情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.2　No.2　19–28 (July 2012)

コンシューマ・システム論文

Webと機器を透過的につなぐ Multimodal Interactionフレームワークの実装

芦村 和幸[1,2,a)]　小松 健作[2,3]　一色 正男[1,4]

受付日 2011年12月19日, 採録日 2012年4月13日

概要：Web は世界中で共通的に利用されるがゆえに，その相互運用性，国際化，入出力方法の多様性，および利用者への親和性を保証する必要があり，そのために，W3C [1] 等による国際的な技術標準化が現在も行われているが，時代とともに Web で扱うコンテンツは多様化し，マルチメディア系コンテンツの需要が飛躍的に増加してきたため，従来の方法で表現することが難しくなってきている．開発者はこのような Web 利用の進化に合わせるため，独自の手法を使って対応するようになっていったが，それがコンテンツ普及の妨げになりつつある．本稿では，標準的なフレームワークに沿ったソフトウェアを Web アプリケーション開発者に提供することで，開発者の力量に依存することなくコンテンツの多様化に対応する方法として，W3C MMI アーキテクチャ [2], [3], [4] を利用した標準化フレームワークに沿ったライブラリの実装方法について論ずる．また，実際に上記ライブラリを実装したうえで評価を行うとともに，開発ソフトウェア標準化の現状について述べる．

キーワード：Web，標準化，コンテンツ多様化，W3C MMI アーキテクチャ，MMI over WebSocket

Implementing Multimodal Interaction Framework for Transparent and Smarter Integration of the Web and CE Devices

Kazuyuki Ashimura[1,2,a)]　Kensaku Komatsu[2,3]　Masao Isshiki[1,4]

Received: December 19, 2011, Accepted: April 13, 2012

Abstract: The Web is used globally and commonly, so Interoperability, Internationalization, Multi-Modality and Accessibility are essential. That's why W3C [1] has been tackling international standardization for Web technologies. However, Web contents have been changed much and multimedia contents are getting more and more populer these days. So it is getting more and more difficult to render the contents using conventional ways. Therefore, this paper proposes a standard library based on the W3C MMI Architecture [2], [3], [4] so that developers can handle variety of Web contents regardless of their skills. Also, this paper discusses how to implement the library and evaluate it, and then explain the current status of standardization of the library.

Keywords: the Web, standardization, variety of Web contents, W3C MMI Architecture, MMI over WebSocket

[1] W3C/慶應（慶應義塾大学 SFC 研究所）
W3C/Keio (Keio Research Institute at SFC), Fujisawa, Kanagawa 252–0882, Japan
[2] W3C マルチモーダル対話ワーキンググループ
W3C Multimodal Interaction Working Group, Fujisawa, Kanagawa 252–0882, Japan
[3] NTT コミュニケーションズ株式会社
NTT Communications Corporation, Minato, Tokyo 108–8118, Japan
[4] 神奈川工科大学
Kanagawa Institute of Technology, Atsugi, Kanagawa 243–0292, Japan
[a)] ashimura@w3.org

1. はじめに

World Wide Web Consortium（W3C）[1] は，Web の可能性を最大限に導き出すことを目的として，Web 技術発明者である Tim Berners-Lee により創設された産業コンソーシアムであり，Web の発展と相互運用性確保に必要な各種プロトコルの開発に取り組んでいる．Web は世界中で共通的に利用されるがゆえに，その相互運用性（Interoperability），国際化（Internationalization），入出力方

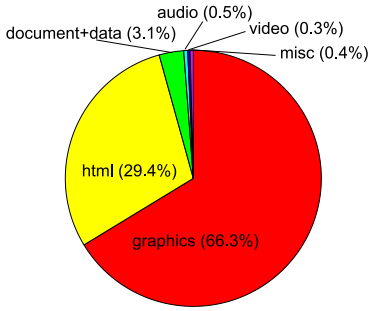

図 1　総ファイル数のファイル種類別シェア
Fig. 1　File number ratio per media type.

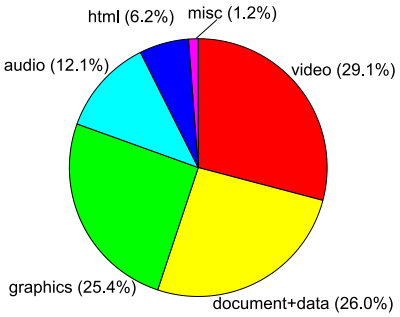

図 2　総データ量のファイル種類別シェア
Fig. 2　Data amount ratio per media type.

法の多様性（Multi-Modality），および利用者への親和性（Accessibility）を保証する必要があり，そのために，W3C 等による国際的な技術標準化が現在も行われている．その Web へのアクセス方法として Web ブラウザが開発され，各種端末上の表示装置およびブラウザ・ソフトウェアの差分を吸収するため，そして開発者が容易に Web アプリケーションの開発を行うことができるように，Hypertext Markup Language（HTML）[5], [6] が標準化され利用されてきた．しかし，時代とともに Web で扱うコンテンツは多様化し，マルチメディア系コンテンツの需要が飛躍的に増加してきたため，従来の方法で表現することが難しくなってきている．たとえば，総務省による 2004 年の「WWW コンテンツ統計調査報告書」[7] によれば，JP ドメインを持つ Web サーバ上の情報においては，**図 1** に示すとおり，静止画（graphics）のファイル数が 66.3%で最も多い．また，**図 2** に示すとおり，データ量では動画（video）が 29.1%で最も多く，静止画（graphics; 25.4%），音声（audio; 12.1%）を含めたマルチメディア系コンテンツ全体では 66.6%を占めるに至っている．

開発者はこのような Web 利用の進化に合わせるため，独自の手法を使って対応するようになっていったが，それがコンテンツ普及の妨げになりつつある．本稿では，標準的なフレームワークに沿ったソフトウェアを Web アプリケーション開発者に提供することで，開発者の力量に依存することなくコンテンツの多様化に対応する方法として，W3C MMI アーキテクチャ [2], [3], [4] を利用した標準化フレームワークに沿ったライブラリの実装方法について論ずる．また，実際に上記ライブラリを実装したうえで評価を行うとともに，開発ソフトウェア標準化の現状について述べる．以下，2 章ではまず関連する現状の技術とその課題について述べる．3 章ではその課題を解決する提案手法について述べ，4 章ではその提案手法の実装方法について述べる．そのうえで，5 章では実装された提案手法の評価を行うとともに，6 章では本提案手法の標準化について述べ，最後に 7 章でまとめを述べる．

