5. メッセージ管理技術の実装

5.1. はじめに

本研究では、SOAP（Simple Object Access Protocol）とよばれる分散環境用プロトコルを用いてWebサービスを実際に運用するにあたり必要不可欠となる機能群、すなわち、サービスプログラムへの仲介・配信、リクエストおよびレスポンスメッセージの管理、非同期メッセージング処理、サーバ間転送（ルーティング）処理、トランザクション管理などを実現する「OpenSOAP サーバ」のアーキテクチャの提案と実装を行う。

次世代ビジネスプラットフォームといわれるWebサービスの枠組みにおいて必須であることは明白でありながらも、SOAP 仕様のみではカバーされないこれら機能を実装したミドルウェアがいまだ少ない現状において、本プロジェクトでは成果物をオープンソースとして広く公開している。

RPC を対象とする SOAP 実装においては、Peer to Peer の接続状況を想定して、SOAP メッセージ作成と HTTP バインディングによるメッセージ受け渡しの仕組みを構築することになるが、本サブテーマではこれに加え、

- 一つのマシン上で異種・複数のサービスが稼動する場合に、メッセージに記述されるメソッド名を参照して適切なサービスプログラムに渡すディスパッチングや、サービスプログラム群のプロセス管理。
- 複雑な処理や、人手を介するような時間的遅延を要する場合に、一度セッションを閉じてから、後に再度処理結果を要求するような非同期メッセージングに伴う、スプール管理や、メッセージ照合の仕組み。
- ファイアウォールを越えてイントラネット内のサービスへの接続などのルーティング処理や、サービス実体はなくとも場所の情報だけをもち適切なマシンへサービスを依頼する機能や、複数マシンの連携によるサービスなどを実現するマシン間転送処理。
- 不正なリクエストに対して、サービスを稼動させる前にエラーメッセージを返信する機能。また、このメッセージが正式のものであるというセキュリティ上の署名・認証機能。
- Web 上に散在する複数のサービスを統括的、体系的に処理する、Web スケールのトランザクション機能。

など､実用的Webサービスに有効となる機能群を実装した｡この実現のために、サーバ内部のアーキテクチャや、セッション管理、リクエストメッセージに依存するメッセージ処理ロジックなど、実装上の課題点も数多く挙げられ、これを解

決すべく設計を行った。

ここで構築する OpenSOAP サーバは、従来型のモノシリックなシステムではなく、複数のプログラム群から構成されるアーキテクチャを採用しており、耐故障性の向上も図られている。現状において OpenSOAP サーバは Linux, Solaris 8, NetBSD, Windows 2000 の各プラットフォームでの動作が可能であり、サーバの開始・終了・再起動、また、動作チェックを簡便に行うためのツール群、ロギング機能も同梱して提供される。本節では、サーバの持つ主要機能の概要と、サーバを構成する内部プログラム群、また実際のサーバの運用に当たっての詳細事項についての説明を行う。

5.2.　OpenSOAP サーバの主要機能

5.2.1.　サービスプログラム仲介・連携機能

OpenSOAP サーバの主な役割は、クライアントからのリクエストメッセージを適切なサービスプログラムに受け渡し、サービスで処理された結果（レスポンスメッセージ）をクライアントに戻すまでの一連の処理を統括管理することである（図 5.2.1）。これによりクライアントは、様々なサービスを稼動しているサービスプロバイダに対して、それぞれのサービスの受け口（Endpoint）を意識することなく、サービスプロバイダの一つの窓口（OpenSOAP サーバ）に対してリクエストメッセージを送信すれば、サーバがリクエストを適切なサービスプログラムに配信する。逆にサービスプログラムは、インターフェースをそれぞれ公開せずとも、サーバとのやり取りのみでサービスを利用者に提供できる。また、負荷集中や、プログラムのダウンなど、サービスプログラムの稼動状況によっては、接続のタイムアウトを SOAP メッセージによってクライアントに通知する。

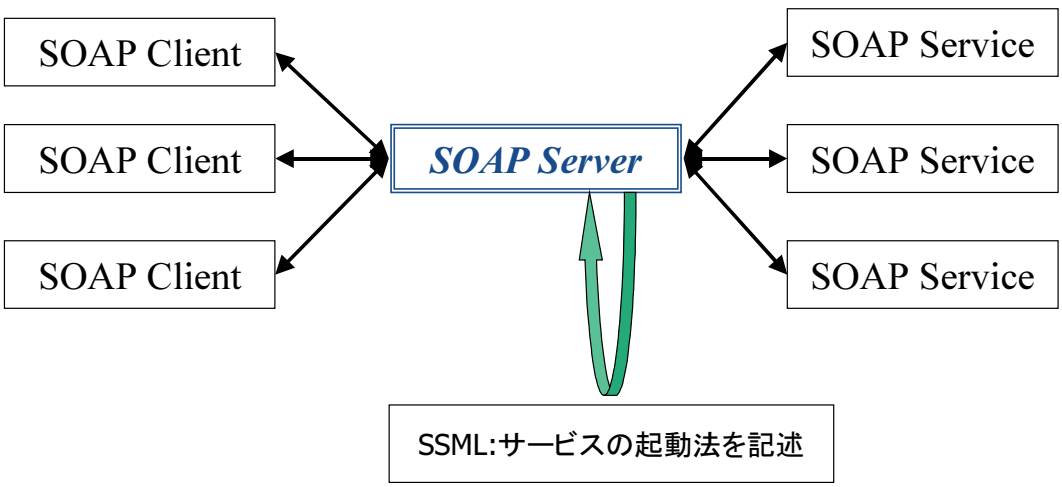

図 5.2.1 OpenSOAP サーバ概念図

現状の実装では、クライアントからのメッセージの窓口として、CGI（Common Gateway Interface）を用いている。つまり、クライアントは Web サーバ上に設置された SOAP インターフェース用 CGI プログラムに SOAP メッセージを送信し、これを介して OpenSOAP サーバはリクエストメッセージを受け取ることができる。

次に、OpenSOAP サーバはリクエストメッセージ内のボディブロックに記述されているサービスメソッド名を参照し、適切なサービスプログラムに配信する。この際、OpenSOAP サーバとサービスプログラムの連携においては、サービス提供者が、独自仕様である SSML (Soap Service Mapping Language)ファイルに、プロセス間接続方式やプログラムの起動方法などの設定をあらかじめ XML 形式によって記述し、OpenSOAP サーバはこれに従ってサービスプログラムとの SOAP メッセージの送受信を行う。SSML ファイルの詳細については後述する。

リリース版ではサービスプログラムを UNIX の inetd に登録し、サーバとは socket 通信によって情報交換を行う。今後はこの他、標準入出力方式、名前付きパイプ方式、プロセス間通信(IPC)方式、Windows COM 方式などをサポートする予定である。これら通信方式のほか、通信に必要なパラメータ、プログラム起動オプション、タイムアウト時間、最大同時起動プロセス数、サービス呼び出しメソッド名などが SSML ファイルに記述される。

サービスプログラムで処理結果を元に作成されたレスポンスメッセージは、インターフェース用プロセスを通じて、HTTP によって再度クライアントに返される。この間、基本的に HTTP セッションは開いた状態で、クライアントは CGI への POST メソッドによって処理結果を入手する形になる。

5.2.2. 非同期処理とメッセージ管理機能

OpenSOAP サーバの大きな特長として、非同期処理を可能としている点がある。ここでいう同期処理とは、クライアントがリクエストを送ってからサービスを受け取るまで接続（セッション）を維持した状態で行う処理である。すなわち非同期処理とは、サーバにクライアントからのリクエストが渡った時点で通信をいったん終了し、再度レスポンスの取得を行うためにサーバとの接続を行う通信方法をとる処理である。これにより、複雑で長時間を要するサービス処理や、人手を介するサービス等にも現実的に対応が可能となる。類似する技術にはたとえば、POP 方式の電子メールシステムがあげられる。電子メールでは、サーバに蓄えられたメールを、クライアントからの要求（ポーリング）に応じて返信する。

このためには、クライアントとサーバの間で、後にクライアントが結果要求する際に、リクエスト内容に関する照合のための約束事が必要になることと、クライアントがサーバ内でいかに結果を保持するか、または、いかなる条件で保持内

容を破棄するかなどのいくつかのクリアすべき技術課題がある。

このことは、同期処理や、後に述べるサーバ間転送などにおいても、複数プロセスの連携によって稼動する本 OpenSOAP サーバのようなアーキテクチャにおいては必須の事項である。すなわち、SOAP メッセージをプロセス間、サーバ・サービスプログラム間でどのように受け渡すか、他の OpenSOAP サーバへの転送時にいかに受け渡したメッセージを保証するかという問題である。

これについて本研究では、サーバ内部でリクエストメッセージに対して、処理開始時間を文字列とした識別子を付加することで、プロセス内部では ID のみの受け渡しとし、メッセージ本体は共有ファイルの形式とすることで、サーバの負荷を軽減している。非同期処理についてもこれにサーバ名とサービスメソッド名を付加した同様の ID を SOAP メッセージのヘッダブロックに付加することで、クライアント・サーバ間でどのリクエストに対する回答かの照合を行う（図 5.2.2）。以下に非同期処理の詳細を述べる。

