sent Events と関連技術を用いて WebSocket を模倣するフレームワークを提案する．このフレームワークを用いることで，Web 開発者はサーバ・クライアント間で双方向の通信が必要な Web アプリケーションを WebSocket API に対して設計・実装することが可能になり，必要に応じて実際の通信方法（WebSocket か Server-Sent Events か）を既存のソースコードをほぼ修正することなく透過的に変更することができる．我々は，提案フレームワークを用いた場合の平均応答時間を生の WebSocket と比較し，提案フレームワークが移植性の高い双方向 Web アプリケーションを開発する Web 開発者の一助たりえるという結論を得た．

1. はじめに

W3C（World Wide Web Consortium）による HTML5 [6] の正式勧告を受けて，ネットワークに関する HTML5 の新機能——WebSocket [9] と Server-Sent Events [7]——が注目されている．WebSocket は TCP/IP に基づく双方向の通信チャネルで，サーバ・クライアント間でのリアルタイムでの通信を可能にする．Server-Sent Events は HTTP に基づく一方向の通信チャネルで，サーバからクライアントへのリアルタイムでの通信を可能にする．

この 2 つの新機能を比較すると，WebSocket の方が Server-Sent Events よりも一般に用いられているようである．我々は，この理由を，WebSocket の方が Server-Sent Events よりも高速で高機能な通信機能を提供するためと考える．

しかしながら，WebSocket はすべての環境で使用できるわけではない．この理由として，以下のことが挙げられる：

(1) WebSocket は HTTP ではなく TCP/IP に基づく新プロトコルであり，ルータ・プロキシ・ファイアウォールなどが対応していない場合があるため．

(2) WebSocket はその性質から自身のサーバプロセスを動作させるために専用のサーバを必要とするため，レンタル Web サーバ上で WebSocket を使うことが困難であるため [*1]．

一方で，Server-Sent Events は HTTP に基づいているため，ほとんどの環境で使用できる．具体的には，Server-Sent Events はほとんどのルータ・プロキシ・ファイアウォールなどが対応しており，CGI（Common Gateway Interface）[*2] が動作する共有 Web サーバ上でも使用できる．

WebSocket と Server-Sent Events のこれらの利点・欠点によって，Web 開発者はある種の課題を抱えることになる——サーバ・クライアント間で双方向の通信が必要な Web アプリケーションはどちらの機能を使って実装すべきか；理想的には WebSocket を使うことが望ましい場合が多いはずだが，環境的な要因で WebSocket が動作しない場合には Server-Sent Events を（他の技術と組み合わせて）使うべきか．どちらの機能を選択しても，Web 開発者は，開発した Web アプリケーションの通信機能を一方からもう一方へと変更する必要が生じた場合には，当該 Web アプリケーションを書き直さざるを得なくなる．

この課題を解決するために，我々は WebSocket を透

[1] 明治大学理工学部情報科学科
School of Science and Technology, Meiji University, Kawasaki, Kanagawa, 214–8571, Japan
[2] 株式会社ラクス
RAKUS Co., Ltd.
a) t_haya@cs.meiji.ac.jp
b) s-koizumi@cs.meiji.ac.jp
c) hikita@cs.meiji.ac.jp

[*1] 一般に，共有 Web サーバでは，プロセスをデーモンとして常駐動作させることが禁止されているため．
[*2] CGI の実装言語は何でも良いが，例として Perl・PHP・Python などが挙げられる．

©2015 Information Processing Society of Japan

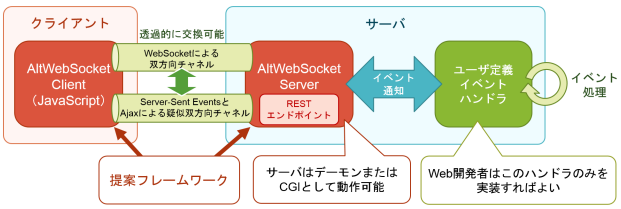

図 1　提案フレームワークの概要
Fig. 1　Overview of our proposed framework.