2. 関連技術

2.1 HTML + プラグイン

Web 上の文書記述のためには，主に HTML [5], [6] が利用されるが，近年，従来のテキスト情報に加えて音声・動画等，より高度なマルチメディア情報に対するニーズが高まっており，プラグインと呼ばれる Web ブラウザの拡張機能が開発されてきた．しかしながら，各種プラグイン・ソフトウェアの具体的実装内容が開発元ごとに異なることから標準化が難しく，開発者および利用者への負担が増加していた．たとえば，音声・動画等のマルチメディア情報を含む Web ページへアクセスした際には，各マルチメディア情報を再生するための適切なプラグインが必要となるが，必ずしも利用者の使用する機器上に必要なプラグインがインストールされているとは限らず，プラグインがインストールされていない場合，期待したマルチメディア情報の再生が行われることなく「プラグインがインストールされていません」等の警告が表示されることになる．また，仮にプラグインがインストールされていても，そのプラグインとブラウザや OS 等とのバージョン不整合により動作しないこともあり，極端な場合には，プラグインを利用する Web ページにアクセスした瞬間にブラウザがエラー終了する可能性もある．

また，一般的な PC と，携帯電話等のモバイルデバイスでは，画面解像度や CPU 処理能力等に違いがあるため，モバイルデバイスでは PC よりも小さな画像を使用したり，画像をテキスト情報に置き換えたりする等，機器ごとに異なるデザインの Web ページを制作する必要があり，制作コストの高騰を招いていた．

2.2 HTML5

上記で触れたようなプラグイン利用等の問題に起因する，開発者および利用者の負担を軽減するとともに，より表現力豊かな Web アプリケーションを実現するための標準的プラットフォームとして HTML5 [8], [9] が提案されている．HTML5 には以下の機能が追加されており，従前の HTML が持つ基本的な GUI 機能に加え，音声・動画等の動的な情報や Web と利用者との相互的なやりとりに対応したものとなっているため，様々な Web アプリケーショ

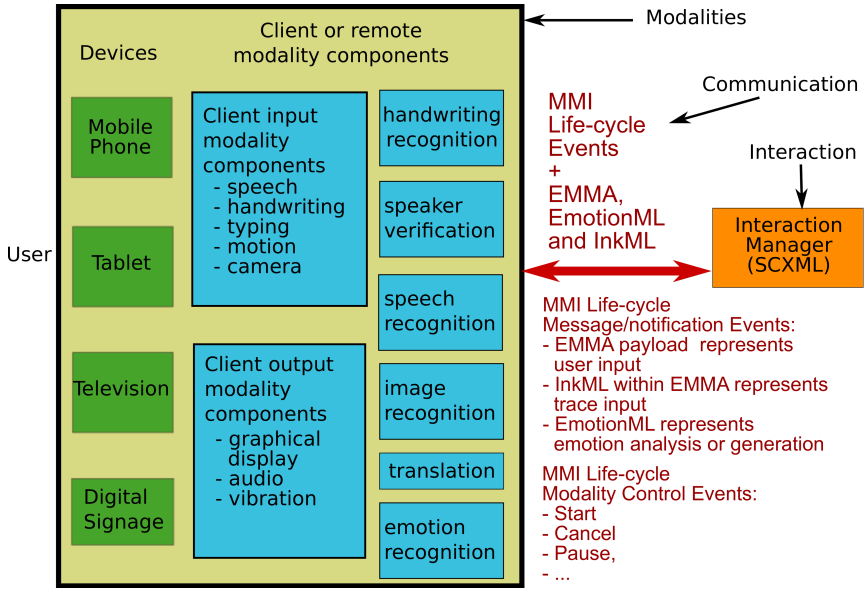

図 3　W3C MMI アーキテクチャ
Fig. 3　W3C MMI Architecture.

ンでの利用が期待されている．

- Video [10] および Audio [11]：プラグインを使わずに動画や音声を再生
- Canvas [10]：二次元画像描画
- Drag & Drop [10]：画像およびテキストの移動・編集
- WebSocket [10], [12], [13]：Web サーバを経由しない，双方向かつ全二重通信
- Web Storage [10]：ローカルクライアント上の Web ブラウザ内でデータ保存
- Web Workers [10]：Web ブラウザ上でのマルチプロセス

しかし，HTML5 を利用した場合においても，様々な機器を実際に組み合わせて利用する際の入出力モダリティ拡張は Web アプリケーション開発者が個々に対応する必要があり，開発者の負担を低減させるためには，個々の開発者の能力に依存しない標準化を前提とした Web アプリケーション開発のフレームワークが必要となる．

3. 提案手法：MMI over WebSocket ライブラリ

3.1 標準化ライブラリ開発の必要性

個々の Web アプリケーション開発者の能力に左右されにくいフレームワークを提供するためには，標準化を前提とした開発方法の提供が必要であり，そのためには，アプリケーションに必要な入出力モダリティを動的に選択できるとともに，各モダリティがどの機器に実装されているかを意識することなく透過的に組み合わせて利用できるような，標準化されたフレームワークに沿った「標準ライブラリ」の開発が必要とされる．そこで本稿では，標準化を前提としたフレームワークの開発方法について論ずる．その際，標準化の基本的アーキテクチャとして，国際的標準化団体である W3C が提唱している Multimodal Architecture and Interfaces（以下，MMI アーキテクチャ）[2] に沿った形で標準化ライブラリの開発を試みる．