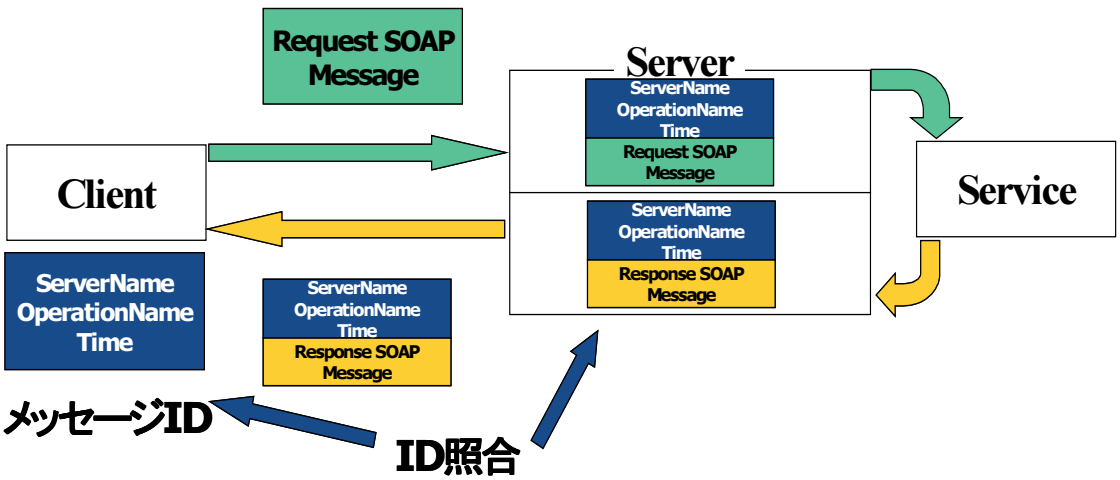

図 5.2.2 OpenSOAP サーバのメッセージ管理技術

クライアントからの非同期処理の要求には、SOAP ヘッダに指定された仕様の記述を行う。OpenSOAP ではこの他にも、サーバ転送処理（ルーティング）や、タイムアウト時間を設定するための OpenSOAP 向けヘッダ拡張を行っている。図 5.2.3 は非同期処理要求を行う SOAP メッセージのヘッダ部をあらわしている。具体的には、OpenSOAP 拡張ヘッダブロック<opensoap-header:opensoap-header-block>ブロック内の<opensoap-header:async>要素に文字列 true が含まれると、サーバは非同期要求とみなす。<opensoap-header:opensoap-header-block>にはこの他、サーバ内でのタイムアウト時間を指定する<opensoap-header:ttl>要素などが含まれる。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <opensoap-header:opensoap-header-block
                    xmlns:opensoap-header="http://header.opensoap.jp/1.0/">
    <opensoap-header:ttl opensoap-header:type="second">80</opensoap-header:ttl>
    <opensoap-header:async>true</opensoap-header:async>
    </opensoap-header:opensoap-header-block>
  </SOAP-ENV:Header>
```

図 5.2.3 非同期処理要求のための SOAP ヘッダ拡張

OpenSOAP サーバはこの要求に対し、上述のユニークな識別子（Message ID）を SOAP ヘッダに付加したレスポンスメッセージ（図 5.2.4）を同期処理でクライアントに返信する。OpenSOAP サーバ内においても、この識別子によってサービスの作業進行状況を管理し、サービスプログラムから返信された処理結果をスプールしておく。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
  <opensoap-header:opensoap-header-block
                    xmlns:opensoap-header="http://header.opensoap.jp/1.0/">
    <opensoap-header:message_id>
      TransactionControl.opensoap.jp.2002022021010200001
    </opensoap-header:message_id>
  </opensoap-header:opensoap-header-block>
  </SOAP-ENV:Header>
  …
```

図 5.2.4 非同期処理のためのレスポンスメッセージ（Message ID）

クライアントはこの Message ID を含む SOAP メッセージを保存しておき、リクエストの結果を受け取る際には、サーバに再接続しこの SOAP ヘッダをそのまま付加して Body を空にした SOAP メッセージを OpenSOAP サーバに送信することで、OpenSOAP サーバは識別子の照合をもって結果を返信する。この際、レスポンスメッセージをスプールしておくテーブルを参照し、目的の Message ID を持つレスポンスがない場合は、その旨を記述した SOAP Fault メッセージを返信する。

5.2.3.　サーバ間転送機能

OpenSOAP サーバは、ローカルに処理可能なサービスプログラムを管理していない場合、もしくは恣意的に転送経路が指定された場合に、他のサーバにメッセージを転送する機能を備えている。つまり、他の OpenSOAP サーバ群と連携をとることによって、クライアントが常用する Endpoint を一つに固定し、そのサーバが保有していないサービスに対するリクエストを行ったとしても、あたかも DNS サーバのように他のサーバにそのサービスの処理を任せるという枠組みが構築可能である。

同時に、ファイアウォールによるグローバルネットワークとの隔離によってセキュリティを確保しようとする考え方が一般的となった現状において、このネットワーク境界を越えた分散処理の実現が SOAP の目的の一つとしてあげられているが、これは、本サーバのようなルーティング処理の実装によって初めて可能となる技術である（図 5.2.5）。ファイアウォール上に OpenSOAP サーバを展開し、転送経路をあらかじめ設定することでグローバルアドレスとローカルアドレスを変換、ローカルネットワーク内の OpenSOAP サーバに転送することによって、外部からイントラネット内のサービスを利用可能となる。このような仕組みを提供した場合、セキュリティが非常に重要な課題となるが、OpenSOAP では、暗号化や認証機構などのセキュリティ対策を実装しているほか、後述するように、これ

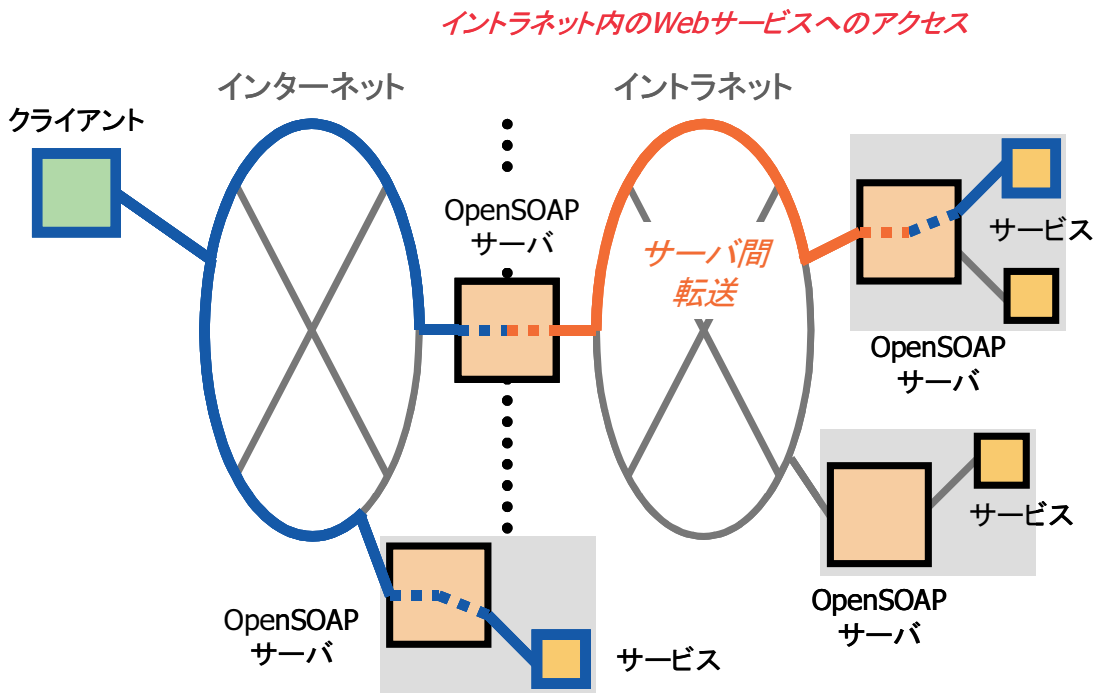

図 5.2.5 OpenSOAP サーバ間転送による Firewall 越しルーティング

を利用して OpenSOAP サーバそのものも署名を付加してメッセージを生成したり、署名付きメッセージの認証を行うことができる。

転送経路は、それぞれの OpenSOAP サーバが常用の転送先を設定しておく他、クライアントが SOAP ヘッダに明示的に設定することもできる。

このようなサーバ間転送機能は、前述の非同期処理と連携することで、より広汎で多様なサービスに対応可能となる。図 5.2.6 は、転送処理の詳細図であり、クライアントとの送受信を非同期的に行った場合を説明している。OpenSOAP サ

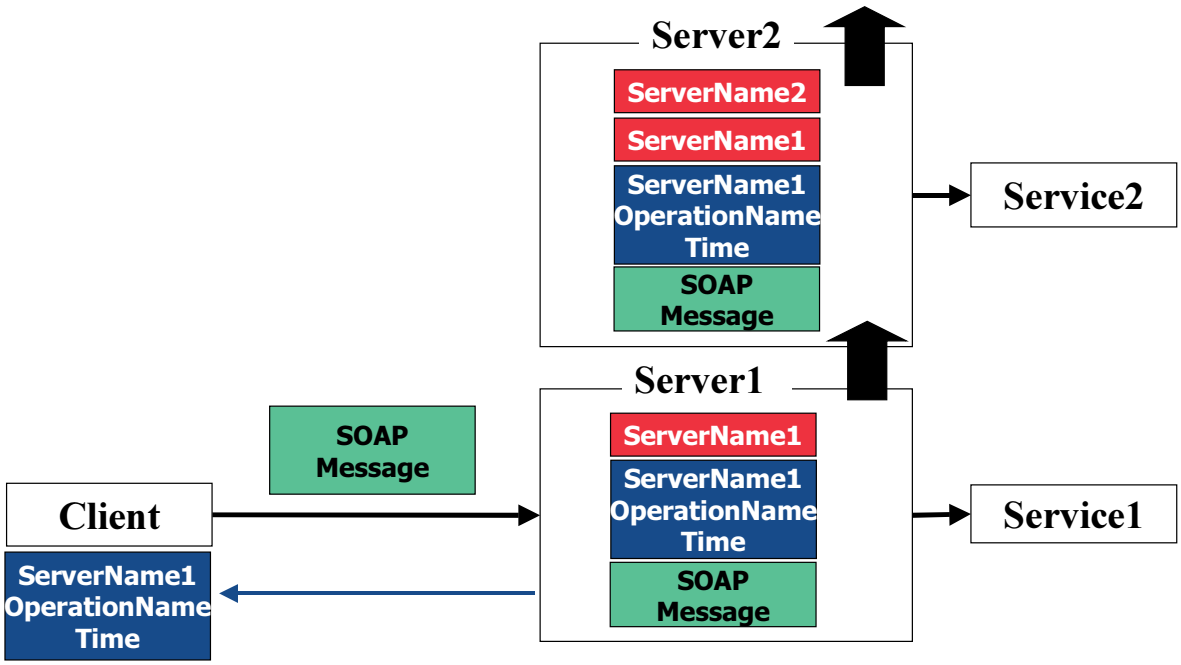

図 5.2.6 非同期転送処理の詳細図

ーバはリクエストメッセージを受け取ると、メソッド名と SSML ファイルの記述を照合し、自己サーバ内でサービスプログラムが管理下にあれば処理を行うが、管理下にない場合、リクエストメッセージの転送設定を参照し、転送ルートが記述されている場合はそれに従って他サーバへの転送を行う。転送設定がリクエストメッセージになされていない場合は、自己サーバの転送設定にしたがって常用の転送先へメッセージを送る。クライアントには、非同期処理の要求がなされていた場合には Message ID を記述したレスポンスメッセージを返信し、非同期処理待ち行列に挿入する。また、転送処理に関する設定が適切になされていない場合は、処理不能の旨のエラーメッセージ（SOAP Fault）を返信する。

