2006－DD－53（3）
2006／1／27

分野横断アプリケーション統合に向けた Web サービスのプリミティブ化に関する実装と考察

佐々木靖彦、村山隆彦*、多田好克
電気通信大学、大学院情報システム学研究科
*日本電信電話株式会社、NTT 情報流通プラットフォーム研究所

多様な分野に対し横断的にアプリケーション統合することを支援する“ Web サービスのプリミティブ化技術 ”に関する検討を行う。本技術は、性能が著しく低いコンピュータ群を使って、XML や SOAP などを用いたメッセージを処理するための基盤技術となる。本稿では、Web サービスのテクノロジスタックを整理した上で、比較的リソースが限定された組込みボード上に Web サービス単独の機能を実装し、処理性能、メモリ使用量、通信データ量についての定量的な解析を行った結果を示す。得られた知見は、（１）処理性能に関しては、Web サービス階層における処理が大きな比率を占めること、（２）メモリ使用量に関しては、約 4MB の実メモリを下限として動作可能であること、（３）通信データ量に関しては、コアデータに対する肥大化率が 3 倍～50 倍にも及び、特に Web サービス階層でのデータ肥大化率が著しいこと、である。今後、関連する課題の解決には、Web サービス階層での新しい技術開発が必要である。

The Implementation and Analysis of Web Service Primitive for Application Integration over Wide Industry Fields

Yasuhiko Sasaki, Takahiko Murayama*, and Yoshikatsu Tada
Graduate School of Information Systems, The University of Electro-Communications
*NTT Information Sharing Platform Laboratories, Nippon Telegraph and Telephone Corporation

We implemented and analyzed web service primitive that enables application integration over wide industry fields. This technology provides the basic platform for a group of low-performance computers to handle XML and SOAP messages. In this paper, after reviewing the technology stack of web services, an implementation example of the web service primitive developed on a relatively resource-constrained embedded-computer-board is shown. The detail analysis on the processing performance, the memory usage, and the amount of communication data is provided. We found that 1) the processing time used for the web service layer occupies the large portion of whole processing time, 2) 4-Mbyte memory is required only for web services functionality, and 3) the data size is inflated to 3 to 50 times that of the core data by conforming it to web services messaging rule. To solve these problems, a new technology especially in the web service layer is demanded.

1　はじめに

Web サービスは、XML、SOAP（Simple Object Access Protocol）、WSDL（Web Services Description Language）といった要素技術をベースとして、ネットワーク上の様々なサービスを統合処理することを可能とする技術として注目されてきた[1]。これは、各要素技術が持つ高い柔軟性（XML による任意タグや WSDL によるインタフェース定義）や高い標準性（SOAP によるエンベロープやスキーマ）を活用することにより、組織の壁を超えたオープンなサービスの連携技術（Service Oriented Architecture）として、新しい応用を生み出す可能性があるからである（図１）。

しかしながら、Web サービス技術の現状を捉えると、実際には当初期待されたほど普及している状況にはない。例えば、UDDI（Universal Description, Discovery and Integration）へのサービス登録数は頭打ち状態にあり、事実上うまく機能しているとは言えない[2]。また、応用事例を見ても、技術提案当初からの定型的かつ固定的な例が多く、やはり技術の普及がうまく進んでいないことを示している。

このような状況の中、Web サービスに関わる技術を従来以上に活用していくためには、新しい展開を検討する必要がある。そうした検討には多様なアプローチが考えられるが、その中でも我々は特に、サービスの粒度に着目したアプローチを検討している。すなわち、提供するサービスの粒度を従来よりも著しく細かくするような基盤技術として、Web サービスのプリミティブ化を検討している。