ここで，W3C の提唱する MMI アーキテクチャとは，個々のアプリケーションに応じて様々な入出力モダリティを動的に選択するとともに，各モダリティがどの機器上に実装されているのかを意識することなく透過的に組み合わせて利用することを可能とする，統一的なメッセージ交換の仕組みである．**図 3** に示すとおり，MMI アーキテクチャでは，通常の GUI に加え音声や手書き入力等の様々なモダリティを動的に組み合わせて利用することが想定されており，W3C で公開されている MMI アーキテクチャ仕様書 [2] により，(1) モダリティ・コンポーネント（クライアント側；以下，MC）とインタラクション・マネージャ（サーバ側；以下，IM）という 2 種類の構成要素，および (2) 各構成要素間で相互に情報を交換するための通信プロトコル（イベントおよびデータ形式）が規定されている．

なお，実際に開発者が，HTML5 を応用したマルチモーダル Web アプリケーションを実装する際には，機能の増大にともなう動作速度の改善も必要となってくる．そこで本稿では，MC と IM の間でやりとりされる通信プロトコルを，一般に利用されている HTTP から WebSocket に変更することで速度の改善を試みる．また，速度改善の程度を把握するために HTTP による通信と WebSocket による通信の比較を行った．その結果，双方向通信が必要な Web アプリケーションにおいては，速度が 29 倍程度改善されることが分かった．なお，具体的な実験環境等，速度改善確認実験の詳細については「5.1 節．予備実験：WebSocket 利用による処理速度改善に関する実験」に示す．

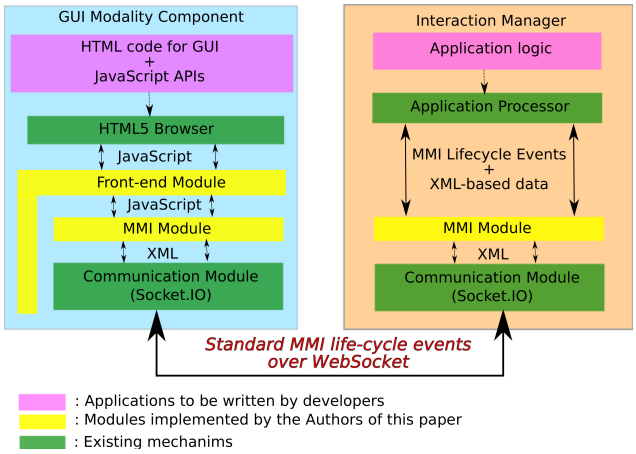

図 4　MMI over WebSocket
Fig. 4　MMI over WebSocket.

3.2　MMI over WebSocket の提案

以上の議論に基づき，本稿では図 4 に示すアーキテクチャを持つ「MMI over WebSocket」（以下，「本ライブラリ」）を提案する．本ライブラリは，(1) 開発者の開発負担削減に必要な標準化として W3C MMI アーキテクチャを利用するとともに，(2) 機能増大にともなう処理量の低減を図るべく通信プロトコルとして HTTP に代わり WebSocket を利用しており，以下の特長を持っている．

3.2.1　柔軟かつネットワーク透過的な Web と機器の統合

本ライブラリが参照している MMI アーキテクチャでは，音声対話システムのための分散アーキテクチャである「Galaxy アーキテクチャ [14], [15]」を参考にしたうえで，構成要素が増えると Hub（サーバ）の制御機能が複雑になるという「Hub-and-spoke モデル」の問題を解決するために入れ子構造を可能としている．これにより，たとえば，スマートフォン上のマイクに相当する音声入力（認識）MC とスピーカに相当する音声出力（合成）MC を，いったん，「音声インタフェース MC」として組み合わせたうえで，IM を通してデジタル TV と連携させ，音声による操作で TV を制御することが可能になる．

また，本ライブラリでは，WebSocket を利用して，JavaScript からライフサイクル・イベントを呼び出すための API を提供しているため，WebSocket に対応した Web ブラウザをそのままマルチモーダル Web アプリケーションにおける GUI モダリティとして利用することが可能である．たとえば，VoiceXML [16], [17], [18], [19]（音声入出力のための標準的マークアップ言語）や CCXML [20]（テレフォニーのための標準的マークアップ言語）を利用した音声入出力 MC と組み合わせることにより，Web ブラウザに音声入出力の機能を拡張することが可能になる．

3.2.2　アプリケーション処理ロジックと入出力モダリティ制御の分離

MMI アーキテクチャは，Web アプリケーション構築のためのデザインパターンである「Model-View-Controller（MVC）モデル [21]」に基づいて，Model（データ），View（入出力モダリティ），および Contoller（アプリケーション・ロジック）を明確に分離したうえで，アプリケーション・ロジックおよびデータは IM が制御し，入出力モダリティは各モダリティの操作に特化した MC が制御するという構造を持っているため，開発にあたってアプリケーション・ロジック（図 4 の「Application logic」）および入出力モダリティ制御（図 4 の「HTML code for GUI + JavaScript APIs」）を明確に分離することが可能である．したがって，GUI や音声インタフェース等の開発者は，アプリケーション全体の処理ロジックを意識することなく各入出力モダリティに必要な処理（ユーザ入力，結果出力等）に専念することが可能であり，逆に，アプリケーションのサービスプロバイダは，個別の入出力モダリティの詳細を意識することなくアプリケーション全体の処理ロジックに専念することが可能である．

3.2.3　複雑なネットワーク制御処理を意識せずにソケット通信を実現

本ライブラリでは，提案手法に必要なソケット通信を実現するにあたり，MMI アーキテクチャに準じたイベント処理を容易に行えるようにするために，Node.js [22], [23] および Socket.IO [24] を利用した．これにより，Web アプリケーション開発者は，複雑なネットワーク制御処理を意識することなくサーバ側とクライアント側での双方向ソケット通信を行うことが可能となる．

4.　実装方法

図 4 に示すとおり，本ライブラリは，(1) フロントエンド・モジュール，(2) MMI モジュール，および (3) 通信モジュールという 3 つのモジュールにより構成される．各モジュールの詳細について以下に説明する．

