社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－SE－147 (10)
2005／3／18

SMTPを用いた非同期型Webサービスの提案と評価

森 晃 青山 幹雄

南山大学 数理情報学部 情報通信学科

本論文は，Web サービスを非同期で利用するための非同期型メッセージングの方法を提案する．現在，分散システムの基盤として普及しつつある Web サービスの多くは同期型であるため，サービスリクエスタはリクエスト送信後，結果を待機するという問題がある．本論文では，トランスポートプロトコルに SMTP を用いた，SOAP over SMTP による非同期型メッセージングの方法を提案する．SOAP over SMTP により，待機時間の少ない，効率の良いサービス利用が可能になる．これを実現するために，Apache Axis を拡張し，Apache James の Mailet 機能を用いて SOAP over SMTP を実装した．例として PDF 変換サービスを試作し，同期型に対する非同期型 Web サービスの有効性を示す．

Asynchronous Web Services Architecture over SMTP and its Evaluation

Akira Mori Mikio Aoyama

Department of Information and Telecommunication Engineering,
Faculty of Mathematical Sciences and Information Engineering, Nanzan University

This article proposes a method of asynchronous Web services. The Web services have been spreading to decentralized systems over the Internet. However, the service requesters have to wait for the reply of request message if the Web service protocol is synchronous. This article proposes the asynchronous Web services with SMTP for transport protocol. To realize asynchronous Web services protocol, we propose a design of SOAP over SMTP with extension of Apache Axis and Mailet of Apache James. We implement PDF Conversion Service as an example of Web services, and evaluated effectiveness of the asynchronous Web services protocol against the synchronous one.

1. はじめに

Web 技術の進歩により，情報社会の多くのサービスが Web 上で提供されている．その中で Web サービスは，XML (eXtensible Markup Language)形式のメッセージ交換によってネットワーク上の自律したアプリケーションを連携することで，企業間のコラボレーションを実現している[2]．

Web サービスによって，Web 上に点在する種々のサービスを組み合わせることによって，まったく新しい，複合的サービスを提供することが可能となる．

現在普及している Web サービスの技術には，トランスポート層に HTTP(HyperText Transfer Protocol)を使用した，SOAP(Simple Object Access Protocol) over HTTP が広く利用されている．この SOAP over HTTP を用いた同期型 Web サービスでは，リクエスト/レスポンス型のメッセージングを行うため，クライアントはサービス要求のメッセージを送信後，プロバイダでサービスの処理を行い，処理結果が返信されるまで，待ち状態になるという特徴を持つ．

同期型 Web サービスは待ち状態中に他のタスクを実行することがないため，要求に対して迅速な応答を必要とするサービスや，短時間で要求を処理できるサービスには有効な方法である[9]．

しかし，プロバイダ側で長時間の処理が必要なサービスでは，クライアントの待ち状態が続き，他のタスクを実行できないという問題が起こる．また，ネットワーク等の障害によりクライアントの要求に対する応答メッセージを返信できない状態にある場合も同様に，待ち状態が長時間化する可能性がある．

さらに，同期型 Web サービスには，応答メッセージを期待しない一方向のみのサービス利用に不向きであるという問題がある．

本論文では，これらの問題点を解決する一方法として非同期型の Web サービスを用いる方法を提案する．本稿では，実際にトランスポート層に SMTP(Simple Mail Transfer Protocol)を用いた，SOAP over SMTP による非同期型 Web サービスを実装し，同期型 Web サービスとの比較を行うことで，非同期型 Web サービスの有効性を示す．

2. 非同期型 Web サービスによる解決策

同期型 Web サービスの待ち状態の長時間化問題を，非同期型 Web サービスを用いて解決する方法を提案する．

非同期型 Web サービスでは，リクエストとレスポンスに関連を持たない一方向型のメッセージングを行うため，

クライアントはサービス要求のメッセージ送信後に待ち状態にならないという特徴を持つ.

非同期型 Web サービスのメッセージ送受信の流れを図 1 に示す. クライアント A はリクエストメッセージ 1 を送信後, レスポンスメッセージ 1 を待機しないため, 複数のリクエストメッセージを連続して送信することが可能となる. さらに, リクエストメッセージ 2 を送信後, プロバイダ C からのレスポンスメッセージ 2 の返信時に Web 上で通信網等の障害が発生しても, クライアント A は同期型のように, 待ち状態が発生し, リソースロックによる他のリクエストが送信できなくなることはない. プロバイダ B,C が長時間の処理を必要とするサービスであってもクライアントが待ち状態にならない, 利用効率の良い Web サービスを提供できると考える.