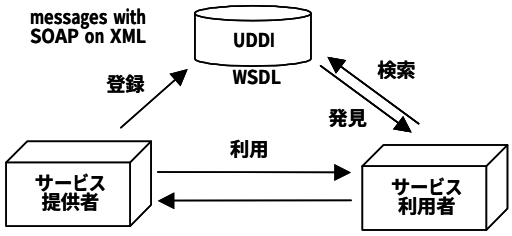

図 1　Web サービス

従来のようにサービスの粒度が大きいと、サービスインテグレータにとって、組み合わせて利用できるサービスの数が限定的となりがちで、かつ統合後のコストが上昇しがちである。本技術は、このような問題を解決しようとするものである。すなわち、粒度を細かくすることで、インテグレータにとって冗長性が低いサービスだけを低コストに組み合わせることを可能とする。その結果、採算性の高い統合サービスを提供しやすくなる。

このような状況は、Web サービス技術において頻繁に引き合いに出されるような旅行業務の統合といったような粒度の大きいアプリケーションの統合とは大きく異なるものである（図２）。これは、従来の情報・

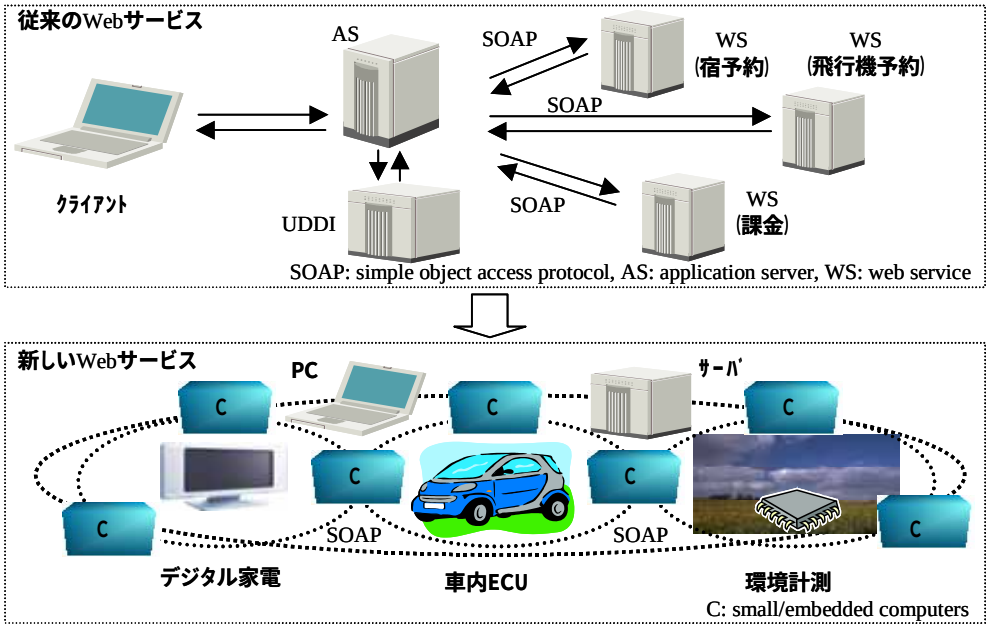

図 2　Web サービスの細粒度化とそれに伴うアプリケーション統合範囲の変化

ソリューション分野における Web サービス技術に留まらず、コンシューマ分野、産業機械分野などにおけるアプリケーションに対しても横断的に関わる Web サービス技術として展開可能となることを意味する。

さて、Web サービスのプリミティブ化にとって、まず最初に解決しなければならないことは、処理性能の低いコンピュータを用いても Web サービスのテクノロジを利用できるようにすることである。これは、現在の Web サービスで利用されているサーバや PC のように数 GHz の周波数で動作する CPU やマルチプロセッサのような高性能なコンピュータではなく、それらより 2 桁から 3 桁下の性能のコンピュータであっても、SOAP や XML といった技術を用いてサービスの提供が行えるようにすることを意味する。