転送は多段で行うことも可能であり、転送元サーバは、転送する SOAP メッセージのヘッダに自己サーバの URL とタイムスタンプを追加記述する。転送先サー

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelop">
<SOAP-Env:Header>
  <opensoap-ext:opensoap-ext-block
                xmlns:opensoap-ext="http://opensoap.jp/ext/" mustUnderstand="1">
    <opensoap-ext:message_id>
      GetProductSpec.takumi-dell.200108130923450001
    </opensoap-ext:message_id>
    <opensoap-ext:received_path>
      <url>http://foo1.bar.com/trans_if.cgi</url>
      <time>2001-08-10T14:20:18+09:00</time>
    </opensoap-ext:received_path>
  </opensoap-ext:opensoap-ext-block>
</SOAP-Env:Header>
…
```

図 5.2.7 サーバ間転送処理 SOAP ヘッダ拡張

バは、非同期処理待ち行列には転送元サーバの URL を追加し、自己サーバ内で処理するかさらに転送処理を行うかの判断を行う。こうすることで、セッションの閉じられた非同期処理手順の場合にも、転送してきた道筋を逆にたどって処理結果メッセージをクライアントまで戻すことができる。処理が完了していないリクエストについても、転送ルート上の各 OpenSOAP サーバが非同期処理待ち行列を参照し、レスポンスメッセージの有無を確認して返答することができる。図 5.2.7 は、OpenSOAP サーバがリクエストメッセージの SOAP ヘッダに転送元サーバ情報（URL, タイムスタンプ）を追加記述した例である。OpenSOAP 拡張ヘッダブロック<opensoap-ext:opensoap-ext-block>内に<opensoap-ext:received_path>要素として、URL と転送処理を行った時間が記述されている。

5.2.4.　エラー処理とサーバ署名

OpenSOAP サーバは、各サービスの運用をサポートするための機能が多く含まれており、その中の一つにエラー処理がある。すなわち、クライアントの目的とするサービスが稼動していない場合、ビジーである場合、処理が送れている場合など、サービスそのものがエラーメッセージを返信できないような状況においても、そのサービスを管理する OpenSOAP サーバが状況を判断し、エラーの内容を記述したエラーメッセージ（SOAP Fault）をクライアントに返信する。

また、OpenSOAP サーバそのものに起因するエラーについても同様であり、SOAP のエラー仕様に従ってメッセージを生成する。

以下に、サーバが生成するエラーメッセージの例を挙げておく。図 5.2.8 は、非同期処理の結果要求に対して、レスポンスメッセージのスプールに該当するメッセージがない場合にクライアントに返されるメッセージである。つまり、Message ID が不正であったり、既にクライアントによって回収が済んでいるリクエストであったり、タイムアウトなどでスプールから消去された場合などが該当する。図 5.2.9 は、クライアントから要求されたサービスメソッド名が当サーバの正当な管理下になく、転送設定もなされていない場合に返信されるメッセージである。

さらに、これらエラーメッセージや、サーバが発信する種々のメッセージについて、確実に当該 OpenSOAP サーバが作成したメッセージであることを証明する署名機構を、当プロジェクト・サブテーマ「総合セキュリティ機能の実装」にお

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  encodingStyle="http://schemas.xml.soap.org/soap/encoding/">
<SOAP-ENV:Body>
   <SOAP-ENV:Fault>
      <SOAP-ENV:faultcode>SOAP-ENV:Server.NoEntry</SOAP-ENV:faultcode>
      <SOAP-ENV:faultstring>No Corresponding Entry in Response Spool</SOAP-ENV:faultstring>
      <SOAP-ENV:detail>This FAULT is created by linux.opensoap.jp.</SOAP-ENV:detail>
   </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.2.8 サーバの生成するエラーメッセージ(1)

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  encodingStyle="http://schemas.xml.soap.org/soap/encoding/">
<SOAP-ENV:Body>
   <SOAP-ENV:Fault>
      <SOAP-ENV:faultcode>SOAP-ENV:Server.OperationNotFound</SOAP-ENV:faultcode>
      <SOAP-ENV:faultstring>Request Operation is not Treated in Server</SOAP-ENV:faultstring>
      <SOAP-ENV:detail>This FAULT is created by opensoap.jp.</SOAP-ENV:detail>
   </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.2.9 サーバの生成するエラーメッセージ(2)

ける成果を OpenSOAP サーバに搭載した。OpenSOAP セキュリティの機能を利用することにより、サーバの署名だけでなく、その署名の身元を保証する認証局の機能をも利用可能であり、強固なセキュリティを要する様々な場面に適用可能となり、このような技術は積極的に応用すべきである。なぜならば、電子ビジネスや Web サービス､RPC など､様々な重要メッセージを仲介するサーバにおいては、改ざん、盗聴、なりすましなど、悪意ある操作も可能であり、上記セキュリティ技術が必須だからである。

5.3.　サーバオプション機能　—Web トランザクション管理—

5.3.1.　トランザクション管理機能概要

OpenSOAP サーバの特徴のひとつとして、サーバの追加機能をサービスプログラムとして提供可能な点があげられる。すなわち、サーバと同梱するサービスプログラムにサーバオプションの機能をもたせ、クライアントは SOAP メッセージ上にメソッドを追加することでこの機能を利用可能となる。

OpenSOAP トランザクション管理機能は、そのようなサービスプログラムの一つとして提供されている。これは、OpenSOAP が提示する仕様に従って、複数の処理要求をそれぞれ SOAP ボディブロックとしてひとつのリクエストメッセージ（SOAP Envelope）に同封し、処理手順と必要項目を指定することで、トランザクションサービスがこれらリクエスト群を分解、Web 上に散在する指定された各サービスの URL（Endpoint）に処理要求を再送信し、収集した各処理結果を再びひとつのレスポンスメッセージとしてクライアントに返信する機能である。この際、全ての処理結果が成功として返信されなければ、トランザクションサービスはすでに中間的処理を要求した各サービスプログラムに対して処理の取り消し要求を行うことで、一連の処理の整合性と安全性を保証する。すなわち、OpenSOAP トランザクションに対応するサービスは、取り消し（ROLLBACK）要求に対応できるよう、処理履歴を保持する仕組みを実装している必要がある。

複雑で長時間を要するトランザクション要求も、OpenSOAP サーバの非同期処理機能によって現実的に実行可能である。また、トランザクションサービスは各サービスプログラムへのリクエストをボディブロック単位で格納する仕様であるため、ボディブロックの内容を保証する署名機能など、OpenSOAP の提供するセキュリティサービスをそのまま享受できる。

以下に、本トランザクションサービスの仕様の詳細を、銀行間送金の例とともに述べる。

5.3.2.　OpenSOAP トランザクションクライアント（リクエストメッセージ）

仕様

クライアントは以下の仕様に従ったリクエストメッセージを送信できる必要がある。

- トランザクション制御記述ブロック

図 5.3.1 は A 銀行に出金の要求をし、それが受け入れられたら B 銀行に同額の入金要求をするトランザクションの例である。例に示されるように、リクエストメッセージの Body（<SOAP-ENV:Body>）に含まれる最初のボディブロックは

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>

  <t:TransactionControl xmlns:t="http://services.opensoap.jp/transaction/">
    <endpoint transactionRequestID="1">http://opensoap.jp/cgi-bin/soapInterface.cgi</endpoint>
    <endpoint transactionRequestID="2">http://opensoap.jp/cgi-bin/b_bank_transfer.cgi</endpoint>
  </t:TransactionControl>

  <t:TransactionBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="1">
    <p:PaymentRequest xmlns:p="http://services.opensoap.jp/a_bank_transfer/">
      <account>1338675</account>
      <amount>$5000</amount>
      <to>b_bank:1252412</to>
    </p:PaymentRequest>
  </t:TransactionBodyBlock>

  <t:TransactionHeaderBlock transactionRequestID="2"
   xmlns:tp="http://services.opensoap.jp/transaction/">
    <e:encrypt xmlns:e="http://security.opensoap.jp/1.0/" api="OpenSOAP">true</e:encrypt>
  </t:TransactionHeaderBlock>

  <t:TransactionBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="2">
    <d:DepositRequest xmlns:d="http://services.opensoap.jp/b_bank_transfer/">
      <account>1252412</account>
      <amount>$5000</amount>
      <from>a_bank:1338675<from>
    </d:DepositRequest>
  </t:TransactionBodyBlock>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.1 OpenSOAP Transaction Request Envelope

<TransactionControl>である必要がある。<TransactionControl>ブロックの namespace は OpenSOAP の仕様に従い、"http://services.opensoap.jp/transaction/"であることが望ましい。<TransactionControl>ブロックには、トランザクション処理の対象となるサービスの数だけ、子要素<endpoint>が、要求される処理の順番に並んでいる必要がある。

<endpoint>要素には各サービスがトランザクション処理として実行可能な SOAP メッセージの送信先 URL を記述し、各要求処理ごとに固有の任意の属性値"transactionRequestID"を付加する。

その他、トランザクション処理に関わる処理設定はこのブロックに記述され、様々な処理条件やロジックに対処するよう拡張可能な仕様となっている。(当初のリリース版での実装は<endpoint>要素のみ)

- トランザクションヘッダブロック

各サービスに送信されるべきメッセージのヘッダ部が存在する場合、トランザクションヘッダブロック<TransactionHeaderBlock>の要素として記述する（図 5.3.1)。この部分に記述された要素は、各サービスに送信される SOAP メッセージのヘッダブロック（<SOAP-ENV:Header>の直下のブロック）として挿入される。

各<TransactionHeaderBlock>ブロックには、<TransactionControl>ブロックの<endpoint>要素の対応する属性値"transactionRequestID"を付加することで、処理内容と送信先を関係付けている。<TransactionHeaderBlock>ブロックは第２ボディブロック以降であれば、どの順番で挿入されていても適切に処理される。

- トランザクションボディブロック

各サービスに送信されるべきメッセージのボディ部は、トランザクションボディブロック<TransactionBodyBlock>の要素として記述する（図 5.3.1)。この部分に記述された要素は、各サービスに送信される SOAP メッセージのボディブロック（<SOAP-ENV:Body>の直下のブロック）として挿入される。