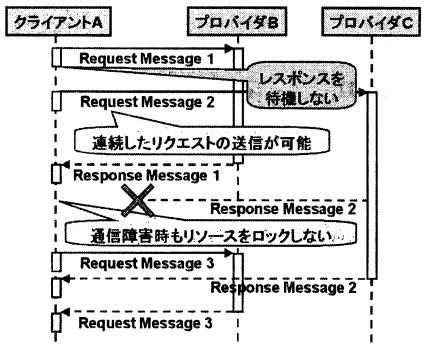


図 1: 非同期型 Web サービス

3. 非同期型 Web サービスの実現

3.1. SOAP over SMTP

Web サービスを利用するためには, プロバイダ側のサービスを呼び出す XML 形式で記述されたメッセージを送受信する必要がある. オブジェクト間のメッセージ連携技術には COM(Component Object Model)や CORBA(Common Object Request Broker Architecture), SOAP などがある. 本論文では, 異なるプラットフォーム間での相互接続が可能で, トランスポートプロトコルにも依存しない SOAP を用いたメッセージ送信を行う.

SOAP メッセージを非同期で送信する場合, トランスポートプロトコルの選択が問題となる. 非同期通信を支援するトランスポートプロトコルには, SMTP や MQ(Message Queue)が挙げられるが, 本研究では以下の理由により SMTP に準拠してメッセージ送信を行う, SOAP over SMTP を実現する.

(1) 一方向のメッセージングが可能
(2) 企業間などでファイアウォールを透過
(3) 既存のメールサーバを使用可能

3.2. SOAP over SMTP による非同期メッセージング

SOAP over SMTP の非同期メッセージングを図 2 に示すアーキテクチャで実現する[7]. Web サービスのプロバイダにはメールサーバを使用し, 電子メールと同様に Web サービス用のメールアドレスを指定して SOAP メッセージを送信する.

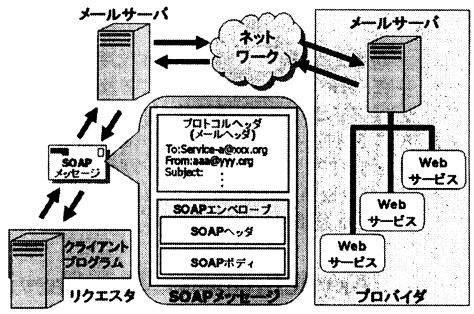


図 2: SOAP over SMTP による
非同期メッセージングのアーキテクチャ

3.3. SOAP メッセージの識別

SOAP メッセージを非同期で送受信する際に, リクエストメッセージを連続して送信すると, プロバイダからのレスポンスメッセージがどの要求に対する返信であるかを識別できないという問題が起こる. これは同期型のリクエスト/レスポンスと異なり, 一方向型のメッセージングをおこなうため, リクエストとレスポンスに対応関係がないことが原因である. そこで非同期型では, リクエストとレスポンスの対応関係を明確にし, メッセージを識別する必要がある.

メッセージの識別には, メッセージに一意に識別可能な ID を付加する方法が必要である. SOAP メッセージへ ID を付加する方法として次の 2 つが考えられる.

(1) SMTP ヘッダへの Message-ID の付加

図 3 に示すように, SMTP ヘッダ部に Message-ID を付加して送信し, 対応するレスポンスメッセージには, リクエストメッセージの Message-ID の値を In-reply-to に付加して返信する.

この方法は, 上位プロトコルの SOAP に依存することなくトランスポート上でメッセージの識別が可能になる.

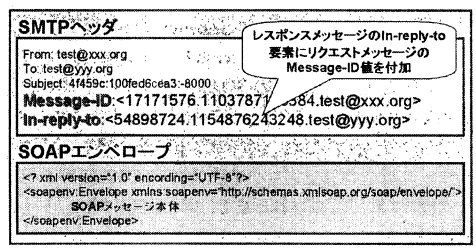


図 3: SMTP ヘッダへの Message-ID の付加

(2) SOAP ヘッダへの ID 要素の付加

図4に示すように，SOAP ヘッダに ID 要素を定義し，レスポンスメッセージにはリクエストメッセージに付けられた ID を付加する要素を定義することにより，リクエストとの対応関係を明確にする．この方法は，一つのリクエストに対する複数のレスポンスに同一のユニークな ID を定義することも可能である．

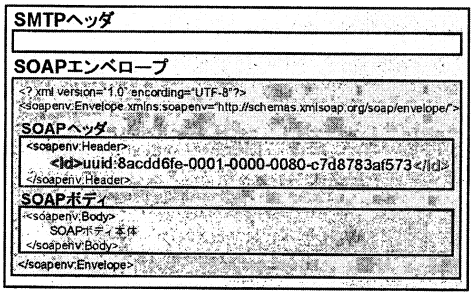


図 4: SOAP ヘッダへの Message-ID の付加

それぞれの識別方法の特徴を表 1 に示す．

本論文では，(2)の SOAP ヘッダに ID 要素を付加する方法を提案する．SOAP ヘッダへ付加することで，トランスポートとは独立し，アプリケーションレベルで完結したメッセージの識別が可能となる．