本稿では、このような Web サービスのプリミティブ化を実現しようとする際に、何が問題となり、今後何を開発する必要があるかについて分析した結果を示す。以下、第 2 章では、Web サービス技術の構成要素について階層的に整理し、各階層での主要処理について概説する。第 3 章では、現在の Web サービスとして利用可能な技術を組み合わせることで可能な、比較的コストを低く抑えた Web サービスについて、実装および評価した結果について考察する。最後に第 4 章では、前章での評価結果をもとに、現状の Web サービス技術のプリミティブ化に関する主要な問題点と今後の方向性についてまとめる。

2　Webサービス技術の階層展開

2.1　3大階層

Web サービスは、多くの階層的な技術を積み上げることにより実現される（図 3）。これらの階層は、その処理の特徴から、大きく 3 つの階層として括ることが可能である。すなわち、(1) Web サービス階層、(2) 通信プロトコル階層、(3) 物理階層である。Web サービスの細粒度化を検討するにあたっては、これらの階層のどの部分が性能やリソースの観点から問題となるかについて明確にし、その結果に基づき、問題の大きい部分から順に対処していくのが望ましい。

さて、3 大階層を、上位のレベルから順におおまかに記述すると、以下のように説明できる。

1) Web サービス階層：　Web サービスに関係の深い機能を実現するために必要となる部分である。具体的には、SOAP 処理、XML 処理、HTTP 処理などがこれに該当する。

2) 通信プロトコル階層：　汎用的な通信上のプロトコルを定義する部分である。例えば、IP/ICMP、TCP/UDP、Socket などがこれに該当する。

3) 物理階層：　有線や無線の各種の信号伝送媒体に直接的に関連する部分である。例えば、Ethernet のリンク層、ファイ層などが含まれる。

2.2　サブ階層

次に、3 大階層の各階層における処理の詳細を見

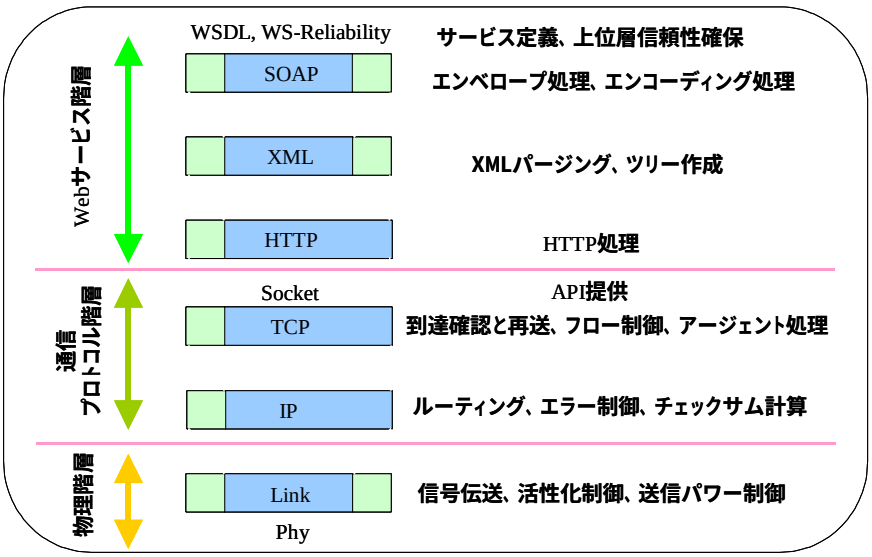

図 3　Web サービスを実現する技術階層とその処理概要

る(図３に､各階層の処理概要についても示している)。

２.２.１　Webサービス階層

XML 処理、SOAP 処理、HTTP 処理、WSDL 処理などが含まれる｡中心となるのは､SOAP 処理､XML 処理、HTTP 処理である。全ての処理について共通に言えることは、いずれも扱っているデータはテキストデータである。HTTP 処理では、サーバに送られてきたメッセージのヘッダの内容に従い情報の取得や返送を行う。XML 処理では、タグの解析を行いドキュメントの構造を決定する。SOAP 処理では、エンベロープ処理、エンコーディング処理を行う。また、WSDL 処理では、サービス定義の読み込み、プロトコルバインディング､ネットワークバインディングなどを行う。