4.1　フロントエンド・モジュール

フロントエンド・モジュール（図 4 の「Front-end Module」）は MC 側で動作し，HTML5 ブラウザとの入出力インタフェースを受け持つとともに，MMI モジュールおよび通信モジュールの制御を行う．フロントエンド・モジュールが HTML5 ブラウザに対するインタフェースを統括し，MMI モジュールおよび通信モジュールの詳細を隠蔽することにより，MC の開発者は，MMI アーキテクチャのライフサイクル・イベントや WebSocket によるネットワーク通信の詳細を意識せずに，HTML コードおよび JavaScript API の記述に専念することができる．

4.2　MMI モジュール

4.2.1　MMI モジュールの役割

MMI モジュール（図 4 の「MMI Module」）は MC およ

び IM の双方で動作し，MMI アーキテクチャ仕様 [2] に基づいてライフサイクル・イベントおよび必要な XML データの生成を行う．MMI モジュールの役割は MC 側と IM 側で異なるため，2 つに分けたうえでそれぞれについて以下に説明する．

MC 側での役割：

MMI モジュールが MC 側で動作する場合は，フロントエンド・モジュールから渡された JavaScript を XML に変換したうえで通信モジュールに渡す一方で，通信モジュールから受け取った XML を JavaScript へ変換したうえでフロントエンド・モジュールに渡す．

IM 側での役割：

MMI モジュールが IM 側で動作する場合は，通信モジュールから受け取った XML に基づいて，MMI アーキテクチャで規定されたライフサイクル・イベントを生成したうえでアプリケーション処理エンジンに渡す．その一方で，アプリケーション処理エンジンから受け取ったイベントに基づいて XML データを生成し通信モジュールに渡す．

なお，図 4 に示すとおり，HTML5 ブラウザでは入力情報として一般的に「HTML および JavaScript」が想定されている一方で，MMI アーキテクチャでは，マルチモーダル Web アプリケーションのための標準的データ形式である EMMA [25] 等の XML ベースのデータ形式を想定しているため，MMI モジュールでは，ブラウザ内で利用される JavaScript と，ネットワーク側で利用される XML との相互変換を行う必要がある．

4.2.2　MMI モジュールで取り扱う標準的ライフサイクル・イベント

MMI モジュールでは，Web アプリケーション開発に利用されるイベントとして，MMI アーキテクチャ [2] で規定されている以下の「標準的ライフサイクル・イベント」を取り扱う．

NewContextRequest および NewContextResponse：

アプリケーション開始にあたって，MC は IM に *NewContextRequest* イベントを送信することができる．IM は，*NewContextRequest* を受信した場合，MC へ *NewContextResponse* イベントを返信しなければならない．

PrepareRequest および PrepareResponse：

データやスクリプト等をあらかじめ準備しておけるように，IM は MC へ *PrepareRequest* イベントを送信することができる．なお，MC は，このイベントに対して返信する必要はないが，返信する場合には *PrepareResponse* イベントを返信しなければならない．

StartRequest および StartResponse：

MC を起動するにあたって，IM は MC に *StartRequest* イベントを送信しなければならない．MC は，*StartRequest* イベントを受信すると，IM に *StartResponse* イベントを返信しなければならない．

DoneNotification：

MC は，自らの処理が終了すると，IM に *DoneNotification* イベントを送信しなければならない．

CancelRequest および CancelResponse：

IM は，MC の処理を強制終了させるために，MC に *CancelRequest* イベントを送信することができる．*CancelRequest* イベントを受信した場合，MC は，自らの処理を停止するとともに IM へ *CancelResponse* イベントを返信しなければならない．

PauseRequest および PauseResponse：

IM は，MC 内の処理を一時停止させるために，MC に *PauseRequest* イベントを送信することができる．MC は，*PauseRequest* イベントを受信した場合，自らの処理を一時停止した後（もしくは一時停止が不可能であると判定した後），IM に *PauseResponse* イベントを返信しなければならない．

ResumeRequest および ResumeResponse：

IM は，*PauseRequest* イベントにより一時停止された MC 内の処理を再開させるために，MC に *ResumeRequest* イベントを送信することができる．MC は，*ResumeRequest* イベントを受信した場合，一時停止された処理を再開するよう試みるとともに IM に *ResumeResponse* イベントを返信しなければならない．

ExtensionNotification：

ExtensionNotification イベントは，IM および MC のいずれからも相手に送信することができ，MMI アーキテクチャ [2] で規定されたライフサイクル・イベント以外の，各アプリケーションに特化した拡張的なイベントをライフサイクル・イベントに隠蔽して受け渡しするために用いられる．

ClearContextRequest および ClearContextResponse：

IM は，今まで利用していたコンテクストが不要になった場合，そのことを知らせるために，MC へ *ClearContextRequest* イベントを送信することができる．MC は，*ClearContextRequest* イベントを受信した場合，IM に *ClearContextResponse* イベントを返信しなければならない．

StatusRequest および StatusResponse：

StatusRequest イベントおよび *StatusResponse* イベントは，「keep-alive 機能」（ネットワーク上で接続が有効であることの確認）のために用いられる．IM および MC のいずれからも *StatusRequest* イベントを相手に送信することができる．*StatusRequest* イベントを受信した場合，MC もしくは IM は，送信側に

StatusResponse イベントを返信しなければならない．

4.3　通信モジュール

「通信モジュール」（図 4 の「Communication Module」）は，WebSocket を利用して MC および IM の間で，MMI アーキテクチャで規定されたライフサイクル・イベントの送受信を行うものであるが，この機能を実現するにあたっては，一般的に利用されている JavaScript 開発環境である Node.js [22], [23] および Socket 通信ライブラリである Socket.IO [24] を流用した．

5.　評価

「3 章．提案手法：MMI over WebSocket ライブラリ」で議論したとおり，開発者の能力に依存しない安定したアプリケーション開発を可能とするための本ライブラリ開発には以下の 2 つの要素がある．

(1) 標準化された手法による入出力モダリティの動的な取扱い
(2) 機能増大にともなう処理量増加への対処

上記のうち (1) については，標準化フレームワークとして W3C の提唱する MMI アーキテクチャを利用するとともに同アーキテクチャの機能を一般的な Web アプリケーションから利用するための JavaScript API を提供して対応する．また，(2) については，通信プロトコルを HTTP から WebSocket に変更することにより，処理速度およびサーバ負荷の低減を試みる．そのため，WebSocket 利用による処理速度改善に関する予備実験を行ったところ十分な効果が得られた．そこで，実装したライブラリを用いて，実際にサーバ負荷について評価を行った．以下に予備実験およびサーバ負荷実験の詳細を示す．