表 1: メッセージ識別方法の特徴

	長所	短所
SMTP ヘッダ	SOAP と独立にトランスポート上でメッセージ識別が可能	送信エラーによる再送時に Message-ID が変更される
SOAP ヘッダ	SMTP と独立に 1 対多の対応関係，リクエスト/レスポンス以外の関連のあるメッセージの対応関係を表現可能	送信側と受信側の両方で ID を識別する方法が必要となる

4. SOAP over SMTP の実装

4.1. Apache Axis による SOAP over SMTP の実現

SOAP over SMTP を用いた非同期型 Web サービスを実現するために，オープンソースの SOAP 実装である Apache Axis を拡張して SOAP over SMTP を実装した．

4.1.1. Apache Axis のアーキテクチャ

Apache Axis は図 5 に示すように，実際に Web サービスを処理する AxisEngine が，メッセージの処理を行う Handler と，一連の Handler を纏めた 3 つの Chain で構成されている[3]．

クライアント側の Axis はアプリケーションからリクエストを受け取ると，AxisClient（クライアント側 AxisEngine）内で Handler が呼び出され，独立した処理が順に進められる．ターゲットとなるプロバイダへのメッセージ送信に必要な，トランスポートプロトコル特有の操作を実行する TransportSender により SOAP メッセージが送信される．

一方，プロバイダ側では，クライアントからの SOAP メッセージを TransportListener が受け取ると，AxisServer（プロバイダ側 AxisEngine）を起動してメッセージを渡す．各 Handler の処理が進むと Provider と呼ばれる Handler が呼び出され，Axis に配置された Web サービスを実行する．

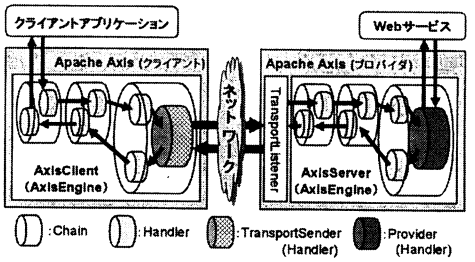


図 5: Apache Axis のアーキテクチャ

次に，クライアントアプリケーションから AxisClient が呼び出されるまでの一連の過程を図 6 のシーケンス図に示す．

クライアント側の Apache Axis の処理は AxisClient を呼び出すまでに，関連する Service, MessageContext, Message を持つ Call オブジェクトの構築を行う．はじめにクライアントアプリケーションは Service インスタンスの createCall メソッドを呼び出し，Call オブジェクトを生成する．次に，Call.setOperation メソッドで適切な Transport インスタンスを生成し，Call.invoke を行う．ここで，MessageContext に関連する Message オブジェクトが生成され，MessageContext に Transport が設定される．最後に AxisClient.invoke を呼び出し MessageContext を AxisClient の各 Handler に渡す．

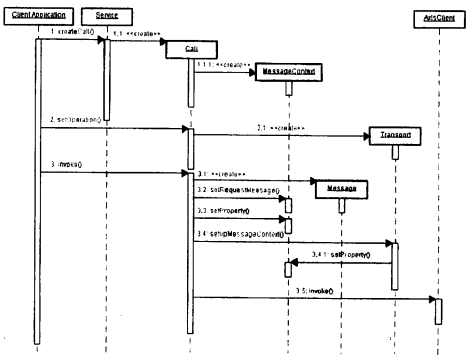


図 6: AxisClient を呼び出すまでのクライアント処理

4.1.2. SOAP over SMTP の実現方法

前節に示したアーキテクチャより，Apache Axis への SOAP over SMTP の実装を行うためは，メッセージの送

受信部をデフォルトの HTTP 用から SMTP 用に変更する．クライアント側の変更点は，MessageContext への SMTP の Transport 設定と，メッセージ送信処理を行う TransportSender，プロバイダ側の変更点は受信したメッセージを AxisServer に渡す TransportListener となる．

SOAP over HTTP では，TransportListener に HTTP サーブレットを使用しているが，SOAP over SMTP ではメールサーバ上で AxisServer を起動しなければならないため，サーブレットは使用できない．そこで，本研究では図 7 に示すように TransportListener に Apache James の Mailet を用いて，メールサーバに届いたメッセージを受信し，AxisServer に渡す方法を提案する．