２.２.２　通信プロトコル階層

PC・サーバ系では IP/ICMP 層、TCP/UDP 層、Socket 層が対応するが､物理階層に応じて方式が代わることが多い。各層における処理の詳細は多数の文献[例えば、3]に記載されているため省略するが、主たる処理はルーティング、到達確認と再送処理、フロー制御、チェックサム計算、上位 API の提供などである。通信プロトコル階層はアプリケーションにより大きく変化する可能性がある。その場合、処理内容によっては､Web サービス階層との間での最適な振り分けも考慮する必要がある。例えば、信頼性の確保などは、通信プロトコル階層で行うに限らず、より上位層で取り扱うアプローチも有り得る。このような場合、SOAP ヘッダを活用した WS-Reliability [4]などを利用することが考えられる。

２.２.３　物理階層

有線方式では Ethernet をはじめ､IEEE1394､USB､I2C、CAN など、また無線方式では、IEEE802.11 系, 802.15 系など多様な方式が存在する。この階層では、物理的な信号伝送はもとより、活性化制御や送信パワー制御などが含まれる。

３　細粒度Webサービスの実装試行と評価

Web サービスの細粒度化を実現する上での課題を整理するために､Web サービスを極力コンピュータリソースが限定された組込みボード上に実装し、評価を行った。評価内容は､(１) 処理性能 (処理時間)､(２) メモリ使用量、(３) 通信データ量、の３つである。

評価に先立って、表１に示されるように組込みボードに Web サービスの機能である３大階層の処理を実装した。３大階層のうち､特に Web サービス階層に関しては、テキスト処理技術で広く利用されている Perl を用いて実装を行った。

表 1　Web サービスのベースシステム

CPU/RAM(server)	SH4(240MHz)/32MB
OS	CELinux (2.4.20)
HTTP	独自実装
XML	XML::Parser
SOAP	独自実装
ネットワーク (chip)	TCP/IP, Ether, 10Mbps (SMSC 91C111)
参考：client CPU/RAM	Pentium4(2.99GHz)/1GB

そもそも、本稿の目的は Web サービスに関する細粒度化を検討することであるから､大量のリソース(メモリや CPU 性能等) を前提とした http サーバ (Apache 等) や各種 Web サービスを備えたフレームワーク (.NET や JAX-RPC 等) を利用することは行えない。したがって、上記実装においては多くの部分で独自実装を行い、極力少ないリソースで動作するように工夫したものを用いている。また、従来一般に用いられているような HTTP 処理と XML 処理や SOAP 処理が別プログラムとなっている構成ではなく、HTTP 処理から SOAP 処理までの階層を一つのプログラムとして構成した。これは、従来の方式で見られるような HTTP サーバから外部プログラム(XML 処理、SOAP 処理)を起動する際に生じる処理時間の増加や、別プログラムによるコードサイズの増加といった問題を解決するためである。

３.１　分析の概要

処理性能の評価では、Web サービスを実行させた上で、これに対しプロファイラを用いたボトルネック解析を行った。メモリ使用量に関しては、Web サービ

スのプログラム実行中に ps コマンドを発行して、動作中の複数のタイミングでのメモリ使用量を測定した。また通信量に関しては、Web サービスを SOAP メッセージに載せて転送する場合のデータ量の肥大化について分析した。以下、処理性能、メモリ使用量、通信データ量のそれぞれに関する分析について順に示す。