同様に、各<TransactionBodyBlock>ブロックには、<TransactionControl>ブロックの<endpoint>要素の対応する属性値"transactionRequestID"を付加する必要がある。<TransactionBodyBlock>ブロックも、<TransactionHeaderBlock>ブロック同様、第２ボディブロック以降であれば、どの順番で挿入されていてもよい。

5.3.3.　OpenSOAP トランザクション対応サービス仕様

OpenSOAP Transaction Service に対応したサービスは、クライアントが生成する

SOAP メッセージの仕様（ヘッダブロック、ボディブロックの内容）とサービスの機能をクライアントに公開すると供に、トランザクションサービスとの接続について、以下の仕様に従う必要がある。

- 要求処理に対する返信

トランザクションサービスによって分割生成された各 Envelope（図 5.3.2、図 5.3.3）は、対応するそれぞれの Endpoint に送信されます。各サービスは、Envelope にしたがって要求された処理を行い、その処理を ROLLBACK できる状態で保存した後、レスポンスメッセージ（図 5.3.4）をトランザクションサービスに返信する。レスポンスメッセージの仕様は各サービスごと独自となるが、第 1 ボディブロックの直下には処理の結果を記述する子要素<TransactionResult>を含む必要がある。

<TransactionResult>には、そのサービスでの処理の成功を通知する"SUCCESS"

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
  <p:PaymentRequest xmlns:p="http://services.opensoap.jp/a_bank_transfer/">
    <account>1338675</account>
    <amount>$5000</amount>
    <to>b_bank:1252412</to>
  </p:PaymentRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.2 A 銀行へ送信される SOAP Envelope

を記述する。それ以外の文字列、もしくはサービスから<SOAP-ENV:Fault>等が返

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
</SOAP-ENV:Header>
  <e:encrypt xmlns:e="http://security.opensoap.jp/1.0/" api="OpenSOAP">true</e:encrypt>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <d:DepositRequest xmlns:d="http://services.opensoap.jp/b_bank_transfer/">
    <account>1252412</account>
    <amount>$5000</amount>
    <from>a_bank:1338675</from>
  </d:DepositRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.3 B 銀行へ送信される SOAP Envelope

信された場合は、そのサービスでの処理は失敗、もしくは拒否されたものとみなされる。

また、<TransactionResult>には、サービス内で固有の識別 ID を属性値"transactionID"として付加することができる。"transactionID"属性を用いることで、次のステップでトランザクションサービスからCOMMITまたはROLLBACKの要求がなされた時に、サービス内で整合をとることができる。

- COMMIT または ROLLBACK 要求

トランザクションサービスはサービスから成功（"SUCCESS"）が返信されると、<TransactionControl>ブロックに記述された順番に従って次のサービスへのリクエストメッセージを送信する。

全てのサービスから SUCCESS が返信された場合、トランザクションサービスは再び全ての Endpoint に対して対応する"transactionID"属性値を付加した COMMIT 要求メッセージ（図 5.3.5）を送信して一連の処理の実行を決定する。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<p:PaymentResponse xmlns:p="http://services.opensoap.jp/samples/a_bank_transfer/">
        <t:TransactionResult                    xmlns:t="http://services.opensoap.jp/transaction/"
transactionID="abank13386751012192663">
        SUCCESS
        </t:TransactionResult>
        <p:Comment>1338675 sent $5000 to b_bank:1252412</p:Comment>
</p:PaymentResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.4 A 銀行からのレスポンスの例

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
  <t:TransactionAction xmlns:t="http://services.opensoap.jp/transaction/"
    transactionID="abank13386751012192663">
      COMMIT
   </t:TransactionAction>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.5 トランザクションサービスから A 銀行への COMMIT 要求

全てのサービスから SUCCESS が返信される前に、途中のサービスから失敗メッセージが返信された場合、その前までに SUCCESS が返信されたサービス群に対して"transactionID"属性値を付加した ROLLBACK 要求メッセージを送信する。

- 処理の完結

COMMIT 要求を受け取ったサービスは要求された処理を実行して終了し、ROLLBACK 要求を受け取ったサービスは対応する処理を取り消して状態を差し戻す。さらに、トランザクションサービスにそれぞれの処理が完了した旨の SOAP メッセージ（図 5.3.6）を返信する。

5.3.4. クライアントへの返信

トランザクションサービスは、一連の処理の結果をリクエストメッセージと同様にひとつの Envelope に結合し(図 5.3.7)、クライアントに返信する。SOAP メッセージの最初のボディブロック<TransactionResponse>ブロックには、トランザクションの最終結果が、COMMIT または ROLLBACK として記述される。２つめ以降のボディブロックは、各サービスから返信されたレスポンスメッセージのヘッダブロックが<TransactionHeaderBlock>ブロックに、ボディブロックが<TransactionBodyBlock>ブロックに、さらに COMMIT もしくは ROLLBACK を送信した際のレスポンスメッセージのボディブロックが<TransactionActionResponseBodyBlock>に、それぞれリクエストメッセージに付加した属性値"transactionRequestID"と同じ属性値が付加されて挿入される。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
        <t:TransactionActionResponse xmlns:t=http://services.opensoap.jp/transaction/
          transactionID="abank13386751012192663">
        COMMITED
        </t:TransactionActionResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.6 A 銀行から COMMIT 要求に対する返信

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>

  <t:TransactionResponse xmlns:t="http://services.opensoap.jp/transaction/">
    COMMIT
  </t:TransactionResponse>

  <t:TransactionBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="1">
    <p:PaymentResponse xmlns:p="http://services.opensoap.jp/samples/a_bank_transfer/">
      <t:TransactionResult transactionID="pt13386751012192663">
        SUCCESS
      </t:TransactionResult>
      <p:Comment xmlns:="">1338675 sent $5000 to b_bank:1252412</p:Comment>
    </p:PaymentResponse>
  </t:TransactionBodyBlock>

  <t:TransactionActionResponseBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="1">
    <t:TransactionActionResponse transactionID="pt13386751012192663">
      COMMITED
    </t:TransactionActionResponse>
  </t:TransactionActionResponseBodyBlock>

 <t:TransactionBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="2">
    <p:DepositResponse xmlns:p="http://services.opensoap.jp/samples/a_bank_transfer/">
      <t:TransactionResult transactionID="dt12524121012226863">
        SUCCESS
      </t:TransactionResult>
      <p:Comment>1252412 received $5000 from a_bank:1338675</p:Comment>
    </p:DepositResponse>
  </t:TransactionBodyBlock>

 <t:TransactionActionResponseBodyBlock xmlns:t="http://services.opensoap.jp/transaction/"
   transactionRequestID="2">
    <t:TransactionActionResponse transactionID="dt12524121012226863">
      COMMITED
    </t:TransactionActionResponse>
  </t:TransactionActionResponseBodyBlock>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 5.3.7 クライアントに返信されるトランザクション処理結果

5.4. OpenSOAP サーバのアーキテクチャとメッセージ処理

5.4.1. OpenSOAP サーバを構成するプロセス群

OpenSOAP サーバはサーバ内処理のそれぞれの手順を担当する複数のプログラム群から構成され、プログラム間のメッセージの受け渡しは File ID Manager が担当する永続化 ID によって行われる。

- SOAP Interface:

プログラム名は soapInterface.cgi であり、クライアントからの SOAP メッセージを HTTP で送受信するためのインターフェースプログラムである。このプログラムによりプロトコルごとの SOAP メッセージの交換の方法を吸収する。

以降で説明する MessageDriver からはクライアント-サーバー間のプロトコルに依存しない実装をすることが可能になっている。そのため SMTP などのプロトコルごとのインターフェースプログラムを開発することにより、OpenSOAP サーバ本体を改造すること無く各種のプロトコルに対応することが可能になる。

このプログラムは Apache などの HTTP サーバに CGI として起動され、管理される。

- Message Driver:

プログラム名は msgDrvCreator である。処理の核となる部分はクラス MsgDrv に実装されている。

MsgDrvCreator クラスは MsgDrv クラスのインスタンスを各接続ごとに生成し、マルチスレッドプログラムとして処理を実行できるように作成されている。

このプログラムの役割は、SOAP メッセージの種類が同期リクエストメッセージ・非同期リクエストメッセージ・同期転送リクエストメッセージ・非同期転送リクエストメッセージ・非同期転送レスポンスメッセージのどれに当たるかを判別し、以下のように次のプログラムに送る。

