

小型 HTTP コミュニケータ SMACH の機能設計

Design and Development of SMACH (Smart Communicator for HTTP)

高田 祐輔†
Yusuke Takada村上 陽祐†
Yosuke Murakami大谷 真†
Makoto Oya

1. はじめに

近年の Web 技術の発展により HTTP[1]は SOAP/Web サービスに見られるようなプロセス間通信の基盤プロトコルとして利用されるようになってきた。一方で従来技術の延長として進歩してきた既存の HTTP 基盤ソフトウェアは機能やプログラムサイズの肥大化、サーバ側クライアント側の非対称性などの問題をもち、ロボット、組み込み機器を含む小型の対等分散型システムの基盤に適していない。これを解決するために小型 HTTP コミュニケータ SMACH(Smart Communicator for HTTP)を開発した。

本論文では、その開発方針、機能設計を中心とした設計方針と解決方法、及び相互運用性についての検証実験の結果を述べる。

2. SMACH 開発の概要

2.1 背景

1. HTTP サーバの肥大化

Web の発達とともに HTTP の上位・周辺に、Web プレゼンテーション、アプリケーション管理・制御、コンテンツ管理、Web サービスなど多種の機能が必要となった。この結果、Apache や IIS などの多くのサーバでは従来からの互換性を保ちつつ、これらの機能をその一部として抱え込むことになり機能面及びプログラムサイズ面で肥大化している。

2. クライアント/サーバシステムから対等分散システムへの推移

インターネット内での通信が、クライアント/サーバの多対一の関係からクライアント/サーバ両方の機能をもったシステム同士が対等に通信する一対一の関係へと HTTP 基盤アーキテクチャが変化している。しかしソフトウェアとしては未だクライアント/サーバは別々のままである。

3. 小型機器への Web サービスの利用

コンピュータシステム間だけではなく、小型機器間での HTTP 通信の適用が期待されるようになってきた。また、ロボット間疎結合に HTTP や Web サービスを使うことも提案されている[2]。これに伴い HTTP 基盤ソフトウェアの肥大化と対等分散非対応がさらに大きな問題となってきている。

2.2 開発方針

(1) SMACH 全体の設計方針

- 機能を HTTP 通信に特化させる。すなわち表示コンテンツ取得を主体とせず、データ交換に必要な機能に極小化させる
- 一つのソフトウェアでクライアント、サーバ両方の機能を実現する

- 既存のサーバ基盤 (Apache, IIS 等), クライアント基盤 (Mozilla, Internet Explorer, Java HTTP クラス等) との接続も可能とする
- プログラムサイズを小型化する

(2) SMACH 概観

図1に SMACH の概観を示す。SMACH はクライアント、サーバ両方の機能を持っており、その間は HTTP で通信する。SMACH の上位アプリケーション (以下 AP) は SMACH が提供する API を通して HTTP メッセージの送受信機能を利用することができる。クライアント AP は SMACH クライアントをライブラリとして利用する。サーバ AP は SMACH サーバのライブラリとして実現される。また、SMACH は既存の HTTP クライアント、サーバと相互通信可能である。

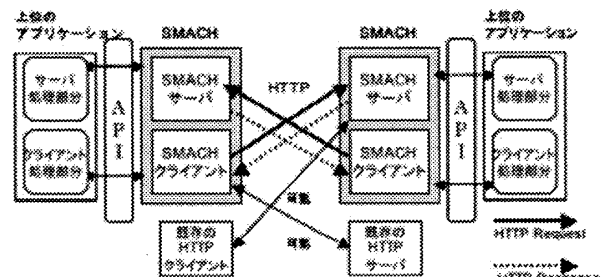


図1 SMACH 概観

2.3 機能設計方針

HTTP 機能

- 最新のプロトコルである HTTP/1.1 に準拠する。
- データ交換で必要となる機能を実現する。
- 目視を目的としたコンテンツ処理機能は持たない。必要なら API を通して上位 AP が実行可能とする。

API 機能

- クライアント、サーバの API に対称性を持たせる。
- プロトコル (書式、伝送の流れ) やメモリ管理を AP から隠蔽する。
- 使用するリソースを極力小さくする。

その他の機能

- プログラムサイズを極力小さくするために、最小限のシステム管理機能を持たせる。
- AP と SMACH はバイナリレベルで分離する。これにより上位 AP は SMACH とは分離開発可能とする。

サポート機能一覧を表1に示す。

† 北海道大学大学院情報科学研究科

表1 サポート機能一覧

分類	機能
HTTP 機能	HTTP/1.1 必須機能 (GET, HEAD メソッド, チャンクのデコード等)
	認証を使ったアクセス制限機能
	性能向上に関わる機能
API 機能	HTTP メッセージの作成, 解析機能
	HTTP メッセージの送受信機能
	クライアント, サーバの対称機能
	セッション管理機能
その他の 機能	Config ファイルによるシステム設定機能
	ログファイル作成機能
	インストール機能
	AP のダイナミックリンク機能