3.2　処理性能に関する分析

3.2.1　分析対象と方法

PC と組込みボードとをネットワーク接続し、クライアントマシン（PC）から Remote Procedure Call（RPC）を行い、サーバ側（組込みボード）の実行性能を計測した。計測では、Web サービスプログラムを走らせたマシン上でプロファイラ（DProf）を使って処理時間の比率を測定した。使用した RPC は、最もプリミティブな Web サービスの処理時間の下限を知るために、単一のストリング型パラメータを受け取り、単一の実数データを返送するというものを用いた。ここでは、Web サービスの機能提供だけに着目した評価を行うために、実アプリケーションでは存在する各種の内部サービスとしては何も処理を行わず、予め用意された規定値を返すだけにしてある。

各処理の概要は以下のとおりである。

1）HTTP 処理：　リクエストヘッダを分離した後、ボディ部を XML 処理部に渡す。

2）XML 処理：　XML ドキュメントのパージングを行い SOAP 処理部に結果を渡す。

3）SOAP 処理：　SOAP の RPC メッセージを解釈した後、それに応答する適切な SOAP レスポンスを返信する。

通信プロトコル階層以下の処理については、OS および半導体チップに搭載されている機能を利用した。また、計測にあたっては、ネットワークの状態などによるばらつきを抑え、またシステム起動時の準備時間を含めないようにするために、1000 回の RPC を行い平均時間を取得した。

3.2.2　分析結果と考察

計測結果を図４に示す。同図は、単一クライアントからのアクセスの場合を示したものである。３大階層の中で処理時間を大きく占めているのは、Web サービス階層である。計測結果の表の上位を占めているのは、HTTP 処理、XML 処理、SOAP 処理であることがわかる。通信プロトコル階層以下では処理時間の数%しか消費しておらず、あまり問題とはならない。例えば、SOCKET による socket/bind/listen などの前