過的に模倣するフレームワークを提案する．提案フレームワークは，Server-Sent Events を Ajax（Asynchronous JavaScript + XML）と REST（REpresentational State Transfer）と組み合わせて擬似的な双方向通信を実現する．提案フレームワークを使うことで，Web 開発者は以下のような利点を得る：

(1) Web 開発者は，（実際の通信が WebSocket か Server-Sent Events かを気にすることなく）Web アプリケーションを一貫して WebSocket API（Application Programming Interface）に基づいて設計できる．
(2) Web 開発者は，必要に応じて透過的に通信手段（WebSocket か Server-Sent Events か）を変更できる．

これにより，環境的な要因で WebSocket が使えない状況下では Server-Sent Events で通信し，環境の変化で WebSocket が使えるようになった場合には即座に移行することができる．

本論文の構成は次のとおりである．2 節と 3 節では提案フレームワークの設計と実装を概説する．4 節では予備評価の結果を報告する．5 節では関連研究を紹介する．6 節ではまとめと今後の予定を述べる．

2. 設計

2.1 概要

図 1 に，提案フレームワークの概要を示す．フレームワークは，サーバ側とクライアント側とで構成される．このフレームワークが通信手段（WebSocket か Server-Sent Events か）を隠蔽するため，Web 開発者は実際の通信手段を意識することなく設計・実装を行うことができる．このフレームワークを使うことで，Web 開発者は通信手段がどちらの場合でも，作成するアプリケーションごとに「ユーザ定義イベントハンドラ」を実装するだけでサーバ機能を実現できる．「ユーザ定義イベントハンドラ」は WebSocket イベントを処理するためのハンドラで，`AltWebSocketServer` から呼び出される．クライアント側のフレームワークは JavaScript で記述されている．これは，JavaScript が Web ブラウザ上で動く事実上の唯一のプログラミング言語であるためである．一方で，サーバ側のフレームワークは CGI として動作するプログラミング言語であれば何でもよい（本論文中のプロトタイプでは PHP を使用している）．

```
/* 第二引数でエミュレーションするかどうかを指定する */
var ws = new AltWebSocket("Server URL", true);
ws.addEventListener('open', function(event){ ws.send('Hello, world!'); });
ws.addEventListener('message', function(event){
  console.log('server said: ' + event.data);
  ws.close();
});
ws.addEventListener('close', function(event){ console.log('closed'); });
ws.addEventListener('error', function(event){ console.log('error'); });
```

図 2　提案フレームワークの使用例（クライアント側）
Fig. 2　Sample usage of our proposed framework (client side).

```
class UserDefinedEventHandler implements AltWebSocketHandler {
    private $clients;
    public function __construct() {
        $this->clients = AltWebSocketStorage.getInstance();
    }
    public function onOpen(AltWebSocketConnection $con) {
        $this->attach($con); /* クライアントを追加する */
    }
    public function onMessage(AltWebSocketConnection $from, $msg) {
        foreach ($this->clients as $client) {
            /* 全クライアントに受信メッセージを送信する */
            $client->send($msg);
        }
    }
    public function onClose(AltWebSocketConnection $con) {
        $this->detach($con); /* クライアントを削除する */
    }
}
```

図 3　提案フレームワークの使用例（サーバ側）
Fig. 3　Sample usage of our proposed framework (server side).

2.2 使用例

図 2 と図 3 に，提案フレームワークの使用例を示す．図 2 より，クライアント側フレームワークは WebSocket と同じ API を持つことが分かる．ここで，`AltWebSocket` のコンストラクタの第二引数に `true` を渡した場合には，提案フレームワークは Server-Sent Events とその他の技術を用いて WebSocket の動作を模倣する；`false` を渡した場合には，提案フレームワークはブラウザに実装されている生の `WebSocket` を用いる．図 3 より，サーバ側フレームワークも WebSocket と同じ API を持つことが分かる．以上のことから，提案フレームワークを使うことで，Web 開発者は実際の通信手段（WebSocket か Server-Sent Events か）を気にすることなく WebSocket の API に対して Web アプリケーションを設計・実装できることが分かる．

2.3 クラス設計

図 4 と図 5 に，提案フレームワークのクラス図を示す．

図 4 より，`AltWebSocket` が 2 つのクラス（生の `WebSocket` と提案フレームワークの `EventSocket`）を持つことが分かる．WebSocket の模倣が不要な場合には生の `WebSocket` を使い，必要な場合には `EventSocket` を用いる．`EventSocket` は 2 つのクラス（生の `EventSource` と `XMLHttpRequest`）を持つ．`EventSocket` は，WebSocket の双方向チャネルを模倣するために，`XMLHttpRequest` を用いてデータをサーバに送信する．これは，Server-Sent Events（`EventSource`）だけでは，サーバからクライアントへの一方向チャネルしか実現できないためである．

図 5 より，`AltWebSocketServer` が WebSocket のサーバ機能を抽象化していることが分かる．このクラスは，Web-

©2015 Information Processing Society of Japan

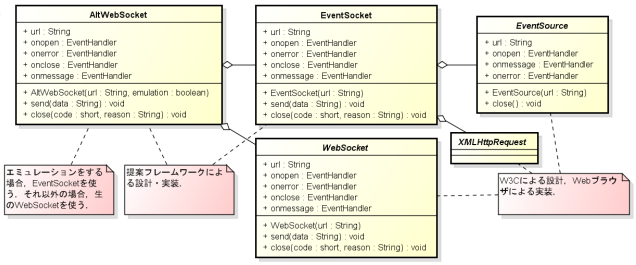

図 4　提案フレームワークのクラス図（クライアント側）
Fig. 4　Class diagram of client-side framework.

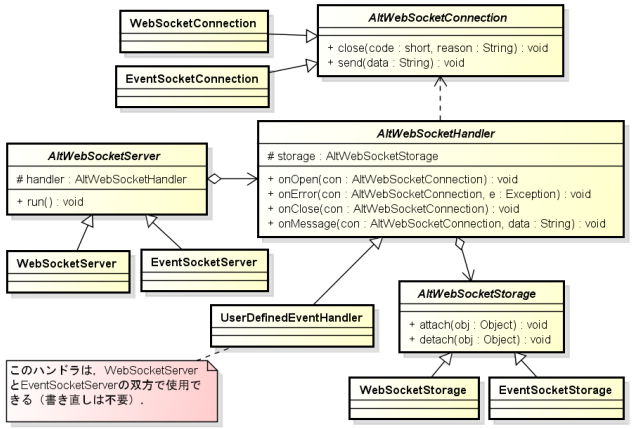

図 5　提案フレームワークのクラス図（サーバ側）
Fig. 5　Class diagram of server-side framework.

Socket イベントを処理するための `AltWebSocketHandler` を持つ．`AltWebSocketConnection` は，クライアントからの接続を表す．`AltWebSocketStorage` は，アプリケーションのデータを永続化するためのストレージを表す．Web 開発者は，`AltWebSocketHandler` から派生する `UserDefiendEventHandler` を実装するだけでサーバ機能を実現できる．

2.4　REST エンドポイント

表 1 に，提案フレームワークの主要な REST エンドポイントを示す．これらのエンドポイントは，`EventSocket` が WebSocket を模倣するために使用する．エンドポイント ‘/’ は `EventSource` の接続先であり，サーバからクライアントへの通信チャネルを提供する．エンドポイント ‘/new’ は，新しい接続を作成するために使用する．エンドポイント ‘/send’ は，メッセージの送信に使用する．エンドポイント ‘/close’ は，既存の接続を閉じるために使用する．

3.　実装

表 2 に，我々がプロトタイプの実装に用いたソフトウェアを示す．クライアント側フレームワークの実装には，Ajax フレームワークとして jQuery [1] を用いた．サーバ側フレームワークでは，REST エンドポイントの実装に Slim [2] を使用し，WebSocket のライブラリとして Ratchet [4] を使用した．

4.　予備評価

予備評価として，ギガビット・イーサネット（GbE）で接続したサーバ・クライアント間で提案フレームワークを用いた際の平均応答時間を計測した．実験手順は次のとおりである：

(1) 提案フレームワークを用いて echo サーバ（受け取ったメッセージをオウム返しするサーバ）を実装する．
(2) クライアントはサーバにデータを送信する．
(3) サーバはクライアントから受信したデータをクライアントに送り返す．
(4) クライアントはサーバからデータを受信し次第，別のデータをサーバに送信する．
(5) サーバとクライアントは（3）と（4）の手順を規定の回数繰り返す．

なお，この予備評価では，提案フレームワークが WebSocket を模倣する際の最悪の場合のオーバヘッドを測定するために，送信するデータは非常に小さいランダムデータ（数バイト程度）とした [*3]．

表 3 に計測結果を示す．表より，提案フレームワークが採用している Server-Sent Events + Ajax による模倣動作時の応答速度は，生の WebSocket と比較して約 25 倍遅いことが分かる．これは，WebSocket に比べて，Server-Sent Events と Ajax による HTTP 通信のオーバヘッドが大きいためである．一方で，提案フレームワークによる WebSocket の模倣時の応答速度も決して遅くはないと我々は考える．提案フレームワークを用いた場合でもサーバ・クライアント間で一往復あたり約 15 ミリ秒での応答を実現しており，実装する Web アプリケーションの種類（例：チャットなど）によっては十分な応答速度である [*4]．

5.　関連研究

Skvorc [3] らは，WebSocket は通信開始時のハンドシェークのオーバヘッドを除けば，TCP/IP と比較して遜色ない性能であると述べている．このことは，表 3 の WebSocket の 100 回試行時の平均応答時間が 1,000 回や 10,000 回試行時の平均応答時間と比較して遅かったという結果とも合致する．

SockJS [5] は Web 関連技術を用いて WebSocket を模倣するフレームワークである．SockJS と提案フレームワークは，他の Web 技術を用いて WebSocket を模倣する点に類似性がある．一方で，SockJS が CGI として動作することを前提とせず専用サーバを必要とするのに対して，提案

[*3] 送信するデータの大小や特性（テキストかバイナリか等）によっても応答時間は変化すると考えられるため，詳細な評価実験が今後の課題である．
[*4] GbE 接続した LAN 環境下での測定値なので，外部の Web サーバを用いての詳細な評価実験が今後の課題である．

©2015 Information Processing Society of Japan

表 1　提案フレームワークの REST エンドポイント（抜粋）
Table 1　List of REST endpoints of our proposed framework.

エンドポイント	メソッド	リクエスト MIME	レスポンス MIME	備考
/	GET	（空）	text/event-stream	EventSource ターゲット
/new	POST	（空）	application/json	新規接続の作成
/send	POST	application/json	application/json	メッセージの送信
/close	POST	application/json	application/json	接続を閉じる

表 2　プロトタイプの実装に用いたソフトウェア
Table 2　Used software for prototype implementation.

ソフトウェア	バージョン	備考
jQuery	2.1.4	Ajax フレームワーク
Slim	2.6.2	PHP マイクロフレームワーク
Ratchet	0.3.3	PHP 用 WebSocket フレームワーク
PHP	5.6.11	プログラミング言語

表 3　サーバ・クライアント間の平均応答時間（ミリ秒）
Table 3　Average response times (ms) between server and client.

試行回数	100	1,000	10,000
WebSocket	110	654	6,145
Server-Sent Events + Ajax	2,655	16,524	152,115

フレームワークは CGI としても動作し共用サーバでも動作する点で異なる．

6. おわりに

本論文では，Server-Sent Events とその他の技術を用いて WebSocket の動作を模倣するフレームワークを提案した．予備評価の結果は，提案フレームワークによる WebSocket 模倣時の速度が生の WebSocket よりは遅いものの，作成する Web アプリケーションの種類によっては実用の範囲であることを示した．この結果から，提案フレームワークが，移植性の高い双方向 Web アプリケーションを開発する Web 開発者にとって一助となるという結論を我々は得た．

今後の予定としては，フレームワークの詳細な評価と拡充改善を行う予定である．具体的には，送受信するデータ量や特性（テキストかバイナリか等）を変化させたり，様々な Web ブラウザを用いての評価を行う予定である．また，スループットを向上させるために，Web Workers [8] を用いた通信の非同期化・並列化なども検討する予定である．

参考文献

[1] jQuery Foundation: jQuery: The Write Less, Do More, JavaScript Library, jQuery Foundation (online), available from ⟨http://jquery.com/⟩ (accessed 2015-07-31).

[2] Lockhart, J.: Slim Framework, Josh Lockhart (online), available from ⟨http://www.slimframework.com/⟩ (accessed 2015-07-31).

[3] Skvorc, D., Horvat, M. and Srbljic, S.: Performance evaluation of Websocket protocol for implementation of full-duplex web streams, *Proc. of 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, IEEE Press, pp. 1003–1008 (2014).

[4] socketo.me: Ratchet - PHP WebSockets, socketo.me (online), available from ⟨http://socketo.me/⟩ (accessed 2015-07-31).

[5] sockjs.org: SockJS - WebSocket emulation, sockjs.org (online), available from ⟨http://sockjs.org/⟩ (accessed 2015-07-31).

[6] W3C: HTML5, W3C (online), available from ⟨http://www.w3.org/TR/html5/⟩ (accessed 2015-07-31).

[7] W3C: Server-Sent Events, W3C (online), available from ⟨http://www.w3.org/TR/eventsource/⟩ (accessed 2015-07-31).

[8] W3C: Web Workers, W3C (online), available from ⟨http://www.w3.org/TR/workers/⟩ (accessed 2015-07-31).

[9] W3C: The WebSocket API, W3C (online), available from ⟨http://www.w3.org/TR/websockets/⟩ (accessed 2015-07-31).

©2015 Information Processing Society of Japan