3. HTTP 機能設計

(1) HTTP 機能

HTTP/1.1 では各プロトコル機能は "MUST", "REQUIRED", "SHOULD", "MAY", "OPTIONAL" などに分類されている。HTTP/1.1 に準拠するためには MUST 条件を必ず満たしていなければならない。以下の機能は MUST であるのでサポートすることにした。

- GET, HEAD メソッド
- Date ヘッダ
- Host ヘッダ
- チャンク形式でエンコードされた HTTP メッセージのデコード
- 主要なステータスコードの解釈

次に MUST ではないが方針に従ってサポートした主要な機能について述べる。

- 持続的接続
一度 TCP 接続を確立すると、その接続を保持したままリクエスト、レスポンスを送り合うことができる。通信の度に新しい接続を確立する場合に比べ、サーバの負荷、TCP 接続確立におけるオーバーヘッドを減らすことができる。
- 認証
システム間でのデータの受け渡しには安全性が求められる。BASIC 認証によりファイルアクセスに対して制限を行い、安全性に配慮した。
- チャンク形式でのエンコード
HTTP メッセージの送信を一括ではなく、チャンク (分割されたメッセージ) 単位で可能にした。これにより、動的に生成されたコンテンツを効率よく送信できる。
- POST メソッド
Web サービスでは POST メソッドが必須である。SMACH はデータ交換が主目的であるのでこれをサポートした。

主な機能を表2にまとめる。

(2) コンテンツ処理機能

SMACH はデータ交換機能を持つが、以下のようなコンテンツ処理機能は持たない。これらの機能は後述する API を用いることで実現できる。

- キャッシュ
- コンテンツネゴシエーション

● クッキー

また、Apache や IIS など既存のサーバや Mozilla や Internet Explorer などのクライアントには上で述べた機能の他に以下のような機能がある。

- CGI
- SSI
- 表示用ファイル配信のための機能 (特定のディレクトリにあるファイルをサーバが読み出し、クライアントに返信する機能)
- 文章表示機能
- HTML や XML のパース機能

これらは表示を目的としたコンテンツを扱うための機能であるのでサポートしない。

表2 HTTP 機能

機能	要求レベル
GET, HEAD メソッド	MUST
Date, Host ヘッダ	MUST
チャンク形式のデコード	MUST
主要なステータスコードの解釈	MUST
持続的接続	SHOULD
アクセス認証	OPTIONAL
POST メソッド	OPTIONAL
チャンク形式へのエンコード	—

4. API 設計

1. メッセージの作成, 解析

メッセージはスタートライン、ヘッダ、ボディで構成され、それぞれ書式が決まっている。送信時に AP から値をもらって書式にそったメッセージを作成する関数、受信時にメッセージを解析して必要な値を抜き出し AP に渡す関数を作成した。これらの機能により AP から HTTP 書式が隠蔽されるようにした。

2. メッセージの送受信

送信時、AP は分割したボディメッセージを複数回にわけて SMACH に渡せるようにした。SMACH 内部では一定のサイズになると送信する。受信時も SMACH は一定サイズで受信し、AP は任意のタイミングでそのデータを受けとれるようにした。これにより AP から HTTP の伝送の流れを隠蔽した。また、一定サイズごとに送受信することでメモリ内になるべくデータを蓄積せず、限られたリソースでの送受信を可能にした。

3. クライアント, サーバの API 対称性

SMACH は HTTP のクライアント/サーバ両機能を 1 つのソフトウェアで実現した。HTTP 通信には送受信処理に対称性があるので、それを利用することで API に統一性を持たせた。表3は AP がメッセージを送信、受信する時に使う API の具体例である。

表3 対称的な API の例

クライアント API
sma_send_request(void *handle, char *body, int size)
sma_receive_response(void *handle, char *body, int size)
サーバ API
sma_receive_request(void *handle, char *body, int size)
sma_send_response(void *handle, char *body, int size)

4. メモリ管理の隠蔽

送受信時に必要なデータは AP からは void 型のハンドルとしてしか見えない。このデータに対するメモリの領域確保、解放を SMACH 内で行うことでメモリ管理を AP から隠蔽した。

5. セッション管理

持続的接続をサポートし、リクエスト/レスポンスを連続実行できるようにした。これにより送受信時のデータがメモリに蓄積されていくので、送受信処理の度にデータをクリアする関数を作り、リソースの消費を抑えた。また、接続が確立してから切断するまでをセッションとする。Connection ヘッダを参照することでセッションの終わりを検知し、それを AP に通知するようにした。