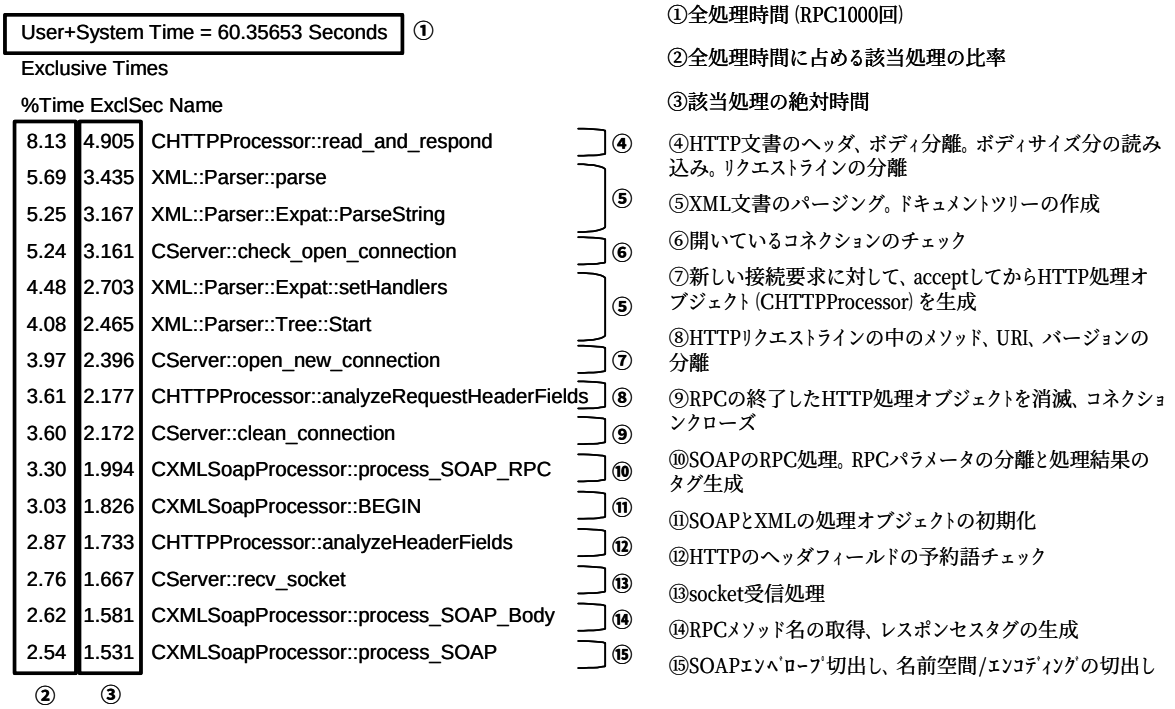

図 4　Web サービス実行に伴う処理時間のプロファイル

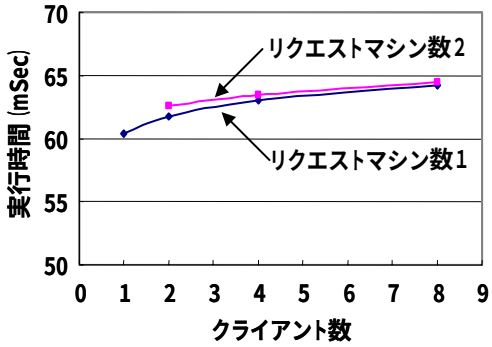

図 5　クライアント数と実行時間の関係

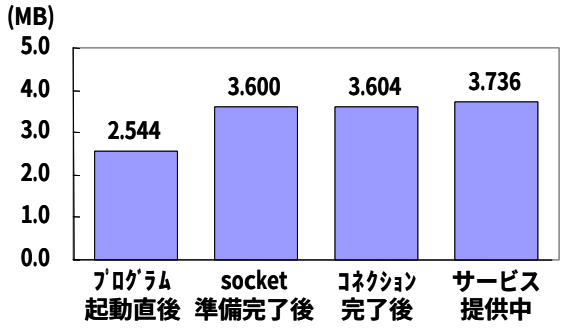

図 6　メモリ使用量の時系列変化

準備（prepare_socket という関数になっている）はリストに登場せず、また recv/send 処理も 3%以下である。

クライアント数を増加させたときの結果を図５に示す。リクエストを出す側のマシンが、１台の場合と 2 台の場合について、サーバ側での処理時間を示している。通信プロトコル階層の処理時間増加はあるものの、Web サービス階層で処理時間の大きな比率を占めるという状況は、大きくは変化しない。

処理時間の絶対値で見ると、１回あたりのサービス時間は約 60msec であることがわかる。組込みボードとしては比較的性能が良いもの（240MHz 動作）であっても、大きな時間がかかっている。先に記したように、現在は単一パラメータの SOAP-RPC 処理のみだが、より複雑な RPC の場合等を想定すると、処理時間は一層増加することが予想される。さらなる“プリミティブ化”で必要とされる上記より１桁～２桁性能が低いコンピュータで Web サービスを実現しようとすれば、数百ミリ秒～数秒という大きな時間を要してしまう。ノード本来のサービス処理を含めない Web サービスのためだけにかかる時間としては非常に大きい。

３.３　メモリ使用量に関する分析

３.３.１　分析対象と方法

メモリ使用量に関する分析の目的は、最低限の Web サービス機能を実行しようとする場合に、どの程度のメモリを必要とするかの知見を得ることにある。現在の Web サービス提供環境として比較的自然な実装ではあるが、できるだけリソースを減らした構成でのメモリ使用量が判断できる。

メモリ使用量の測定は、先に性能分析のところで述べたものと同一の実行条件で行った。具体的には、プログラム中の幾つかのタイミングでプロセス状況を見るための ps コマンドを発行し、その結果からデータを取得した。

３.３.２　分析結果と考察

図６にメモリ使用量の計測結果を、時系列の形で示す（左から右へ）。なお、ソケット処理でのバッファサイズは 8KB としてある。

計測結果の要点を以下にまとめる。

1) プログラム起動直後のメモリは約 2.5MB である。
2) ネットワーク処理（socket）の準備に必要な実メモリは約 1.1MB である。
3) 一つのコネクションあたりに必要な実メモリは約 0.04MB である。
4) サービスの提供に必要な総実メモリは約 3.7MB である。

メモリの必要量という観点で見ると、複数のクライアント（１０アクセス以下）からの接続がある状況でも、およそ 4MB があればよいことになる。この値は、Web サービスを提供するノード開発時のシステム設計における一つの目安として利用することが可能である。

３.４　通信量に関する分析

３.４.１　分析対象と方法

通信量に関する分析の目的は、データ量がどの程度肥大化するのか、また、3 大階層（さらにはより詳細な階層）のどの部分でデータの肥大化が著しいかについての知見を得ることである。データの肥大化は、複数のノードを持つシステム内でノード間のデータ転

送に影響を与え、システムの性能や運用性に大きく関係する。したがって、より効果的にデータの肥大化を抑制する方法を探るために、その原因をブレークダウンする。

先のRPCのSOAPレスポンスメッセージを対象に、データの肥大化を分析した。最上位のSOAP層から一つずつ降りていく形でデータを積み上げ、最終的にどの程度肥大化するかを調べる。

データの肥大化とは具体的には、SOAP層、XML層、HTTP層まで（Webサービス階層）は、テキストによるドキュメントデータ構造の付加やヘッダの付加である。次に、TCP層、IP層まで（通信プロトコル階層）とLink層（物理階層）では、ヘッダとトレイラが付加されることを意味する。物理階層では、高速系のインタフェースにおいては、実際にはビットスタッフィング、8B/10B変換などさらにデータ肥大化の要素が加わるが、ここでは見積りから除外してある。

3.4.2　分析結果と考察

分析結果について表2に示す。表2では、テキスト数16の数値データ（16バイト相当）を送ろうとした場合に、各階層別に前節で示したデータの積み上げを行い、最終的にデータがどの程度肥大化するかを示している。送ろうとしているデータをコアデータと呼ぶことにすれば、データ数が1個の場合、コアデータの全体データに占める比率はわずか1.9%に過ぎないことがわかる。データの肥大化という見方をすれば、約53倍にデータがふくれあがる。

表 2　階層別に見たデータの増大（単位：bit）

階層	ヘッダサイズ	データサイズ	トレイラサイズ
SOAP	3232	128	0
XML	368	3360	648
HTTP	1992	4376	0
TCP	160	6368	0
IP	160	6528	0
Ether	112	6688	32
合計	6832（コア占有率 128 / 6832 = 1.9%）		

ただし、これは単一データだけをやりとりする

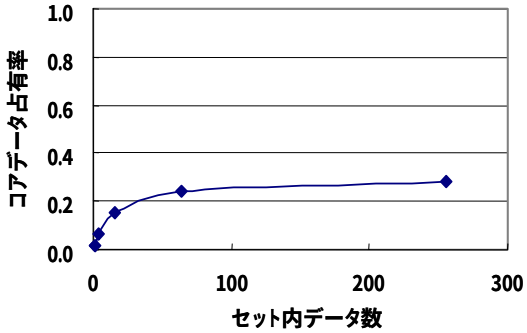

図 7　データ数とコアデータ占有率の関係

RPCであって、最も効率の悪い場合である。効率の向上を行おうとした場合に、複数のデータをセットの形でまとめてやりとりするケースも想定される。図7は、セット内データ数を1～256まで増加させたときの様子を示している。先の単一データの場合と比べて、大きくデータの増加率を削減できていることがわかる。しかしデータ数が増加するにつれやがて飽和し、最終的に肥大化は約3倍程度になる。

図7においてセット内データの数が小さい部分に着目した場合、データ数が16以下ではコアデータの占有率は16%以下である。このようなケースは実際によく利用される領域と考えられるが、肥大化率は6倍以上であることがわかる。

図8は階層別に見たときにデータの肥大化がどこで著しく生じているかを示している。単一データレスポンスの場合には、SOAP、XML、HTTP層でのデータ肥大化が著しい。また、セット内データ数256の場合にはSOAPとXML層での肥大化が大きいことがわかる。いずれにしても、データ肥大化の原因のほとんどはWebサービス階層であることがわかる。

以上をまとめると、次のようになる。

1）ノード間のデータ転送量を削減するためには、できるだけ大きな単位をセットにしたレスポンスメッセージを構成することが望ましい。しかし、これでもデータの肥大化率は3倍以上である。

2）実応用で少量のデータでレスポンスメッセージを返送しなければならないケースではデータの肥大化率は6倍から53倍にもおよぶ。

3）肥大化の主たる原因はWebサービス階層にある。

さて、データ肥大化率に関してかなり大きな結果となることを示したが、ここでは注意も必要である。

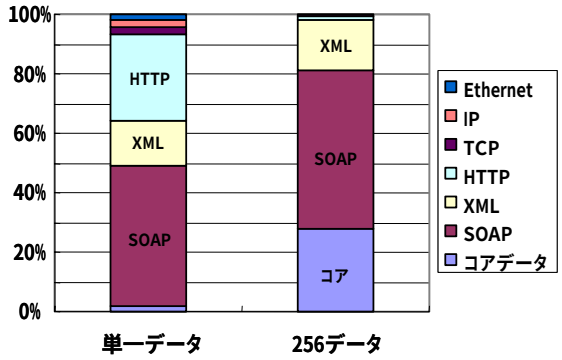

図８　階層別に見たデータ量占有率

つまり、コアデータと呼んでいるものだけが SOAP メッセージの運んでいる情報ではないということである。例えば名前空間やエンコーディングなどの情報（スキーマ情報）も SOAP メッセージの中には一緒に埋め込まれている。つまり、上述の比較はこの情報が欠落しているコアデータとの比較を行っているために、このような程度の大きい肥大化として見えている。全く同一条件で比較するには、これらを考慮しなければならない。

しかしながら、プロプライエタリ（クローズド）なシステムの場合には、このような名前空間やエンコーディングの情報があらかじめ理解済みとして不必要であり、それとの比較といった見方をすれば、先の肥大化率は妥当である。つまり、クローズドなシステムとも競争できるオープンな統合システムを構築しようとすれば、上述の肥大化は大きな課題であると言える。特に、分野横断的なアプリケーション統合まで意識するなら、この点は避けて通れない。また、クローズドで短命なシステムと比較して、オープンかつ長命なシステムのメリットは大きいため、データ肥大化率に関する課題を解決することはやはり重要である。

４　結論

Web サービスの細粒度化を実現する上での課題を整理すべく、そこで用いられるテクノロジスタックを整理した。さらに、比較的リソースが限定された組込みボード上に Web サービス単独の機能を実装し、処理性能、メモリ使用量、通信データ量について定量的な解析を行った。

その結果、処理性能に関しては、Web サービス階層での処理が大きな比率を占めることがわかった。またメモリ使用量に関しては、約 4MB の実メモリを下限として動作可能であることがわかった。通信データ量に関しては、できるだけ大きな単位でレスポンスメッセージを構成することが望ましいが、一般には肥大化率は 3 倍～50 倍におよぶ。階層別に見た場合では、やはり Web サービス階層でのデータ肥大化率が著しいことがわかった。

プリミティブ化に向けた総合的な意味での改善を考えると、処理性能や通信データ量の観点からも Web サービス階層での対策が最も効果的である。したがって、今後、Web サービス階層での新しい技術開発が必要であると考えられる。

参考文献

[1] http://www.w3.org/2002/ws/

[2] Web サービス最新動向調査、GMO 総合研究所、シードプラニング編、2003 年

[3] K. Washburn and J. Evans, “TCP/IP: Running a Succesful Network,” Addison-Wesley Publishing Company, 1993.

[4] OASIS WSRM TC, “WS-Reliability Ver1.1,” http://www.oasis-open.org/specs/ index.php, 2004.