・同期ローカル処理リクエストメッセージ：Sync Service Driver
・非同期ローカル処理リクエストメッセージ：Request Queue Manager
・同期転送処理リクエストメッセージ：
　最終の受取りサーバでない場合：Forward Queue Manager
　最終の受取りサーバである場合：Sync Service Driver
・非同期転送処理リクエストメッセージ：
　最終の受取りサーバでない場合：Forward Queue Manager
　最終の受取りサーバである場合：Request Queue Manager

・非同期転送処理レスポンスメッセージ：
最初の受取りサーバでない場合：Forward Queue Manager
最初の受取りサーバである場合：Response Spool Manager

また、このプログラムはメッセージを識別するためのメッセージ ID を ID Manager から取得しリクエスト SOAP メッセージに付加する。非同期リクエストの場合は SOAP Interface に対してメッセージ ID を送る。さらに非同期転送の場合には、<backward_path>タグにレスポンスを返してもらう SOAP Interface の URL を記述する。

- Sync Service Driver:

プログラム名は srvDrvCreator である。処理の核となる部分はクラス SrvDrv に実装されている。このプログラムは、Message Driver から受け取ったリクエストメッセージを SOAP サービスに送り、レスポンスを受け取るプログラムである。
受け取ったメッセージは Message Driver に返される。

- Async Service Driver:

プログラム名は srvDrvCreator_async である。プログラムの実体は Sync Message Driver と同じく srvDrvCreator であるが、引数 1 をつけて実行することにより、非同期モードで動作する。
処理の核となる部分はクラス SrvDrv に実装されている。非同期ローカル処理の場合、後述の Request Queue Manager から受け取ったリクエストメッセージを SOAP サービスに送り、返されたレスポンスメッセージにリクエストメッセージに付加されていた Message ID をつけて、後述される Response Spool Manager に送る。非同期転送処理の場合は、SOAP サービスから受け取ったレスポンスを後述される Forward Queue Manager に送る。

- Request Queue Manager:

プログラム名は queueManager である。処理の核となる部分はクラス Queue に実装されている。このプログラムは Message Driver から非同期リクエストメッセージを受け取り、複数のメッセージをキューに保存し、順に Async Service Driver に送る。このプログラムが Message Driver からリクエストを受け取るのは、非同期ローカル処理の場合と、非同期転送処理の最終の受け取りサーバである場合である。

- Response Spool Manager:

プログラム名は spoolManager である。処理の核となる部分はクラス Spool に実

装されている。

このプログラムは Aysnc Service Driver からレスポンスを受け取り、スプールに保存する。Message Driver から Message ID のみの SOAP リクエストメッセージを受け取った場合、リクエストの Message ID とレスポンスメッセージにある Message ID を照合し、合致する場合はそのリクエストを Message Driver に送る。

- Forwarder:

プログラム名は fwderCreator である。処理の核となる部分はクラス Forwarder に実装されている。同期転送メッセージの場合は、Message Driver から受け取ったメッセージを他の SOAP サーバに転送してレスポンスを受け取り、そのレスポンスメッセージを Message Driver に返す。

非同期転送メッセージの場合は、Forward Queue Manager から受け取ったメッセージを他の SOAP サーバに転送してスレッドを終了する。

- Forward Queue Manager:

プログラム名は queueManager_fwd である。プログラムの実体は Request Queue Manager と同じく queueManager であるが、引数 1 をつけて実行することにより、非同期モードで動作する。処理の核となる部分はクラス Queue に実装されている。

このプログラムは Message Driver から非同期転送リクエストメッセージを受け取り、複数のメッセージをキューに保存し、順に Forwarder に送る。

- File ID Manager:

プログラム名は idManager である。処理の核となる部分はクラス IdManager に実装されている。このプログラムは SOAP メッセージをファイルに保存して永続化し、そのファイル名を File ID として返す。また、File ID を受け取った場合は、実際の SOAP メッセージに変換して返す。

各プログラム間を受け渡される SOAP メッセージは、実際の SOAP メッセージではなく、このプログラムが作成した File ID である。

- SSML Attribute Manager:

プログラム名は ssmlAttrMgr である。処理の核となる部分はクラス SSMLAttrMgr に実装されている。このプログラムは SOAP メッセージに含まれているタグの値または属性値を簡単に取得・設定するためのものである。

サーバ内プログラムがメッセージの判別・作成を行うために使用する。

- TTL Manager:

プログラム名は ttlManager である。処理の核となる部分はクラス TTLManager に実装されている。このプログラムは SOAP メッセージの最大処理待ち時間 TTL (Time To Live) を監視するためのプログラムである。

TTL を超過したメッセージは Request Queue Manager, Forward Queue Manager, Response Spool Manager から削除される。

- Reload Server Conf:

プログラム名は reloadServConf である。処理の核となる部分はクラス SrvConfAttrHandler に実装されている。このプログラムは設定ファイルの再読み込みの際に使用される。

- Server Configuration Attribute Manager:

プログラム名は srvConfAttrMgr である。処理の核となる部分はクラス SrvConfAttrMgr に実装されている。

このプログラムは/var/tmp/OpenSOAP/conf/server_conf/ 以下にあるサーバ設定ファイル forwarder.conf, backward.conf, signature.conf の設定を読み込み、Message Driver などからの設定ファイルの内容の問い合わせに応答する。

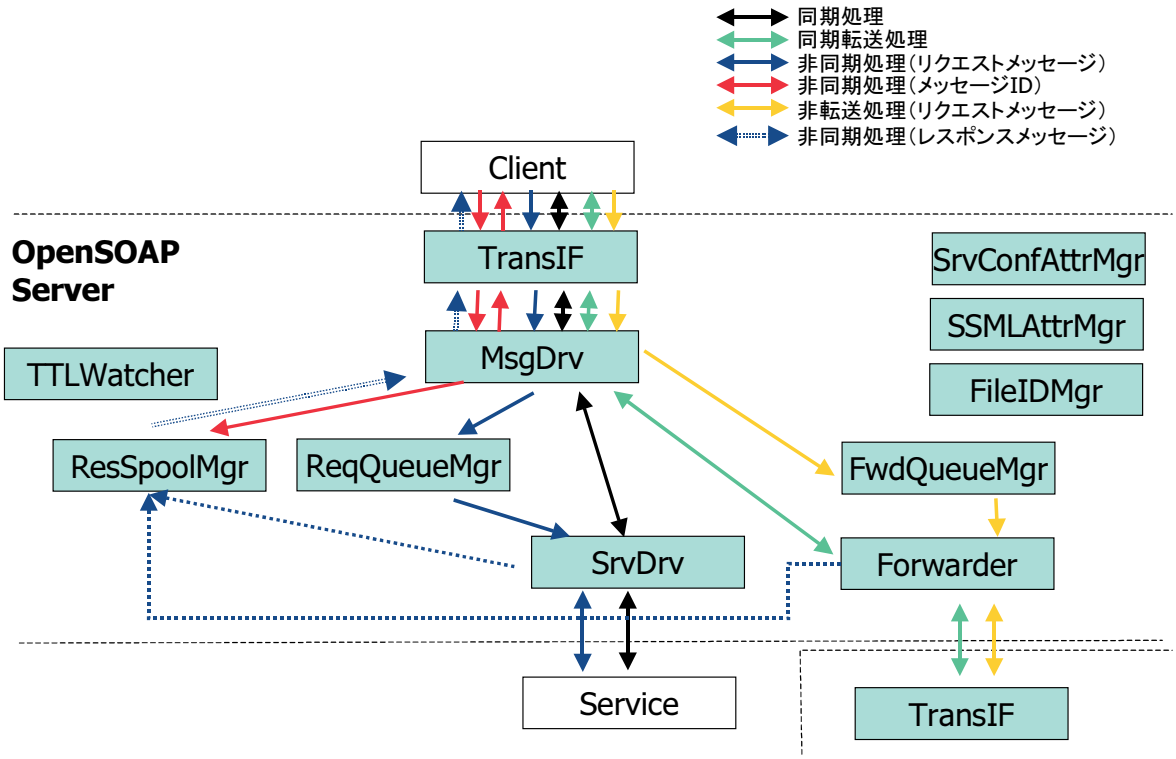

図 5.4.1 OpenSOAP サーバ・内部プロセス関係図

- SSML Attribute Manager:

プログラム名は ssmlAttrMgr である。処理の核となる部分はクラス SSMLAttrMgr に実装されている。/var/tmp/OpenSOAP/conf/ssml/ 以下にある SSML とよばれる OpenSOAP サービスの設定ファイルのを読み込み、Service Driver などからの設定ファイルの内容の問い合わせに応答する。

5.4.2. サーバ内処理手順（プロセス間関係図）

図 5.4.1 は、5.4.1 節で説明した各プロセスの関係図を示している。図中の矢印は、メッセージの流れである。要求される処理手順（同期ローカル処理、非同期ローカル処理、同期転送処理、非同期転送処理）によって、プロセス間の連携体制も異なり、主に Message Driver が判断しこれを統括管理する。

図 5.4.2 は、全ての種類の処理の基本となる同期ローカル処理の場合のフローである。Transport Interface を通過してサーバ内に取り込まれたメッセージは、Message Driver が同期処理が要求されていることを判断し、サービスプログラムとの通信部である Sync Service Driver に送られる。サービスプログラムからレスポンスメッセージを受けとったサーバは、同じプロセス通過手順をとり、クライアントプログラムへ返信する。

図 5.4.3 は単独のサーバ内で非同期処理を行う場合のフローである。Message Driver は、非同期要求であると判断すると、クライアントに対して Message ID を SOAP ヘッダに挿入した空のメッセージを返信する。同時に、非同期処理を要求されるサービスは、その処理実行時間が長いことが予想され、Async Service Driver

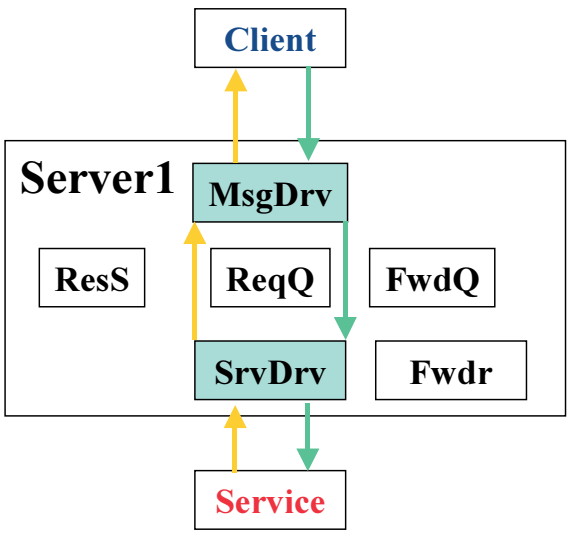

図 5.4.2 条件別処理フロー（同期ローカル処理）

へ受け渡す前に、Request Queue Manager で待ち行列に入れられる。サービスからのレスポンスメッセージは、同期処理と異なり、Response Spool Manager で、クライアントからの受け取り要求待ちのリストに加えられる。クライアントは、サーバから返された Message ID 入りのメッセージを再度 OpenSOAP サーバに送信することで、Response Spool Manager 内のメッセージの有無をポーリングし、既に処理結果がリストに照合されればレスポンスメッセージを受信することができる。

次に、図 5.4.4 は、複数の（Open）SOAP サーバ間で同期的に転送処理を行う

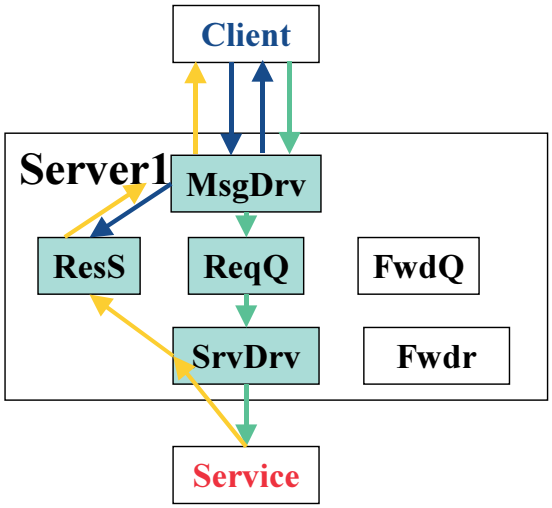

図 5.4.3 条件別処理フロー（非同期ローカル処理）

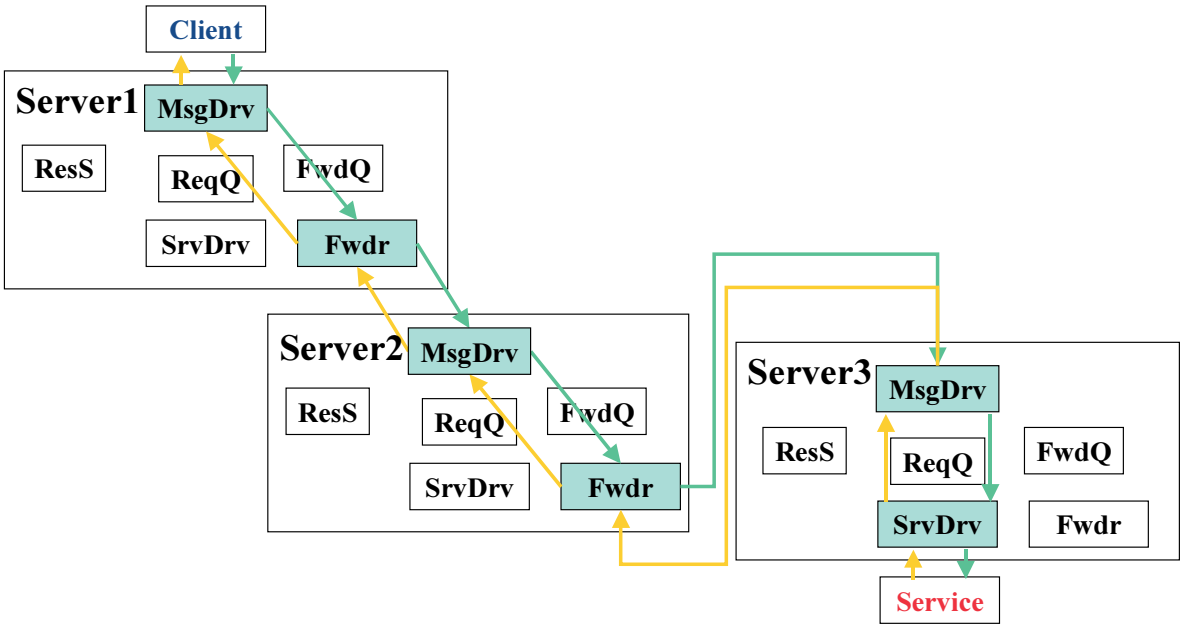

図 5.4.4 条件別処理フロー（同期転送処理）

場合のフローをあらわしている。Message Driver は、クライアントメッセージの内容、もしくはサーバ設定ファイルの内容から、転送処理要求であると判断すると、Forwarder にメッセージを渡す。Forwarder は、他のサーバから見ればクライアントの役割を果たし、転送処理が多段になる場合はこれを繰り返す。最終的にサービスプログラムを管理し、リクエストを処理可能なサーバは、同期ローカル処理をもってこれを処理する。多段処理もセッションを通じたままで行うため、同じ道順を辿ってレスポンスメッセージはクライアントに返信される。

図 5.4.5 は、この転送処理を非同期的に行う場合のフローを示している。クライアントに Message ID を返信すると同時に、転送のための待ち行列、Forward Queue Manager に送り、Forwarder が待ち行列内のリクエストを順次転送先サーバに送信する。最終サーバからのレスポンスの返信にもこの手順を用い、転送元サーバに順次返信していく。始めにクライアントからのリクエストを受信したサーバまでレスポンスが返信されたら、非同期ローカル処理と同様に、Response Spool Manager のリストに挿入される。

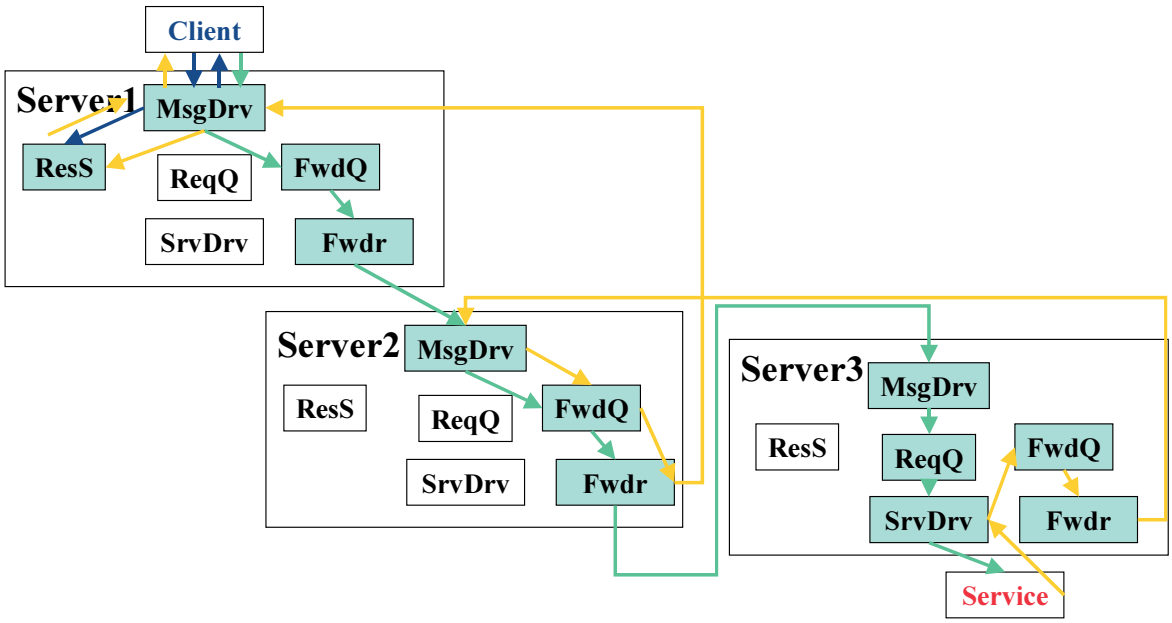

図 5.4.5 条件別処理フロー（非同期転送処理）

5.5. OpenSOAP サーバ運用マニュアル

5.5.1. OpenSOAP サーバ制御コマンド

UNIX コマンドにおける OpenSOAP サーバの制御コマンドは opensoap-server-ctl である。制御コマンドの引き数とその動作を以下に示す。

start:　　OpenSOAP サーバの開始。
stop:　　OpenSOAP サーバの終了。
reload:　　設定ファイルの再読み込み。

opensoap-server-ctl は 、 srvDrvCreator, srvDrvCreator_async, queueManager, spoolManager, fwderCreator, queueManager_fwd, idManager, ssmlAttrMgr, ttlManager, serverConfAttrMgr, ssmlAttrMgr の起動・停止を行う。

これらのプログラムは、起動時に/var/tmp/OpenSOAP/run 以下にプロセス ID を書き込む。

opensoap-server-ctl の stop コマンドが発行されると、このプロセス ID が書かれたファイルを基にプロセスの停止が行われる。

opensoap-server-ctl start を続けて２回以上実行すると、プロセス ID ファイルが上書きされてしまうので、stop を行っても OpenSOAP サーバのプロセスが残ってしまう。この場合は ps コマンドで OpenSOAP サーバのプロセスを探して手動で kill を行わなければならないので注意すること。

reload コマンドは srvConfAttrMgr, ssmlAttrMgr の停止･起動を行う。これにより、後述する OpenSOAP サーバ関連設定ファイルの再読み込みを行い、サービスの追加・削除や設定変更に対応する。

5.5.2.　OpenSOAP サーバ設定ファイル

/var/tmp/OpenSOAP/conf/server_conf/ 以下の各設定ファイルについて説明する。

- forwarder.conf:

この OpenSOAP サーバで SOAP メッセージを処理できなかった場合に、処理を任せるために転送する先のサーバの URL を設定する。

forwarder.conf の例を以下に示す。

```
<?xml version='1.0' encoding='UTF-8' ?>
<forwarder xmlns='x-schema:ForwarderConfSchema.xml'>
<protocol>http</protocol>
<hostname>fwdhost.opensoap.jp</hostname>
<port>80</port>
<cgi_location>/cgi-bin/soapInterface.cgi</cgi_location>
</forwarder>
```

図 5.5.1 設定ファイル forwarder.conf の例

forwarder.conf では、転送先の URL をプロトコル・ホスト名・ポート番号・OpenSOAP Interface へのパスに分けて設定する。

現在のところプロトコルとして指定可能なものは http のみである。ポート番号が指定されていない場合、80 がデフォルトとなる。

同期処理の場合、SSML ファイルと SOAP リクエストメッセージを照らし合わせ、送られてきたリクエストを処理できるサービスがローカルサーバにない場合はこの URL にリクエストメッセージを送信する。

同期処理の場合は接続を切らずにセッションを維持してレスポンスを待つ。非同期転送処理のうち以下のように最大転送台数の指定がなされた場合も、この URL にリクエストメッセージをおくる。

サーバープロセスの Forwarder は、次のサーバに送る際に SOAP ヘッダの <hopcount>要素の値を-1 して SOAP ヘッダを更新し、次のサーバに転送する。(図 5.5.2)

hopcount の値が 1 である SOAP リクエストが届いたサーバにおいて対応できるサービスが存在しない場合は、SOAP フォールトを作成し、前のサーバに順にレスポンスを戻す。この時に使用される返送経路の情報は、下記で説明する backward.conf を用いて作成される。

- backward.conf:

非同期転送の際にレスポンスを返してもらうためのこの OpenSOAP サーバの URL を設定する。

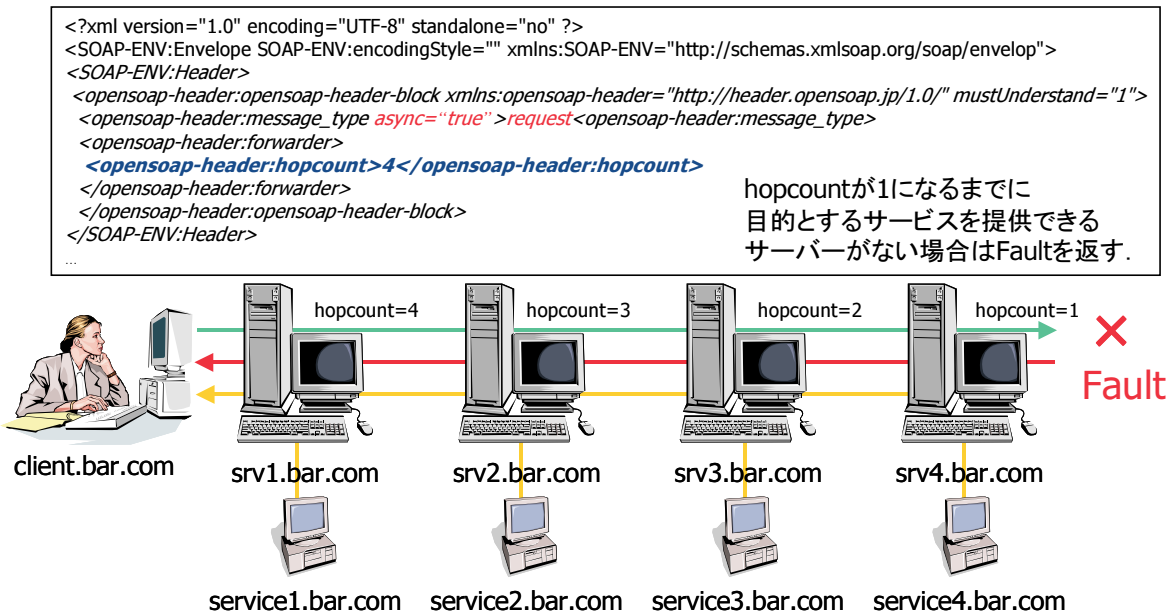

図 5.5.2 OpenSOAP サーバを利用した転送処理のための SOAP ヘッダ拡張と処理の流れ

backward.conf の例を以下に示す。

```
<?xml version="1.0"?>
<backward xmlns='x-schema:BackwardConfSchema.xml'>
<protocol>http</protocol>
<hostname>192.168.1.1</hostname>
<hostname>myhost.opensoap.jp</hostname>
<port>80</port>
<cgi_location>/cgi-bin/soapInterface.cgi</cgi_location>
</backward>
```

図 5.5.3 設定ファイル backwarder.conf の例

backward.conf の内容は SOAP ヘッダに<backward_path>を追加する際に使用される。非同期転送処理の場合、一度セッションが閉じられるためレスポンスメッセージの返送経路を記録するために図 5.5.4 に示すように転送サーバを経由するごとにヘッダの<backward_path>要素の追加・更新が行われる。<backward_path>要素は、レスポンス時にレスポンスメッセージが同じ転送経路を逆にたどるために、転送経路のサーバがヘッダ内容を更新すると同時に、サーバ内でレスポンスの返送先サーバの URI を保持するために用いられる（図 5.5.5）。ここで同時に追加される<received_path>要素は主にルーティング時のループなどの問題を検出するために用いられる。

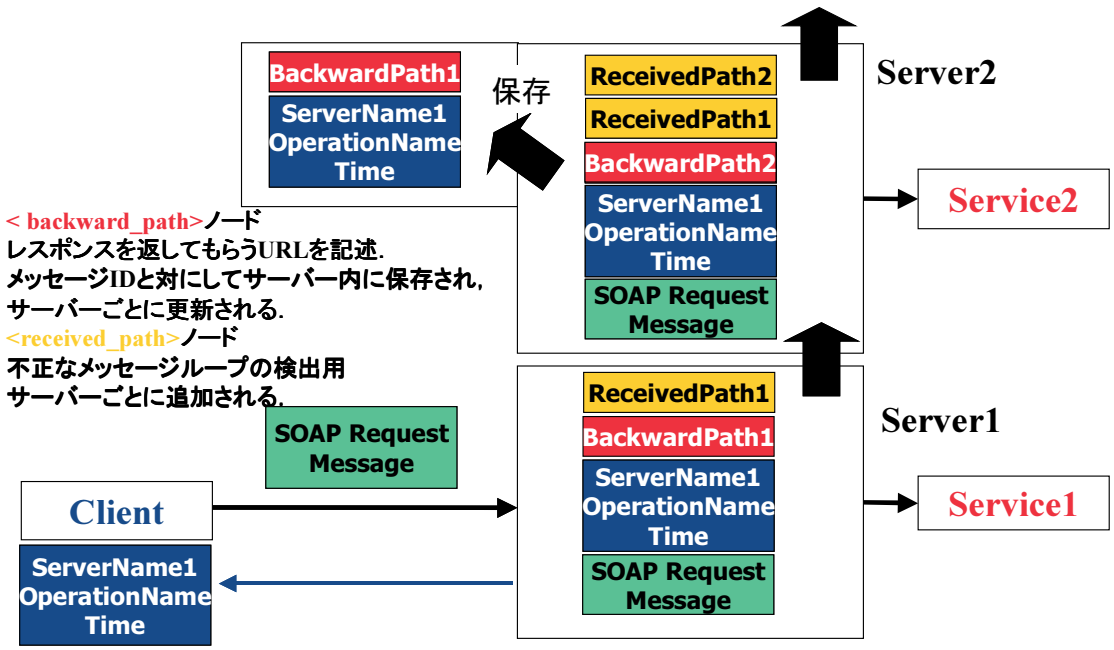

図 5.5.4 非同期転送処理に伴う拡張ヘッダ追加（リクエスト時）

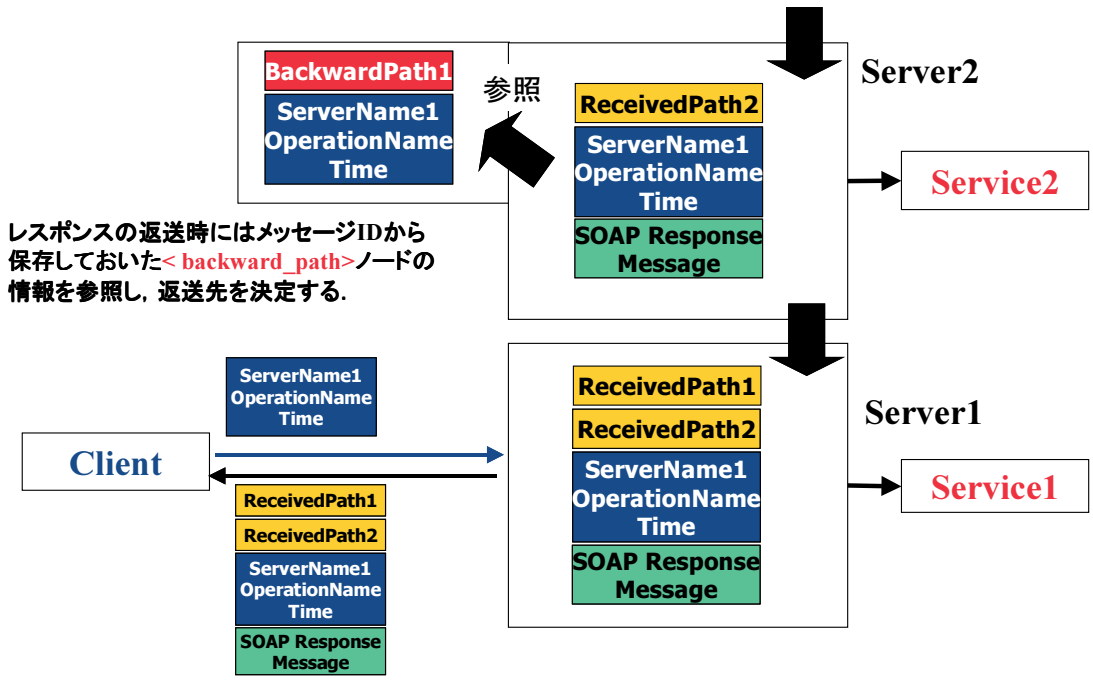

図 5.5.5 非同期転送処理に伴う拡張ヘッダ参照（レスポンス時）

この設定ファイル backward.conf の形式は forwarder.conf に似ているが、ホスト名を複数指定できる点が異なる。最初に指定されたホスト名が自サーバにおいて拡張 SOAP ヘッダの<backward_path>要素を設定する際に使用される（図 5.5.6）。

これ以降に設定されるホスト名は、メッセージの無限ループを検出する際などに、すべて参照されることでこれらを防止するためのものである。そのため 1 台のホストに複数のマシン名・IP アドレスがある場合は、全て列挙しておく必要がある。

- signature.conf:

<add_signature>の値を true にすると OpenSOAP サーバの発行するメッセージにそのサーバの署名を付加する。

デフォルトでは false になっている。値を true に設定するとサーバは PKI 方式にもとづき、サーバ内にある秘密鍵ファイル
/var/tmp/OpenSOAP/Data/Security/privKey.pem
を用いて署名を行う。OpenSOAP サーバから署名付きのメッセージを受け取ったクライアント、もしくはサーバ、もしくはサービスは privKey.pem に対応する公開鍵を用いることにより確かに特定のサーバから送られてきたメッセージである

かどうかを検証できる。

PKI 認証は OpenSOAP API を使って行うことが可能である。署名・認証の詳細はセキュリティの項で述べられている。

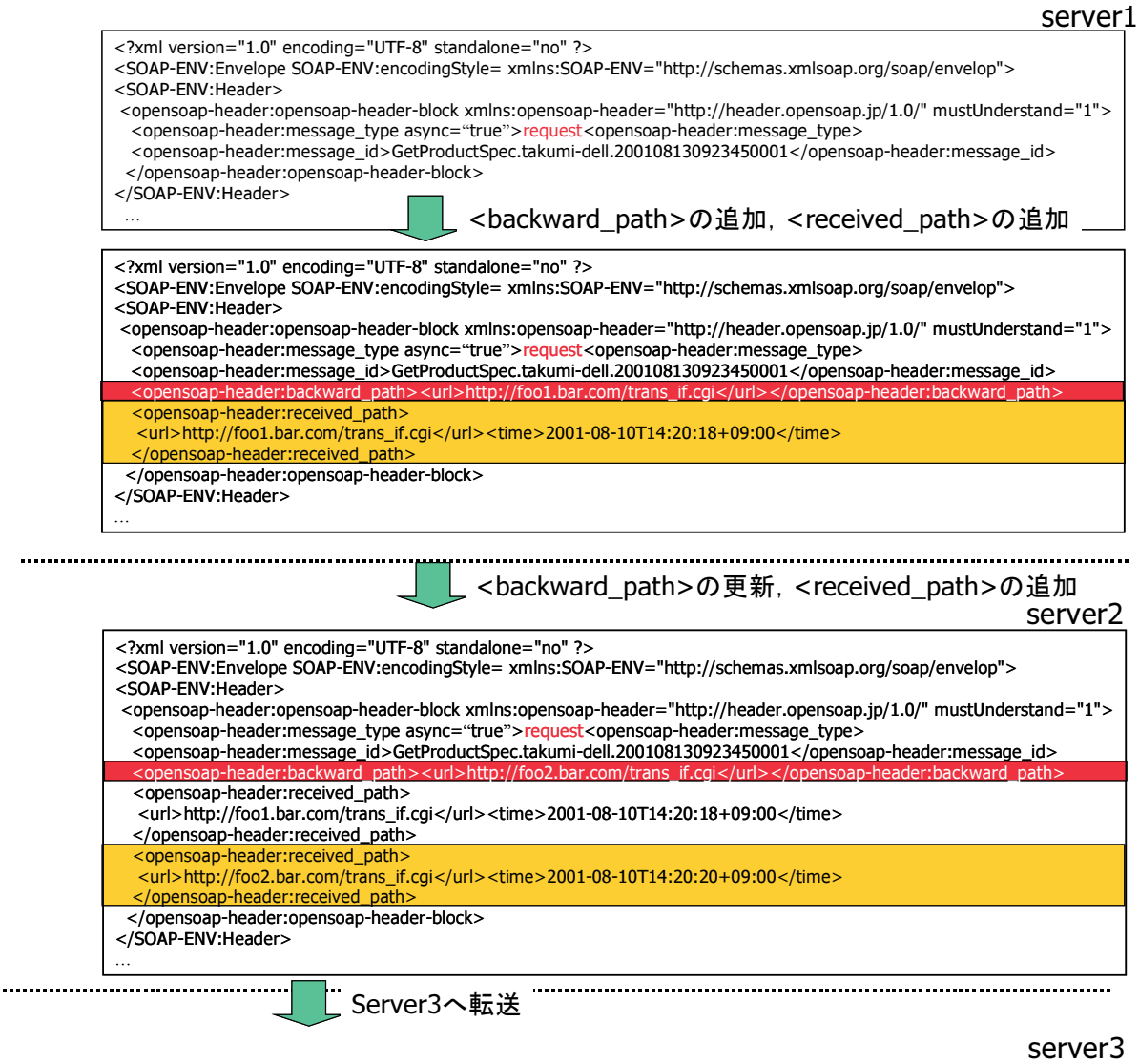

図 5.5.6 拡張 SOAP ヘッダの詳細例

5.5.3.　OpenSOAP サービス設定ファイル SSML

本節では、OpenSOAP サーバ上で稼動するサービスプログラムについて、OpenSOAP サーバとの連携方法を記述する独自仕様の設定ファイル、SSML (Soap Service Markup Language)の詳細について、以下の Calc.ssml ファイル（図 5.5.7）を例に説明する。