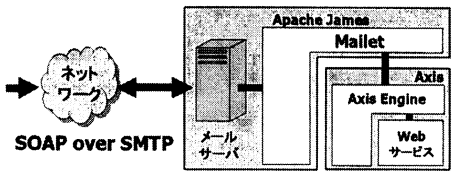


図 7：Mailet によるメッセージの受け渡し

4.1.3. Apache James と Mailet

Apache James は Java で書かれた電子メールサーバエンジンであり，その機能の一つである Mailet は，メールサーバがメールを受信し，設定した条件に適合したときに特定の Java アプリケーションを起動することができる[4]．ある Web サービス用のアドレス宛の SOAP メッセージを受信したときに，AxisServer を起動し，SOAP メッセージを MessageContext として渡すことができれば，SOAP over SMTP で送信された SOAP メッセージからメールサーバに配置された Web サービスを利用できる．

4.2. 非同期型 SOAP メッセージの送信

4.2.1. SMTP 用 TransportSender の実装

SOAP over SMTP で SOAP メッセージを送信するために図 8 に示すように，クライアント側の Apache Axis に SMTP 用の TransportSender である SMTPSender を実装，配置した．

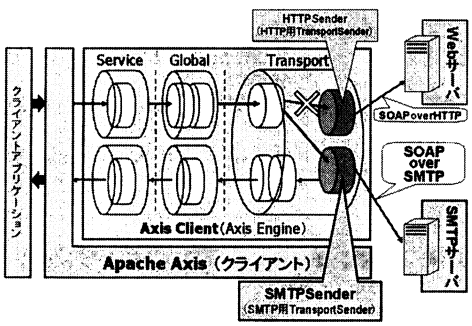


図 8：クライアント側のメッセージパス

SMTPSender は，各 Handler で処理された SOAP メッセージのバインディングヘッダに SMTP ヘッダを付加し，Web サービスを提供するメールサーバに対してリクエストメッセージを送信する処理を行う．

SMTPSender が行う処理の流れを図 9 のシーケンス図に示す．

Apache Axis の AxisClient から invoke メソッドが呼び出されると，SMTPSender は MessageContext オブジェクトを受け取る．SMTPSender は MessageContext から送信元と送信先のメールアドレスを取得し，新規に作成した MIME(Multipurpose Internet Mail Extension) 形式の電子メールメッセージを表す MimeMessage オブジェクトにセットする．最後に MessageContext から SOAP で記述されたリクエストメッセージ本体を取り出し，MimeMessage にバイト列として出力しメッセージを電子メールとして Web サービスプロバイダへ送信する．

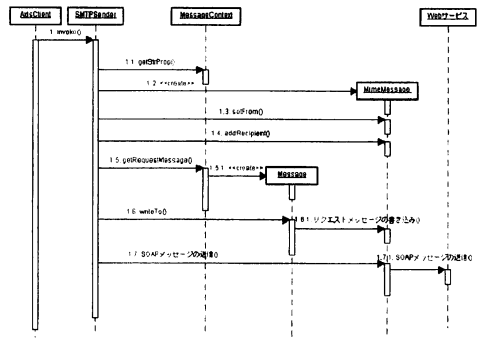


図 9：SMTPSender のシーケンス図

4.2.2. Apache Axis への SMTP 用 Transport の追加

4.1.2 節で提案したように，SMTP の TransportSender を AxisClient から呼び出すためには，MessageContext の Transport 設定を SMTP 用の Transport に変更する必要がある．そこで Transport クラスを拡張した SMTPTransport を作成した．

クライアントアプリケーションは Call オブジェクトを作成後，SMTPTransport のパッケージ名を登録する．さらにトランスポート名"smtp"とそれに対する Transport クラスの SMTPTransport を登録する．それにより，MessageContext の Transport を SMTPTransport に設定し，SMTPSender へ MessageContext を受け渡すことが可能となった．

4.3. Mailet を用いたサービスプロバイダの実装

4.3.1. SMTP 用 Transport Listener の実装

前節で説明した方法を用いて SOAP over SMTP で送信された SOAP メッセージを受信し，AxisServer を起動して Web サービスを呼び出すために，図 10 に示すよ

うに Apache James の Mailet を用いて，TransportListener に Apache Axis 用の Mailet である AxisMailetを実装，配置した．AxisMailetは電子メールとして受信したリクエストメッセージの本体となるSOAPメッセージを MessageContext にセットして，起動した AxisServer の Handler に渡す処理を行う．

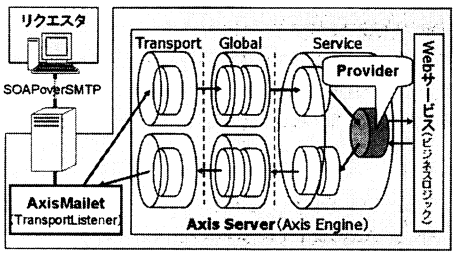


図 10：プロバイダ側のメッセージパス

AxisMailet の処理の流れを図 11 のシーケンス図に示す．

AxisMailet は Apache James から起動されると，Mail オブジェクトを受け取る．最初に Mail から MimeMessage を取得し，AxisServer オブジェクトを作成する．次に，MessageContext オブジェクトと Message オブジェクトを作成し，MimeMessage から取得したリクエストメッセージを Message にセットし，さらに MessageContext に Message をセットする．最後に AxisServer の invoke メソッドを呼び出して AxisServer に MessageContext を渡す．

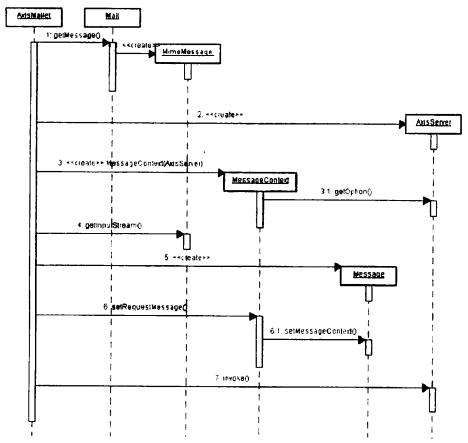


図 11：AxisMailet のシーケンス図

表 2：SOAP over SMTP の開発規模

	プログラム行数	クラスファイル(新規)
クライアント	310	4
プロバイダ	105	1
合計	415	5

前節の SMTP 用 TransportSender の実装を含む，Apache Axis への SOAP over SMTP の実装で作成した Java プログラムの規模を表 2 に示す．

4.3.2. AxisMailet の追加と設定

作成した AxisMailet を Apache James に登録するためには，James の設定ファイルである config ファイルに Mailet のパッケージを追加する．

さらに，AxisMailet が Web サービス用のメールアドレス宛にメッセージが届いたときのみ起動するためには，Mailet の起動条件を記述する必要がある．Apache James には Mailet の起動条件を指定する Matcher インタフェースが実装されており，メールのタイトルや受信者アカウント，送信者アドレスなどによって条件を変更できる．ここでは，受信者が Web サービス用のアカウントの場合のみ AxisMailet を起動する設定にすることで，サーバが受信する他の電子メールと区別する．

4.3.3. Apache Axis へのビジネスロジックの配置

Apache Axis へサービスを配置するには，Apache Tomcat などのサーブレットコンテナ上では Web サービスとするクラスファイルを classes ディレクトリに設置し，設定ファイルにサービスの情報を記述する方法を用いる．

しかし，Apache James は現段階で，james ディレクトリのサブディレクトリ内にクラスファイルや設定ファイルを設置できないので Axis へのサービス登録ができない．

そこで，Apache James 上の Apache Axis にサービスを配置するためには，クラスファイルを jar ファイルへ変換し，lib ディレクトリに設置するという方法を用いる．設置したサービスを Aapache Axis に登録するには，org.apache.axis.server パッケージ内に含まれるデフォルトの server-config.wsdd ファイルに直接サービス情報を記述することで，サービスを配置した．

4.4. WS-Routing を用いた SOAP ヘッダへの ID 付加

3.3 節で議論した SOAP メッセージへの ID の付加を WS-Routing(Web Services Routing Protocol)を実装した WS-Routing4Axis を用いて実現する．

WS-Routing とは，2001 年 10 月に Microsoft によって提案された(現時点では未標準化)Web サービスに関する仕様である[8]．SOAP メッセージ内にメッセージのパスを記述する方法と，メッセージ間の対応付けの機構を定義したものである．WS-Routing4Axis は Java を用いた Axis 用の WS-Routing の実装である．

WS-Routing にはメッセージを識別し，あるメッセージを他のメッセージに対応付ける id 要素と releaseTo 要素が定義されている．この 2 つの要素を SOAP ヘッダに記述することでメッセージを識別する．図 12 に WS-Routing 要素を付加した SOAP メッセージを示す．

WS-Routing を用いて識別する ID には，可能な限りの一意性を持たせるために UUID(Universally Unique IDentifier)を用いる．

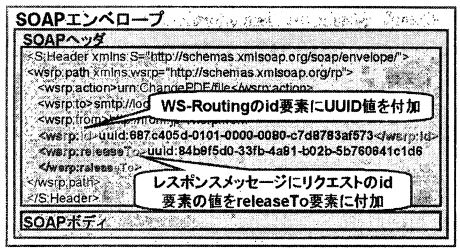


図 12: WS-Routing 要素を付加したメッセージ

5. PDF 変換サービスの試作と評価

SOAP over SMTP を用いた非同期型の Web サービスの効率を評価した[6]. プロバイダで長時間の処理を行う PDF(Portable Document Format)変換サービスを例として, SOAP over HTTP を用いた同期型と, 本研究で提案する非同期型の 2 つの方法で試作, 実行し, Web サービスの動作の違いと性能を比較した. 実行環境は, リクエスタとプロバイダ共に Pentium 4(2.66GHz), 1GB RAM を使用し, ネットワーク速度を 100Mbps とした. OS は WindowsXP で, Apache Axis 1.2 Beta を用いた.

5.1. PDF 変換サービスの試作

PDF 変換サービスは, PS (PostScript)ファイルを PDF ファイルに変換するサービスである. 図 13 に示すように, クライアントは PS ファイルをリクエストメッセージに添付してプロバイダに送信する. プロバイダは, 受信した PS ファイルを Adobe Acrobat Distiller に渡し, PDF ファイルに変換する. 変換された PDF ファイルは SOAP over HTTP ではレスポンスメッセージに添付し, SOAP over SMTP ではリクエストで指定したアドレスへメールに添付して返信する.

PDF 変換サービスを試作するために実装した Java プログラムの規模を表 3 に示す.

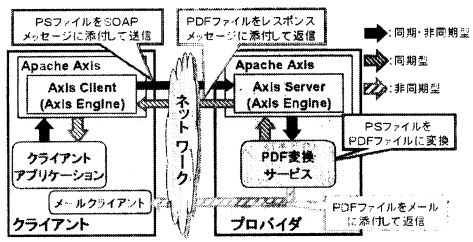


図 13: PDF 変換サービス

表 3: PDF 変換サービスの開発規模

	プログラム行数		クラスファイル数(新規)	
	同期型	非同期型	同期型	非同期型
クライアント	420	440	5	5
プロバイダ	100	155	2	2
合計	520	595	7	7

5.2. 同期型・非同期型 Web サービスの動作比較

図 14 に同期型 PDF 変換サービスの処理の流れをシーケンス図により示す. 同期型 PDF 変換サービスでは, クライアントアプリケーションから PS ファイルを送信すると HTTP 用の Transport Sender である HTTPSender によって SOAP メッセージが送信される. しかし同期型では, SOAP メッセージの送信が完了しても, プロバイダで PDF 変換処理が終了し, PDF ファイルをレスポンスとして受信するまでは, クライアントアプリケーションに処理が戻らない状態となる. クライアントがサービス利用に必要とする時間 T_s は式(1)で定義できる.

$$T_s = T_1 + T_p + T_2 \quad (1)$$

T_p : PS ファイルから PDF ファイルへの変換処理時間
T_1 : SOAP over HTTP リクエスト送信時の遅延時間
T_2 : レスポンス受信時の遅延時間

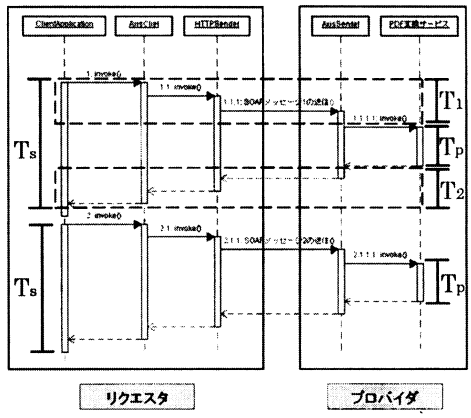


図 14: 同期型 PDF 変換サービスのシーケンス図

次に, 非同期型 PDF 変換サービスの処理の流れを示したシーケンス図を図 15 に示す.

非同期型 PDF 変換サービスではクライアントアプリケーションから PS ファイルを送信すると SMTPSender によって SOAP メッセージが送信される. ここで, 非同期型は送信が完了すると直ちにクライアントアプリケーションに処理が戻される. プロバイダ側で前のリクエストの PDF 変換処理が終了していなくても, 連続して PS ファイルを送信可能となった.

非同期型でクライアントがサービス利用に必要とする時間 T_a は式(2)で定義できる.

$$T_a = T_3 + T_4 \quad (2)$$

T_3 : SOAP over SMTP でリクエスト送信時の遅延時間
T_4 : クライアントアプリケーションへ処理を返す時間

以上のことから, 同期型と非同期型 PDF 変換サービスの処理時間の差は式(3)の Δt の値で求められる.

$$\Delta t = T_s - T_a = (T_1 + T_p + T_2) - (T_3 + T_4) \quad (3)$$

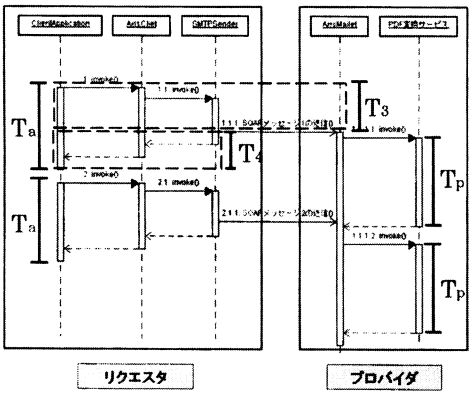
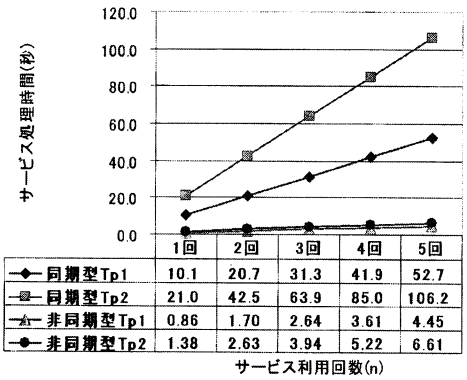


図 15: 非同期型 PDF 変換サービスのシーケンス図

図 16 に，異なるサイズの 2 つの PS ファイルを同期型と非同期型の PDF 変換サービスで PDF ファイルに変換したときの，クライアントのサービス処理時間(T_s,T_a)の計測結果を示す．2 つの PS ファイルの PDF 変換処理時間(T_p)は，T_{p1} が約 7 秒，T_{p2} は約 15 秒である．



	1回	2回	3回	4回	5回
同期型Tp1	10.1	20.7	31.3	41.9	52.7
同期型Tp2	21.0	42.5	63.9	85.0	106.2
非同期型Tp1	0.86	1.70	2.64	3.61	4.45
非同期型Tp2	1.38	2.63	3.94	5.22	6.61

図 16: PDF 変換サービス処理時間

実験結果より，同期型 PDF 変換サービスのサービス処理時間は，T_{p1},T_{p2} にかかわらず，全体の約 70%を T_p に費やしているため，式(1)に示したように T_p に依存し，その値によって大きく変化することがわかる．

一方，非同期型は式(2)より，T_p の影響を受けないサービス利用が可能であるため，2 つのファイルのサービス処理時間を比較しても，その差は小さく，利用回数が増加しても利用時間の増加量は少ない．

以上の結果より，サービスを利用する回数(n)と，T_p の増加により，同期型と非同期型の処理時間差 Δt が増大する．

PDF 変換サービスの評価結果から，非同期型 Web サービスを用いることは，リクエスタのサービス処理時間を短縮し，効率の良いサービスの利用を実現できる有効な方法であるといえる．

6. 考 察

Web サービス形態の多様化により，同期型のメッセージングのみでは Web サービスの利便性を十分に活かせない．そこで，本研究では，同期型 Web サービスでは困難なサービス形態にも対応する一方法として，SOAP over SMTP による非同期型 Web サービスを提案した．SOAP over SMTP で Web サービスのリクエストメッセージを送信することで，一方向型のメッセージングが実現され，同期型でメッセージングを行う際に発生するレスポンスメッセージの待ち時間を大幅に短縮できる．

例えば，試作した PDF 変換サービスでは，プロバイダ側で PDF 変換処理を行うため，同期型 Web サービスではレスポンスメッセージを受信するまでに PDF 変換処理時間に依存した待ち時間が必要となる．しかし，非同期型 PDF 変換サービスを用いることで，PDF 変換処理時間に影響されず，リクエストの送信処理の時間間隔でサービスが利用可能となる．したがって，PDF 変換処理に時間のかかるファイルを連続で送信しても短時間でサービスを利用できる．

さらに，非同期型メッセージに ID を付加し，メッセージ間の関連を明らかにすることで，複数のメッセージに対応関係を持たせることができる．これにより，複数のリクエストを集計し結果を返すような，多重処理を行う Web サービスを実現することも可能である．

例えば，時間差で発生するリクエストの集計処理を行うサービス形態を考えたとき，同期型ではすべてのリクエストが集計されるまで待機し続けるため，適切ではない．しかし，非同期型では，集計するリクエストに関連する ID を付加して，一方的に送信するだけでサービスを利用できる．

待ち状態を持たず，一方向型メッセージを行う非同期の特性を活かすことで，新たな Web サービスの利用形態にも対応が可能になると考える．

7. 関連研究

非同期型 Web サービスを実現する種々の方法が提案されている．

本論文では，SOAP over SMTP による一方向型の非同期メッセージングを提案した．しかし，他にも双方向のメッセージ交換を行う要求/応答型や送信請求/応答型，一方向のメッセージ送信を行う通知型などがある．これらの通信プリミティブの特性を基にしてパターン化を行った，非同期パターンが提案されている[1]．

さらに，この非同期パターンの一つである，ポーリングによる要求/応答型のメッセージングの一方法として，同期型の SOAP over HTTP を，処理要求と結果取得に分離し，HTTP を 2 回接続することで非同期メッセージングを行う提案もある[5]．この方法では，同期型と非同

期型のトランスポートプロトコルに同じ HTTP を用いることで，プロバイダは Web サーバ上で，同期型と非同期型のサービスを同時に提供することを可能にする．しかし，1 回のサービス利用に 2 回の HTTP 接続が必要であるため，サービスの性能処理や複雑さの点で問題になると考えられる．

また，現段階で非同期サービスをサポートしていない SOAP を非同期に対応させるための研究も行われている[10]．SWAP (Simple Workflow Access Protocol)や AWSP (Asynchronous Web Services Protocol)などの仕様を基に SOAP を拡張した，ASAP (Asynchronous Service Access Protocol)が策定されている．ASAP は一般的な非同期サービスの実現に必要な最小限の機能だけに限定し，実装しやすい仕様を目指している．しかし，この方法では，SOAP を独自に拡張するため，SOAP 標準との互換性を保つことが問題である．

8. 今後の課題

今後の課題としては，次の項目が挙げられる．

(1) 応答メッセージのポーリング

本研究で提案した方法により，SOAP over SMTP による一方向型の非同期型 Web サービスを実現した．しかし，7 章でも述べたように，非同期型 Web サービスでも，レスポンスメッセージを期待する双方向型の転送プリミティブが考えられる．今後は，SOAP over SMTP でも双方向型のメッセージ形態に対応するため，リクエスタがプロバイダに対して応答メッセージの準備ができたかどうかを問いかけるポーリングを行い，メッセージを受信する機能を新たに付加する必要がある．双方向型の非同期メッセージングが実現されることで，非同期型の特徴を活かした，非同期型 Web サービスの有効的な利用形態が実現できると考えられる．

(2) リクエストの認証

実際に SOAP over SMTP による非同期型 Web サービスを提供するためには，不正なサービスの利用を防止する方法が必要となる．

例えば，メッセージ送信時にサービスを利用できるユーザかどうかをアカウントやパスワードで認証する方法が考えられる．このとき，ユーザ認証をトランスポートプロトコルのレベルで行うか，SOAP メッセージ内に認証用のアカウント要素やパスワード要素を定義して，トランスポートに依存しない認証を行うかという問題もある．

9. まとめ

本研究では，同期型 Web サービスが持つ問題点を解決するために，非同期型の一方向メッセージングを行う Web サービスを実現する方法を提案し，Apache Axis を用いた SOAP over SMTP の実装を行った．さらに，非同期型 Web サービスとして PDF 変換サービスを試作し，クライアントがサービスを利用するための処理時間を計測，比較することで非同期型 Web サービスの有効性を確認した．非同期型 Web サービスを用いることで同期型 Web サービスよりも効率よくメッセージを送信可能になり，サービスの利便性の増大が期待できる．

また，非同期型 Web サービスを実現する方法として，使用するトランスポートプロトコル，SOAP メッセージの識別方法の問題があった．これに対して，我々は SMTP を用いた非同期メッセージング，ID の付加による複数のメッセージの関連性を付与する方法を提案し，問題の解決をした．

このアプローチは，標準のプロトコルに準拠し，既存のシステムを再利用することで，容易に利用することが可能となった．したがって Web サービス本来の特性に沿ったものであるといえる．

今後は，双方向型メッセージ形態への対応方法を検討し，実現したいと考えている．

参考文献

[1] H. Adams, Asynchronous operations and Web Services, 2002, http://www-106.ibm.com/developerworks/library/ws-asynch1.html.

[2] 青山 幹雄，ソフトウェアサービス技術へのいざない，情報処理，Vol.42，No.9，Sep.2001，pp.857-862.

[3] The Apache Software Foundation: Axis Documentation, http://ws.apache.org/axis/java/index.html.

[4] The Apache Software Foundation, James Documentation, http://james.apache.org/documentation_2_1.html.

[5] 木村 利幸，非同期 Web サービス実現アーキテクチャ提案，2004，http://ws.apache.org/~toshi/docs/JSR224-F2F-no1-jp.pdf.

[6] 森 晃，服部 隆尚，非同期型 Web サービスに関する研究，南山大学 数理情報学部 情報通信学科 卒業論文，Jan. 2005.

[7] 本 俊也，最新 Web サービスマスタリングハンドブック，秀和システム，2004.

[8] H. F. Nielsen, and S. Thatte, Web Services Routing Protocol (WS-Routing), 2001, http://www.microsoft.com/japan/msdn/webservices/dnsrvspec/ws-routing.asp.

[9] 日本ユニテック，SOAP/UDDI/WSDL Web サービス技術基礎と実践 徹底解説，技術評論社，2002.

[10] J. Ricker, et al., 2005, http://www.oasis-open.org/committees/documents.php?wg_abbrev=asap.