図2、図3に開発した API(sma というプレフィックスがついている)を用いたプログラムの例を示す。

5. 検証実験

以下の3つのテストを行った。すべて、クライアントが HTML ファイルの取得要求を出しサーバは HTML ファイルを返却するようにした。(表4)

(テスト1) SMACH API を使ったクライアント AP とサーバ AP との接続

(テスト2) SMACH API を使ったクライアント AP と既存のサーバ (Apache, IIS) との接続

(テスト3) 既存のクライアント (Mozilla, Internet Explorer, JAVA HTTP クラス) と SMACH API を使ったサーバ AP との接続

表4 検証実験結果

	クライアント	サーバ	結果
テスト1	SMACH-C+ AP	SMACH-S+ AP	○
テスト2	SMACH-C+ AP	Apache	○
		IIS	○
テスト3	Internet Explorer	SMACH-S+ AP	○
	Mozilla		○
	Java HTTP クラス		○

以上の結果からテスト1で作成した SMACH の API が妥当であったことが検証された。次にテスト2, 3からは SMACH は SMACH 同士だけではなく、広く一般的に使われている HTTP 基盤ソフトウェアとも相互通信可能であることが検証された。

6. おわりに

SMACH は HTTP 機能への特化と、対等分散システムへの対応を目的に開発した。以上述べたとおり、この目標は達成できた。また、実行ファイルのサイズは 46KB となり、Apache が約 1222KB であるのに対してかなり小さなシステムとなった。これにより小型機器にも適したプログラムサイズのソフトウェアとなった。さらに、検証実験結果から既存のシステムとも通信可能であることを確認できた。

今後は SMACH をオープンソースとして公開する予定であり、ロボットや組み込み型機器等の小型機器の基盤ソフトウェアとして広く普及させたい。

```
int server_application(const void *handle) {
    int rval = 0;
    char language[10];
    int size = 1024;
    char req_body[1024];
    char res_body1[] = "<?xml version = ...>";
    char res_body2[] = "<soapenv:Envelope...>";
    char res_body3[] = "</soapenv:Envelope>";
    /*受信したリクエストメッセージに対する処理*/
    /*ヘッダの値を取得する*/
    sma_get_accept_language(handle, &language);
    /*ボディを取得する*/
    while(1) {
        rval = sma_receive_request(handle, body, size);
        if (rval == 0) break;
        /*ここでボディメッセージに対する処理を行う*/
    }
    /*レスポンスメッセージの作成*/
    /*ステータスラインの作成*/
    sma_set_status_line(handle, 200);
    /*ヘッダの作成*/
    sma_set_res_content_language(handle, "ja");
    /*ボディの作成*/
    sma_send_response(handle, res_body1, strlen(res_body1));
    sma_send_response(handle, res_body2, strlen(res_body2));
    sma_send_response(handle, res_body3, strlen(res_body3));
    sma_send_response(handle, NULL, 0);
    return(0);
}
```

図2 サーバアプリケーション例

```
int client_application(void) {
    void *handle;
    int size = 1024;
    char res_body[1024];
    char req_body1[] = "<?xml version = ...>";
    char req_body2[] = "<soapenv:Envelope...>";
    char req_body3[] = "</soapenv:Envelope>";
    char uri[] = "www.smach.com";
    char language[10];
    /*接続の確立*/
    sma_bind(&handle, uri, 80);
    /*データの初期化*/
    sma_initialize(handle);
    /*リクエストメッセージの作成*/
    /*リクエストラインの設定*/
    sma_set_request_line(handle, POST, "/service");
    /*ヘッダの設定*/
    sma_set_req_content_language(handle, "ja");
    /*ボディの設定*/
    sma_send_request(handle, body1, strlen(body1));
    sma_send_request(handle, body2, strlen(body2));
    sma_send_request(handle, body3, strlen(body3));
    sma_send_request(handle, NULL, 0);
    /*レスポンスメッセージの受信*/
    /*ヘッダを取得*/
    sma_get_res_content_type(handle, language);
    /*ボディを取得する*/
    while(1) {
        rval = sma_receive_response(handle, body, size);
        if (rval == 0) break;
        /*ここでメッセージボディに対する処理を行う*/
    }
    /*接続を切断*/
    sma_close(handle);
    return (0);
}
```

図3 クライアントアプリケーション例

参考文献

[1]T. Berners-Lee, et al., "Hypertext Transfer Protocol HTTP/1.1," W3C Recommendation, 1999.

[2]M.Oya, et al., "Loose Robot Communication over the Internet," Journal of Robot and Mechatronics, Vol.16, No.6, pp. 626-634