LL-002

小型 HTTP コミュニケータの方式設計

Architecture of SMACH（Smart Communicator for HTTP）

村上 陽祐† 高田 祐輔† 木村 将希† 大谷 真†
Yosuke Murakami Yusuke Takada Masaki Kimura Makoto Oya

1. はじめに

HTTP[1]は，SOAP/Web サービスなどの対等分散システムにおける通信基盤プロトコルとして，インターネットでの利用が拡大している．更に，コンピュータシステム間だけではなく，インターネットで接続されたロボットや小型機器間通信への適用も期待されている[2]．しかし，現状の HTTP 基盤ソフトウェアは，クライアント側とサーバ側が異なるソフトウェアとなっているため，対等分散システムの実装基盤として十分ではない．また，Apache 等の HTTP サーバや Mozilla 等の HTTP クライアントに代表されるような既存の HTTP ソフトウェアは，文書アクセスと文書表示を中心に開発されている．このため Web サービスなどによるプロセス間通信には不要な機能によってソフトウェアが巨大化しており，計算機資源が少ない機器に組み込むのは適していない．以上の問題を解決すべく，クライアント・サーバ両機能を持ち，HTTP 通信実行に特化した，小型 HTTP コミュニケータ SMACH（Smart Communicator for HTTP）を設計，実装した．

本論文は，アーキテクチャ及び処理方式設計の面から，設計アプローチと具体的な解決方法を述べる．また，開発した SMACH に対して性能検証を行うことでアーキテクチャの妥当性を評価した結果を示す．

2. SMACH の開発方針と機能概要

（1）開発方針

以下を主要な方針として，設計・開発した．

- HTTP 通信実行部分に特化する．
- HTTP 通信より上位の機能（文書処理，ファイル処理，SOAP 処理など）を，アプリケーションプログラム（以下 AP）内で行えるに十分な API を準備する．
- クライアント側，サーバ側の両方をサポートする．
- 双方向通信を実行しやすいように，クライアント側 AP とサーバ側 AP との API に対称性を持たせる．
- 最新のプロトコル仕様である HTTP/1.1 に準拠する．
- C 言語で開発を行う．（Java/VM など特定の環境を前提としない）

（2）概観

図 1 に SMACH の概観を示す．SMACH 本体は，クライアント側とサーバ側に分かれており，その間は HTTP/1.1 で通信が行われる．AP は，SMACH で定めた API によりインタフェースを持つ．なお，SMACH のクライアント機能，サーバ機能はそれぞれ既存の HTTP サーバ・クライアントとも接続可能である．クライアント機能を利用する場合，SMACH はライブラリとして利用される．サーバ機能を利用する AP には複数の形態があり，

- ライブラリの関数として実行
- SMACH サーバの子プロセスとして実行
- SMACH サーバとは別のプロセスとして実行

から選べるようにする．

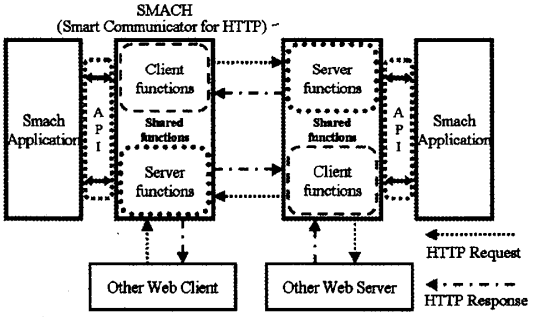

図 1 SMACH 概観

（3）機能概略

一般的な HTTP クライアント・サーバの機能と SMACH のサポートを表 1 に示す．前述の設計方針に基づき，SMACH は HTTP メッセージの作成，セキュリティや認証に関する処理，ログ収集などの機能を備えている．しかし，文書処理を主体とした HTTP サーバやブラウザでは必須の

- クライアントからの要求により，URL で指定されたファイルを送信する
- HTML ドキュメントをレンダリングして表示する

といった機能は実装しない．代わりにこれらの機能を実現する AP に対して必要な情報を受け渡しするための API を提供する．キャッシュ，クッキーなどは API を用いて AP 側での実装を可能とした（これらを表 1 中の△で示す）．

表 1 サポート機能

機能	実装
サーバ	
リクエストライン，ヘッダの解析	○
ステータスライン，ヘッダの設定	○
URLで指定されたファイルの取得と返答	△
CGI	△
BASIC認証	○
ロギング	○
クライアント	
リクエストラインの設定	○
ステータスライン，ヘッダの解析	○
HTMLファイルのレンダリング機能	△
共通	
コンテンツネゴシエーション	○
持続的接続	○
チャンク形式	○
キャッシュ	△
クッキー	△

† 北海道大学大学院情報科学研究科

3. SMACH の方式設計

3.1 アーキテクチャ設計

アーキテクチャでは以下を方針として設計した．

- 小型とする

SMACH に必要な処理を HTTP/1.1(RFC2616) から具体化し，クライアント/サーバへ切り分け，共通処理を抽出した．共通処理を抽出することにより，モジュール間での冗長な処理部分を削減し，小型化を実現した．