```
<?xml version='1.0' encoding='UTF-8' ?>
<SSML xmlns="x-schema:ssmlSchema.xml">
        <service name='Calc' xmlns:service="http://namespaces.opensoap.jp/calc/">
            <connection name='Calc1'>
                <Socket hostname='localhost' port='8765'/>
                <asynchronizedTTL >8000</asynchronizedTTL>
                <synchronizedTTL count="second" >20</synchronizedTTL>
                <MaxProccessNumber>5</MaxProccessNumber>
            </connection>
            <connection name='Calc2'>
                <StdIO>
                    <exec prog='/usr/local/bin/CalcCpp2' option='-u -N' />
                </StdIO>
                <synchronizedTTL count="second" >10</synchronizedTTL>
                <MaxProccessNumber>12</MaxProccessNumber>
            </connection>
            <connection name='Calc3'>
                <FIFO send_pipename='/home/sato/soap/var/SERVER_TO_SERVICE'
                        receive_pipename='/home/sato/soap/var/SERVICE_TO_SERVER'>
                    <exec prog='../service/service' />
                </FIFO>
                <synchronizedTTL count="second" >5</synchronizedTTL>
                <MaxProccessNumber>12</MaxProccessNumber>
            </connection>
            <connection name='Calc4'>
                <IPC KeyFilename='satok'/>
                <synchronizedTTL count="second" >10</synchronizedTTL>
                <MaxProccessNumber>12</MaxProccessNumber>
            </connection>
            <connection name='Calc5'>
                <Module name='satomodule' configfile='/home/sato/.satomodule' >
                    <exec prog='/usr/local/bin/CalcCpp2' option='-u -N' />
                </Module>
                <synchronizedTTL count="second" >10</synchronizedTTL>
                <MaxProccessNumber>12</MaxProccessNumber>
            </connection>
            <connection name='Calc6'>
                <COM dispID='12345' >
                    <exec prog='/usr/local/bin/CalcCpp2' option='-u -N' />
                </COM>
                <synchronizedTTL count="second" >10</synchronizedTTL>
                <MaxProccessNumber>12</MaxProccessNumber>
            </connection>
            <operation type ='Calc1'>add</operation>
            <operation type ='Calc2'>sub</operation>
            <operation type ='Calc1'>Add</operation>
            <operation type ='Calc1'>Subtract</operation>
            <operation type ='Calc1'>Multiply</operation>
            <operation type ='Calc1'>Divide</operation>
            <operation type ='Calc1'>Sub</operation>
            <operation type ='Calc1'>Mul</operation>
            <operation type ='Calc1'>Div</operation>
            <fault signature='1' />
            <MaxProccessNumber>15</MaxProccessNumber>
        </service>
</SSML>
```

図 5.5.7 Calc.ssml

SSML(Soap Service Markup Language)の最上位要素は<SSML>であり、サービス名を定義した<service>要素をひとつ含む。SSML ファイル名は便宜的にこのサービス名と同一にする。

<service>ノードにはひとつ以上の<connection>要素と、任意の数の<operation>要素、ひとつの<fault>要素、ひとつ以下の<MaxProccessNumber>要素を含んでいる。

<connection>要素では、サービスとの接続方法を記述する。接続方法には、

・socket:	<Socket hostname='サービスプログラムのあるホスト名' port='ポート番号'>
・標準入出力:	<StdIO>（要素は実行プログラム名とオプション）
・名前つきパイプ:	<FIFO>（1.0 版では未実装）
・IPC:	<IPC>（1.0 版では未実装）
・COM:	<COM>（1.0 版では未実装）
・その他接続モジュール:	<Module>（1.0 版では未実装）

を指定することができる。

さらに、非同期プロセスでのタイムアウト時間(秒)<asynchronizedTTL>、同期プロセスでのタイムアウト処理<synchronizedTTL>（count 属性で秒'second'、またはメッセージのホップ回数'hoptimes'（α版では未実装）を指定可能。）を指定する。

これらは同時に指定することができ、リクエストメッセージによる同期・非同期の指定によって使い分けることができる。

<MaxProccessNumber>要素で、この接続方法による最大接続数を指定でる。（1.0 版では未実装）

<operation>要素では、<connection>の name 属性で指定した接続方法を指定してオペレーション名を記述する。ひとつのサービスに接続法の異なる複数のオペレーションがあってもかまわない。ここでは、そのサービスに対してクライアントから呼び出される可能性のあるオペレーション名を全て記述する必要がある。

<fault>要素ではサーバが返す Fault メッセージにサーバの署名を付加するかどうかを signature 属性('0' or '1')で指定する。

<MaxProccessNumber>でサービス全体のプロセス起動数を制限できる。（1.0 版では未実装）

5.6.　まとめ

本研究では、複数のクライアントと複数のサービス間でメッセージ通信を可能

とするためメッセージの管理を行う OpenSOAP サーバの開発を行った。

メッセージの処理としては、他の SOAP 実装でサポートされている同期処理機能を標準で備え、さらに処理に時間を要するサービス用として、クライアントが再度サーバに接続した時点でレスポンスを返す非同期処理も実装した。またファイアウォールの内外との SOAP メッセージをやり取りするための同期転送処理を実装し、転送経路指定と最大転送台数指定によりメッセージ転送の方法を指定できるような機能を備えている。また非同期の場合にもこの機能を使用可能で、同じく転送経路指定、転送台数指定による経路指定を可能としている。非同期処理・同期転送処理・非同期転送処理は、これまでにある SOAP 実装でもサポートされていない機能であり、OpenSOAP プロジェクトが構築したサーバーシステムの大きな特徴であるといえる。またこれらの機能はビジネスソリューションとして OpenSOAP の成果物を使用する上で必須の機能であるとともに、他の実装に比べて非常に有利な機能であるといえる。

これに加え、電子商取引の際に必要となるメッセージのコミットとロールバックをサポートするためのトランザクション管理機能も実装している。またサーバ内で処理されるメッセージの流れを記録するログ管理機能を備えている。さらにセキュリティ管理機能の成果を利用し、サーバでメッセージが処理されたことを証明する署名の機能も備えている。サーバは非同期処理の際の処理待ち行列を備えており、サーバが不正に終了した場合に処理を終えていないリクエストメッセージを保存しておく機能も備えている。

OpenSOAP サーバは Linux, Solaris 8, NetBSD, Windows 2000 での動作が可能であり、サーバの開始・終了・再起動を簡便に行うためのツール群も同梱して提供される。

これらの成果物は、インターネットを通じてオープンソースとして公開しており、誰もがダウンロードして利用可能となっている。