5.1　予備実験：WebSocket 利用による処理速度改善に関する実験

通信プロトコルとして，HTTP の代わりに WebSocket を用いることにより処理速度を改善することができるかどうかを確認するために，予備実験として，日本語テキストの分かち書き処理を行う Web アプリケーション [26] を用いて，HTTP と WebSocket のそれぞれにおける処理時間を 100 回繰り返して測定した．以下に実験環境および実験結果を示す．

5.1.1　実験環境

本実験に用いた実験環境およびそのネットワーク構成を図 **5** に示す．

5.1.2　実験結果

通信プロトコルとして HTTP と WebSocket を用いて処理時間をそれぞれ 100 回繰り返して測定し，取得した処理時間結果を 20 ms ごとの bin に分けて度数分布を確認したところ，**図 6** に示すとおり，HTTP 接続（図 6 中の緑色のグラフ）において最も頻度の多かった処理時間は 4,880 ms〜4,900 ms であり，その中央値は 4,890 ms であった．また，WebSocket 接続（図 6 中の赤色のグラフ）において最も頻

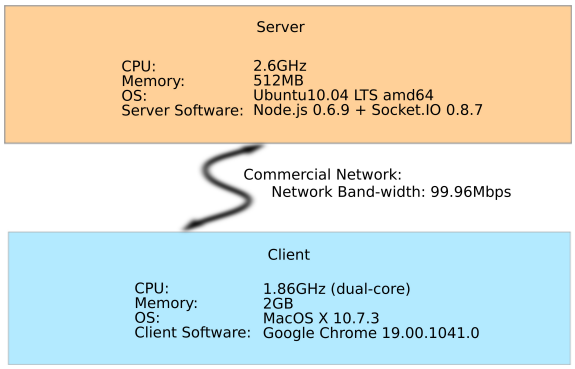

図 5　WebSocket 利用による処理速度改善実験環境
Fig. 5　Speed test environment.

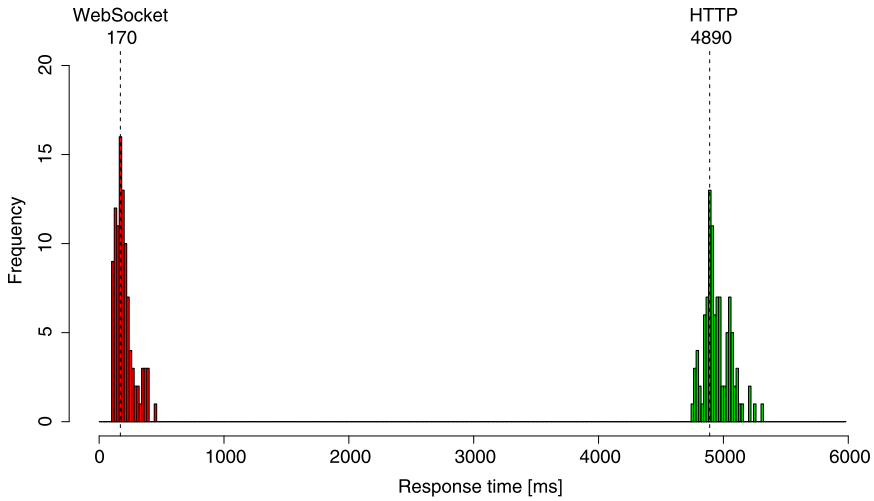

図 6　WebSocket 利用による処理速度改善実験結果
Fig. 6　Speed test results.

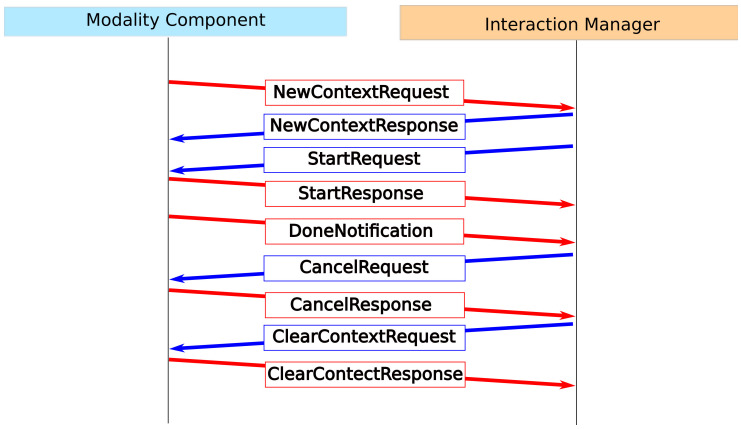

図 7　サーバ負荷低減実験で想定したライフサイクル・イベントのやりとり
Fig. 7　Lifecycle Event transaction during the load test.

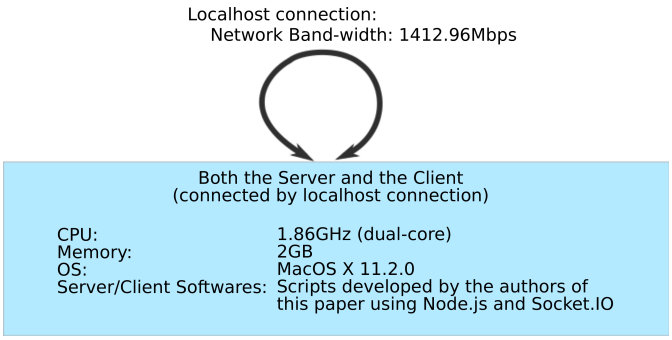

図 8　WebSocket 利用によるサーバ負荷低減実験環境
Fig. 8　Load test environment.

度の多かった処理時間は 160 ms～180 ms であり，その中央値は 170 ms であった．したがって，WebSocket を利用することにより約 29 倍の速度向上が望めることが分かった．

5.2　WebSocket 利用によるサーバ負荷低減に関する実験

本ライブラリにおいて，通信プロトコルとして HTTP の代わりに WebSocket を用いることによりサーバ負荷を低減することが可能かどうかを確認するべく，以下の 2 つを作成したうえで，単一の IM に対して同時に複数（1 件～100 件）の MC が接続した場合における，IM 側での CPU 負荷およびメモリ使用量の推移を確認した．

- HTML5 ブラウザに相当する簡易なクライアント・ソフトウェアを組み込んだ MC
- アプリケーション処理エンジンに相当する簡易な処理系を組み込んだ IM

なお，本実験では，1 つのアプリケーションの開始から終了に至る過程で，MC と IM の間で，図 7 に示すライフサイクル・イベントのやりとりが行われることを想定した．

以下に本実験の実験環境および実験結果を示す．

5.2.1　実験環境

本実験に用いた実験環境およびそのネットワーク構成を図 8 に示す．

5.2.2　実験結果

CPU 負荷については，HTTP を利用した場合，3 件の同時接続時において 100%に達し，それ以上の負荷に関する確認を行うことができなかった．なお，参考情報として 2 件の同時接続において CPU 負荷は 70%であった．一方で，WebSocket を利用した本ライブラリでは，図 9 に示すとおり，100 件同時接続において CPU 負荷が約 50%であることが分かった．

また，メモリ使用量については，HTTP を利用した場合，CPU 負荷が 100%に達していた影響で，3 件以上の同時接続は測定不能であった．なお，参考情報として 2 件同時接続時においてメモリ使用量は 26 MB であった．一方で，WebSocket を利用した本ライブラリでは，図 9 に示すとおり，100 件同時アクセスした場合の使用量が 24 MB 程度であることが分かった．

5.3　実験結果に基づく考察

「5.1 節．予備実験：WebSocket 利用による処理速度改善に関する実験」の実験結果より，通信プロトコルとして HTTP に代わり WebSocket を利用することにより約 29 倍の速度向上が望めることが分かったため，「4 章．実装方法」に説明したとおり本ライブラリのアーキテクチャを詳細化し実装を行った．さらに，「5.2 節．WebSocket 利用に

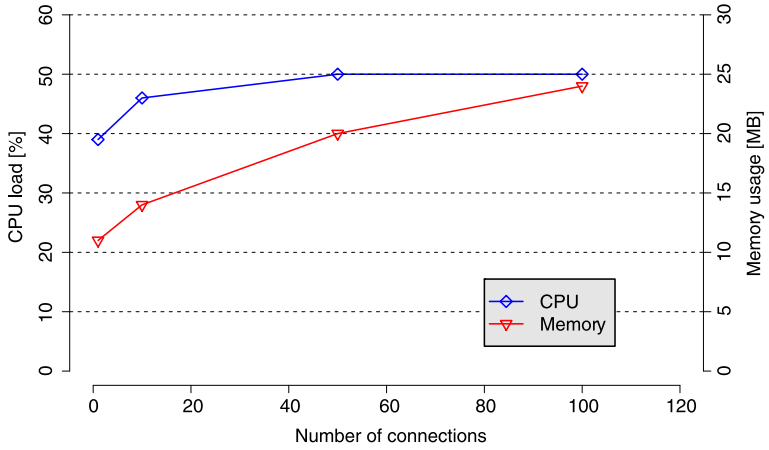

図 9　負荷試験結果
Fig. 9　Load test results.

よるサーバ負荷低減に関する実験」に示すとおり，実装したライブラリを用いて実際にサーバ負荷について評価したところ，想定していたとおりサーバ側の CPU 負荷およびメモリ使用量を低減できることが分かった．したがって，標準化フレームワークに沿ったライブラリである提案手法により，アプリケーションに必要な入出力モダリティを動的に取り扱うことができるようになるとともに，機能増大にともなう処理量増加にも対処できることが明らかになった．なお，HTTP を利用した場合，3 件以上の MC が同時接続すると，サーバ側の CPU 負荷が 100%を超え処理が不可能となったが，本ライブラリを利用した WebSocket 接続の場合，100 件の同時接続においても 50%の CPU 負荷で処理を行うことができたため，3 件以上の同時接続においては本ライブラリの利用が効果的であると考えられる．

6.　標準化

著者らが参照した MMI アーキテクチャは W3C において国際標準化が進んでいる．W3C では，今までに MMI アーキテクチャに関する 2 回の国際ワークショップ（MMI アーキテクチャ自体に関するワークショップ [27] および Web と機器の連携に関するワークショップ [28]）を開催しているが，これらのワークショップにおける国際的な議論を通じ，同アーキテクチャの仕様化にあたって以下の 2 点の重要性が認識されている．

(1) 家電機器を含む様々な機器を透過的に Web と連携させること
(2) Web と機器の連携を，開発者の負担を低減させる形で実現すること

なお，著者らのうち芦村は MMI アーキテクチャの標準化に取り組む W3C Multimodal Interaction Working Group (MMI-WG) [29] の活動責任者として，2005 年以来，同 WG の活動を主導するとともに上記 2 点の重要性について WG へ継続的に提案してきた．上記提案のうち (1) については，

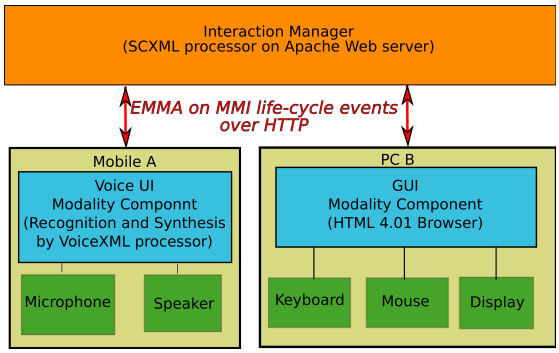

図 10　HTTP コネクションによる MMI プロトタイプ（第 1 版プロト）
Fig. 10　MMI Prototype (Ver. 1) by HTTP Connection.

すでに同 WG の 2011～2013 年活動計画書（“Multimodal Interaction Working Group Charter”）[30] に採用されており，MMI アーキテクチャで利用される標準的なデータ形式である EMMA [25] を拡張するための EMMA Version 1.1 [31] の標準化が行われることとなった．また，(2) についても，同 WG 内に専門のタスクフォース（Interoperablity Testing Task Force; Interop TF）が設置され，開発者負担を軽減するべく，MMI アーキテクチャ仕様 [2] に基づいて開発されたシステム実装の相互運用性検証，および実装ガイドライン作成に取り組んでいる．Interop TF では，すでに**図 10** に示すような，(a) 音声 MC（Voice UI Modalty Component），(b) GUI MC（GUI Modality Component），および (c) IM で構成され，各要素間の通信を HTTP で行うプロトタイプの実装が完成しており，実装の詳細および開発時に洗い出された問題点等については，“MMI interoperability test report” [32] という W3C の正式な文書として公開されている．さらに，同 TF では，本ライブラリを適用した第 2 版プロトタイプの開発が始まっており，こちらについても，実装の詳細および開発時の問題点等について，W3C の正式な文書として公開することが予定され

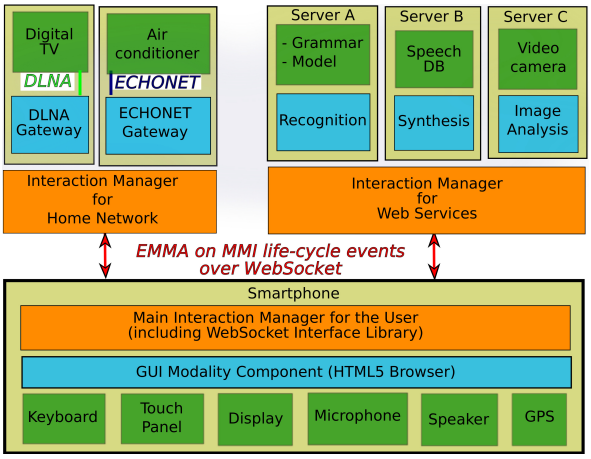

図 11　WebSocket による MMI プロトタイプ（第 2 版プロト）
Fig. 11　MMI Prototype (Ver. 2) by WebSocket.

ている．なお，第 2 版プロトタイプでは，図 **11** に示すとおり，デジタルテレビやエアコン等の家電機器と HTML5 ブラウザとの連携が対象とされており，今後，WebSocket を利用した本ライブラリにより，機器と Web のネットワーク透過的な連携を実現していくための基本実装として期待されている．

7.　まとめ

本稿では，まず，Web アプリケーション開発に関連する現状の技術とその課題について触れたうえで，その課題を解決するべく，標準的なフレームワークに沿ったソフトウェアを Web アプリケーション開発者に提供することで，開発者の力量に依存することなくコンテンツの多様化に対応する方法を提案するとともに，W3C MMI アーキテクチャ [2], [3], [4] を利用した標準化フレームワークに沿ったライブラリの実装方法について論じた．さらに，実際に上記ライブラリを実装したうえで評価を行い，提案手法の通信プロトコルとして WebSocket を利用することにより処理速度を向上するとともにサーバ負荷を低減できることを確認した．これにより，標準化フレームワークに沿ったライブラリである提案手法により，アプリケーションに必要な入出力モダリティを動的に取り扱うことができるようになるとともに，機能増大にともなう処理量増加にも対処できることを確認した．

なお，提案手法は MMI アーキテクチャの国際標準化に取り組む W3C に対しても提案されており，今後は，W3C と協調して提案手法を組み込んだプロトタイプ・システム（図 11）の実装を行い，提案手法により，HTML5 ブラウザ，各種機器，およびクラウド・サービスの透過的な連携を実現できることを実証していく予定である．また，提案手法を実装した「MMI over WebSocket ライブラリ」は，世界中の開発者がマルチモーダル Web アプリケーションを構築する際の一助とするべく一般公開する予定である．

参考文献

[1] W3C: W3C Top Page, W3C (online), available from ⟨http://www.w3.org/⟩ (accessed 2012-01-20).
[2] Barnett, J. et al.: Multimodal Architecture and Interfaces (MMI Architecture), W3C (online), available from ⟨http://www.w3.org/TR/mmi-arch/⟩ (accessed 2012-01-12).
[3] Maes, S.H. and Saraswat, V.: Multimodal Interaction Requirements, W3C (online), available from ⟨http://www.w3.org/TR/mmi-reqs/⟩ (accessed 2003-01-08).
[4] Grifoni, P. et al.: Multimodal Human Computer Interaction and Pervasive Services, IGI Global, 701 E. Chocolate Ave. Hershey, PA 17033, USA (2009), available from ⟨http://www.igi-global.com/book/multimodal-human-computer-interaction-pervasive/787⟩.
[5] Wikipedia: HTML, Wikipedia (online), available from ⟨http://en.wikipedia.org/wiki/HTML⟩ (accessed 2012-01-20).
[6] Ragget, D., Hors, A.L. and Jacobs, I.: HTML 4.01 Specification, W3C (online), available from ⟨http://www.w3.org/TR/html401/⟩ (accessed 1999-12-24).
[7] 佐伯千種，島田博也，田畑伸哉：WWW コンテンツ統計調査報告書—我が国の Web 上のコンテンツ情報量から見たインターネットの発展 (No.2004-I-02)，総務省（オンライン），入手先 ⟨http://www.soumu.go.jp/iicp/chousakenkyu/data/research/survey/telecom/2004/2004-1-02-1.pdf⟩（参照 2004-07）.
[8] Wikipedia: HTML5, Wikipedia (online), available from ⟨http://en.wikipedia.org/wiki/HTML5⟩ (accessed 2012-01-20).
[9] Hickson, I.: HTML 5 A vocabulary and associated APIs for HTML and XHTML, W3C (online), available from ⟨http://www.w3.org/TR/html5/⟩ (accessed 2012-03-29).
[10] Google: HTML5 ROCS, Google (online), available from ⟨http://www.html5rocks.com⟩ (accessed 2012-01-20).
[11] Humphrey, D. et al.: Audio Data API, Mozilla (online), available from ⟨https://wiki.mozilla.org/Audio_Data_API⟩ (accessed 2012-01-20).
[12] Wikipedia: WebSocket, Wikipedia (online), available from ⟨http://en.wikipedia.org/wiki/WebSocket⟩ (accessed 2012-01-20).
[13] Hickson, I.: The WebSocket API, W3C (online), available from ⟨http://www.w3.org/TR/websockets/⟩ (accessed 2011-12-08).
[14] Bayer, S.: *Building a Standards and Research Community with the Galaxy Communicator Software Infrastructure*, Springer, Berlin, Germany (2005). (in *Practical Spoken Dialog Systems*, Dahl, D.A. (Ed.), Vol.26, pp.166–196, Dordrecht: Kluwer Academic Publishers).
[15] MIT: Galaxy Communicator Documentation, MIT (online), available from ⟨http://communicator.sourceforge.net/sites/MITRE/distributions/GalaxyCommunicator/docs/manual/⟩ (accessed 2003-09-24).
[16] Sharma, C. and Kunins, J.: *VoiceXML*, John Wiley and Sons, Inc., New York, USA (2002).
[17] McGlashan, S., Burnett, D.C., Carter, J., et al.: Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C (online), available from ⟨http://www.w3.org/TR/voicexml20/⟩ (accessed 2004-03-16).
[18] Oshry, M., Auburn, R., Paolo, B., Bodell, M., et al.:

Voice Extensible Markup Language (VoiceXML) 2.1, W3C (online), available from ⟨http://www.w3.org/TR/voicexml21/⟩ (accessed 2007-06-19).
[19] Hocek, A. and Cuddihy, D.: *Definitive VoiceXML*, Prentice-Hall, New Jersey, USA (2002).
[20] Auburn, R. et al.: Voice Browser Call Control: CCXML Version 1.0, W3C (online), available from ⟨http://www.w3.org/TR/ccxml/⟩ (accessed 2011-07-05).
[21] Wikipedia: Model-view-controller, Wikipedia (online), available from ⟨http://en.wikipedia.org/wiki/Model-view-controller⟩ (accessed 2012-01-20).
[22] Wikipedia: Node.js, Wikipedia (online), available from ⟨http://en.wikipedia.org/wiki/Node.js⟩ (accessed 2012-01-20).
[23] Joyent: Node.js official site, Joyent, Inc. (online), available from ⟨http://nodejs.org/⟩ (accessed 2012-01-20).
[24] Rauch, G.: Socket.IO: the cross-browser WebSocket for realtime apps., LearnBoost, Inc. (online), available from ⟨http://socket.io/⟩ (accessed 2012-01-20).
[25] Johnston, M. et al.: EMMA: Extensible MultiModal Annotation markup language, W3C (online), available from ⟨http://www.w3.org/TR/emma/⟩ (accessed 2009-02-10).
[26] 小松健作：Wakachi demo（WebSocket による日本語テキスト分かち書きデモ），Komasshu（オンライン），入手先 ⟨http://wakachi.komasshu.info⟩（参照 2012-01-20）.
[27] Ashimura, K.: Workshop on W3C's Multimodal Architecutre and Interfaces — Summary, W3C (online), available from ⟨http://www.w3.org/2007/08/mmi-arch/summary.html⟩ (accessed 2007-11-26).
[28] Ashimura, K.: Workshop on Conversational Applications — Summary, W3C (online), available from ⟨http://www.w3.org/2010/02/convapps/summary.html⟩ (accessed 2010-06-29).
[29] Ashimura, K.: Multimodal Interaction Working Group, W3C (online), available from ⟨http://www.w3.org/2002/mmi/⟩ (accessed 2002-02-14).
[30] Ashimura, K.: Multimodal Interaction Working Group Charter, W3C (online), available from ⟨http://www.w3.org/2011/03/mmi-charter.html⟩ (accessed 2011-12-23).
[31] Johnston, M., Baggia, P., Bodell, M., Burnett, D.C. and Dahl, D.A.: EMMA: Extensible MultiModal Annotation markup language Version 1.1, W3C (online), available from ⟨http://www.w3.org/TR/emma11/⟩ (accessed 2012-02-09).
[32] Kliche, I., Kharidi, N. and Wiechno, P.: MMI interoperability test report, W3C (online), available from ⟨http://www.w3.org/TR/mmi-interop/⟩ (accessed 2012-01-24).

芦村 和幸　（正会員）

1967 年生．1992 年京都大学理学部数学科卒業．2005 年奈良先端科学技術大学院大学情報科学研究科博士後期課程単位取得退学．1992 年 NTT ソフトウェア（株）入社．（株）ATR 音声翻訳通信研究所，（株）アルカディア，JST/CREST「表現豊かな発話音声のコンピュータ処理」研究員を経て，2005 年より W3C/慶應．音声，マルチモーダルおよび Web&TV に関する国際標準化に従事．慶應義塾大学大学院政策・メディア研究科特任准教授．電子情報通信学会，日本音響学会各会員．

小松 健作

1972 年生．1997 年横浜国立大学大学院電子情報工学専攻修士課程修了．同年 NTT 入社．現在 NTT コミュニケーションズ先端 IP アーキテクチャセンタ主査．HTML5 等最新 Web 技術の研究開発に従事．主な著書に『徹底解説 HTML5 API ガイドブック コミュニケーション系 API 編』（単著．秀和システム）．

一色 正男　（正会員）

1956 年生．1982 年東京工業大学理工学研究科修士卒業．（株）東芝で約 30 年，新規技術開発と新規事業開発を中心に働く．2009 年 1 月より慶應義塾大学教授として，国際標準化団体 W3C（World Wide Web Consortium）のサイト・マネージャに就任．W3C 運営委員．神奈川工科大学ホーム・エレクトロニクス学科教授．慶應義塾大学大学院政策・メディア研究科特任教授．工学博士．情報処理学会 CDS 研究会幹事．機械学会会員．ECHONET コンソーシアム 2008 運営委員長，現フェロー．経済産業省 HEMS タスクフォース座長．著書：『これからの日本が世界で戦うために必要なこと』（By デジカル，電子書籍）．