- モジュール間の結合を疎とする

各モジュールが複雑に関係を持つのではなく，できる限りシンプルに関係を持つこと，また，それぞれのつながりが疎となるように設計を行った．

以上を考慮した結果，図２のようアーキテクチャ設計を行った．上がクライアント，下がサーバの設計である．また，それぞれのモジュール担当処理を表２示す．

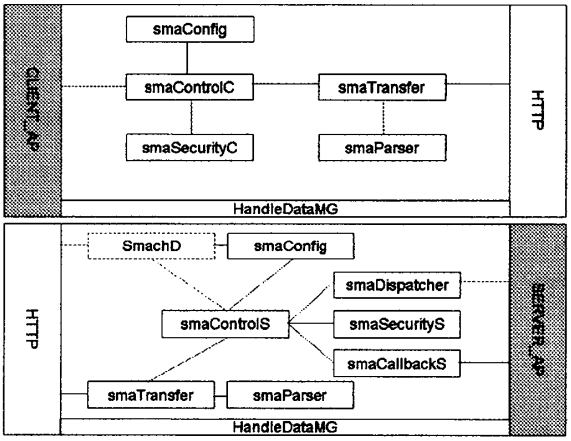

図２　アーキテクチャ設計図

クライアント/サーバそれぞれ smaControlC/S により，プロトコル制御を実現している．プロトコル制御部分がハブになることで，今後新たな処理を加える場合に拡張が容易となる．HTTP の性質上，送受信，メッセージのデータ構造がクライアント/サーバで対称であることから，処理の共通化が可能となり，共通モジュールでの実装を実現した．共通化できない部分に関しては，それぞれで独自のモジュールを利用している．

表２　モジュール担当処理

クライアントモジュール	
SmaControlC	クライアントのプロトコル制御
SmaSecurityC	セキュリティに関する処理
サーバモジュール	
SmachD	TCP 接続確立，スレッド生成
SmaControlS	サーバのプロトコル制御
SmaDispatcher	アプリケーションのディスパッチ
SmaCallbackS	コールバックの制御
SmaSecurityS	セキュリティに関する処理
共通モジュール	
smaTransfer	送受信制御
smaParser	構文解析
smaConfig	Config 設定
HandleDataMG	データの GETTER，SETTER

3.2 処理方式設計

処理方式設計として主に以下ものがあげられる．

- サーバの多重化

複数リクエストに応答するため，サーバの多重化を行った．多重化の方法として

- プロセスを用いる方法
- スレッドを用いる方法
- プロセス，スレッド両方を用いる方法

がある．スレッドを生成する方法は，親となるプロセスやスレッドに異常が発生すると同じプロセスに関連するスレッドすべてが影響を受けるため，他の方法と比べ安定性に劣るが，

- メモリ消費量が小さい
- 起動/停止の時間が短い

という利点がある．SMACH では小型軽量の観点からスレッドを用いる方法を選択した． 多重化の処理方式を図３に示す．SmachD が smaControlS をスレッドとして起動することで smaDispatcher，smaSecurityS，smaCallbackS，smaTransfer，smaParser，HandleDataMG を多重化している．

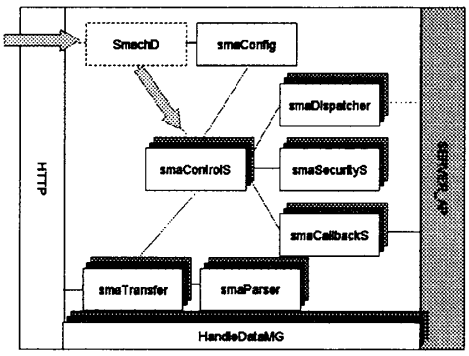

図３　サーバの多重化処理

- アプリケーションからの Callback

SMACH 上のサーバ AP は SMACH からライブラリとして呼び出される．しかし，サーバ AP は SMACH 内部の機能を利用し，レスポンスを行う必要がある．この実現方法として，Callback 関数を採用した．図４のように smaDispatcher からがサーバ AP を呼び出す際に，SMACH サーバの API への関数ポインタの配列 Function Vector をサーバ AP に渡すことにより，callback を実現している．smaCallbackS において，Function Vector の作成，callback 時の処理を行うように作成した．

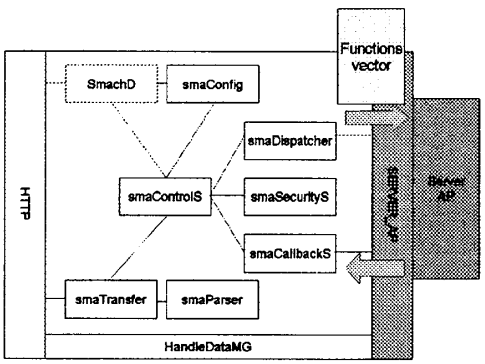

図４　アプリケーションの Callback 処理

- 持続的接続

SMACH では持続的接続[3]を行っているが，可能な限り TCP 接続の切断を HTTP ヘッダに Connection : Close 付加することで実現した．TCP の切断を Connection : close で行うことにより，サーバは適切なときにスレッドを終了することが可能となるため，小型に適した構造となった．

4. 性能検証

実験機器と OS

実験機器，また OS は以下のとおりである．

サーバの実験機器と OS

- OS : Fedora Core 3(2.6.10-1.760-FC3)
- メモリ : 384MB
- CPU:Athlon(tm)XP1700 + 1466.601MHz

クライアントの実験機器と OS

- OS : Redhat 9(2.4.20-8)
- メモリ : 192MB
- CPU:Pentium III Coppermine(597.416MHz)

OS には SMACH が主に Linux を対象にしている点，かつ，比較的よく使われているディストリビューションである点から Fedora Core 3，Redhat 9 を選出した．

4.1 クライアント性能比較

Java との性能測定を行った結果を図 5 に示す．グラフは縦軸にレスポンスタイム（msec），横軸にデータ量（byte）をとったものである．SMACH は Java と比較し 36%～65%の性能向上の結果が得られた．SMACH に性能の優位差が得られた点に関し，Java は VM 上で動作することが上げられる．つまり，Java は VM が存在する必要があり，小型機器では適さない側面が存在する．また，Java がデータを一度 Unicode にエンコードしている点もオーバーヘッドとなり性能劣化を引き起こしていると推察する．

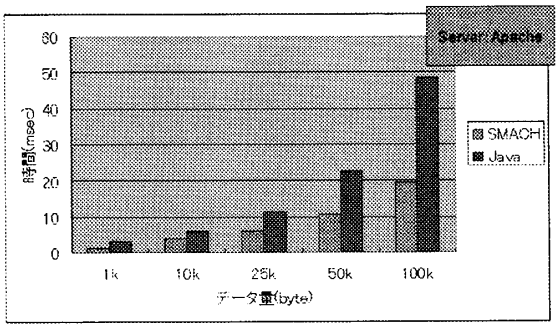

図 5　クライアントの性能比較

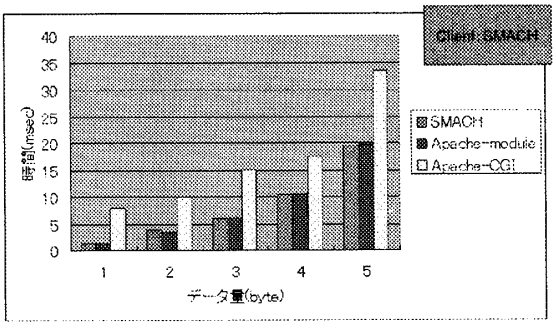

図 6　サーバの性能比較

4.2 サーバ性能比較

サーバの性能測定を行った結果を図 6 に示す．グラフは縦軸にレスポンスタイム（msec），横軸にデータ量（byte）をとったものである．ApacheCGI との比較で 16%～60%の性能優位，Apache モジュールとの比較で同程度の性能であるという結果が得られた．SMACH AP が静的に SMACH から呼び出されている状況から，ApacheCGI と比較する場合 CGI 起動のオーバーヘッドは無視できないが，一般的に Apache でサーバ AP を作成する場合には CGI が利用されるため，有益な比較結果である．Apache モジュールに対し同程度の性能が得られたことに関して，Apache モジュールの開発は一般技術者には難しい点があるため SMACH を利用するメリットがあるといえる．

4.3 小型化に関する比較

開発方針である小型化に関してソースファイル数，ソースライン数，実行ファイルサイズを Apache，thttpd を対象とし性能比較を行った．結果を表 3 に示す． SMACH，Apache，thttpd が持つ機能に違いがあるため単純に SMACH が最も小さいとはいえないが，SMACH はクライアント/サーバ両方を含んだサイズである点も考慮すると目標方針が達成できたといえる．

表 3　小型に関する性能比較

software	Version	source files	source lines	exe(byte)
SMACH	1.0b	20	6697	46,385
thttpd	2.25b	21	11129	85,596
Apache	2.0.53	608	264505	1,222,809

5. おわりに

今回，開発方針にそって SMACH の方式設計を行った．方式設計の過程でアーキテクチャ設計，処理方式設計を行った．アーキテクチャ設計で SMACH の小型化，モジュール間の疎結合を実現した．処理方式でサーバの多重化，アプリケーションからの callback，持続的接続などの開発方針を満たすための各種の処理方式を設計実装した．性能評価により，SMACH の性能向上からアーキテクチャ設計，処理方式設計の妥当性を示せた．

課題として，SMACH の性能を体系的に分析し，性能の向上を図ること，また，HTTP 自体の性能分析を行っていくことが挙げられる．SMACH は純国産のオープンソースとしてスクラッチから開発した．今後，さらに改良を図るとともに，利用普及を進めて生きたい．

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.

[3] Venkata N.Padmanabhan, Jeffrey C.Mogul, Improving HTTP Laytency,book Computer Networks and ISDN System volume 28 page 25-35, 1995,Dec, Slightly revised version of paper inProc. 2nd International WWW Conference’94: