ESB環境での高可用性設計

丸田 康正　山口 崇　趙 京揚

High-Availability Designs in ESB Environments

Yasumasa Maruta　Takashi Yamaguchi　Kyohyoh Choh

現在SOA（Service Oriented Architecture）は実装フェーズに入ってきている. SOAシステムの中核をなすエンタープライズ・サービス・バス（ESB）を構築し, 実運用を行っていくためには非機能要件の充足が必要不可欠である. そこで本論文ではWebSphere® Application Serverバージョン6のサービス統合バス（SIBus：Service Integration Bus）のWebサービス拡張であるSIBWS（SIBus for Web Services enablement）を具体例として, 非機能要件の中でも特に高可用性の観点からESBシステムを構築する際のトポロジー・デザイン, 耐障害設定などの具体的実現手段を述べる.

SOA (Service-Oriented Architecture) has entered the implementation phase. In order to configure an Enterprise Service Bus (ESB) forming the core of an SOA system, and to go as far as operating it in a real environment, it is absolutely necessary to fulfill non-functional requirements. In this paper, using the real example of the SIBWS (SIBus for Web Services enablement), being the web service extension of the WebSphere® Application Server V6 Service Integration Bus (SIBus), we discuss specific means for achieving topology design, fault-tolerant settings and other factors when building ESB systems from the viewpoint of non-functional requirements, and in particular from the viewpoint of high-availability.

Key Words & Phrases：高可用性, エンタープライズ・サービス・バス, サービス統合バス, WebSphere Application Server
high availability, ESB, SIBus, WebSphere Application Server

1. はじめに

システム構築にあたり, 機能要件の設計と非機能要件の設計というように役割を分離させる『関心の分離』に注目が集まっている［1］. SOA（Service Oriented Architecture）の中核をなすエンタープライズ・サービス・バス（以下ESB）とは, 分散配置された各種サービスを疎結合で統合し, クライアントから透過的にアクセスできる通信経路として位置付けられているが, そこで様々なアプリケーションやシステムを統合し基幹システムとして構築するためには, ESBの非機能要件の実現が重要になってくる. 非機能要件の中でも特に, システムの高可用性は基幹システム構築の上で不可欠であるといえる. 本論文では, ESBを用いてサービス統合を実現するシステムにおいて, システム運用時間の延長・ノンストップ運用といった高可用性の実現手法を述べる（その有効性は実装によって確認してある）.

提出日：2005年8月31日　　再提出日：2006年9月14日

なお, ESBという言葉は単体製品を指すものではないが, 本論文ではESB機能を実現するコンポーネントとして, WebSphere® Application Server（以下WAS）バージョン6より提供されるようになったサービス統合バス（SIBus：Service Integration Bus）のWebサービス拡張であるSIBWS（SIBus for Web Services enablement）［2］に関して述べる.

以下、2章で、SIBWSシステムにおける高可用設計領域として三つに大別する. 3章から5章で、各領域の具体的な設計手法を紹介するとともに, 5章の中では, より高度な高可用実現手法について提案する. さらに6章で、SIBWSを拡張した構成について紹介する. 最後に、7章で、高可用設計に関するまとめを述べる.

2. SIBWS高可用設計における課題と領域

SIBWSの高可用性設計と一口に言っても考慮すべき対象は多岐に渡る. ここではまずSIBWSを使用してESBシステムを構築する場合の一般的なトポロジーを解説し, 後半の理解の助けとしたい.

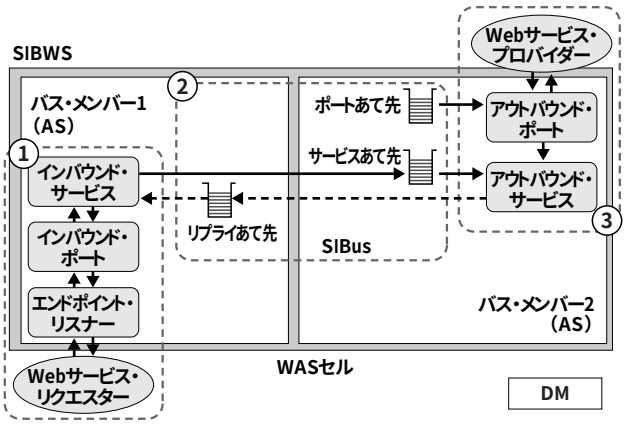


図1.　SIBWS略図

WASのセル環境は、複数アプリケーション・サーバー（以下AS）および管理用デプロイメント・マネージャー（以下DM）より構成される. SIBWSの基盤となるSIBusとは, WASのセル環境上に, 分散配置されたキュー・マネージャーおよびキューの連携システムである. SIBusは複数のASにまたがって構成することができ, ASやASクラスターをバス・メンバーとして登録する. SIBWSはこのSIBusに対し, 入り口部分と出口部分にWebサービスのインターフェースをかませたものである（**図1**参照）. SIBWSを使用するリクエスターは, キューに対するAPIアクセスの代わりにWebサービスを呼び出すことにより, Webサービス・プロバイダーの実際のロケーションを考慮せずに呼び出せるというESBの利点を享受できるのである［3］.

SIBWSのようなESB機能が新たに介在することにより, ルーティングなどの役割からクライアント・アプリケーションを解放し, コンポーネント間の疎結合を実現することができる. しかし, 要件に応じた高可用性を提供しなくてはならない基盤運用担当者にしてみれば, 今までも十分複雑な分散システムに対し, Webサービスコンポーネントやキュー・マネージャーなど考慮すべきコンポーネントが飛躍的に増えるという課題が新たに発生する.

この課題に取り組むにあたり, まず具体的にSIBWSシステムを構築する際の, 高可用性を考慮しなくてはならない領域を大別すると, 以下に分類される（番号は図1の破線で示した領域番号に対応）.

① WebサービスリクエスターからSIBWS
② SIBWS内部のSIBus
③ SIBWSからWebサービス・プロバイダー

上記いずれの部分がダウンした場合でも, SIBWSは適切にリクエストを届けることができず, リクエスター, ひいてはエンドユーザーまでシステムの適切な使用ができなくなる. 次章以降では各領域における高可用性対処方法について述べる.

3. WebサービスリクエスターからSIBWSにおける高可用性設計

初めに, WebサービスリクエスターとSIBWS間の経路に関する高可用性を検討する. SIBWSはSOAP/HTTPおよびSOAP/JMS（Java™ Messaging Service）［4］という二つのリクエスト・プロトコルに応じたエンドポイント・リスナーでリクエストを受け付ける. エンドポイント・リスナーはバス・メンバーのAS上で稼動するエンタープライズ・アプリケーションであるため, AS障害のケースや, アプリケーションが利用するリソースの障害に対して対応する必要がある. これらについて, それぞれのプロトコル別に検討して行きたい.

3.1　SOAP/HTTPエンドポイント・リスナー使用の場合

SOAP/HTTPを使用する場合, リクエストはインバウンド・サービスが提供するサーブレットにより処理される. 通常のHTTPリクエストと同様にリクエストはWebサーバーで受け付けられ, Webサーバー・プラグイン経由でインバウンド・サービスの配置されたASに送信される.

この領域での高可用性は, インバウンド・サービスを配置するASをクラスター構成とすることで実現できる（**図2**参照）. これでインバウンド・サービスが稼動するASクラスター・メンバーの一部に障害が発生した場合でも, Webサーバー・プラグインが検知およびリルートを行うため, Webサービス・リクエスターとSIBWS間経路の高可用性は確保される. さらにプラグインが稼動するWebサーバーの高可用性確保のために, 複数Webサーバーを配しその前段に負荷分散装置を配置することを推奨する.

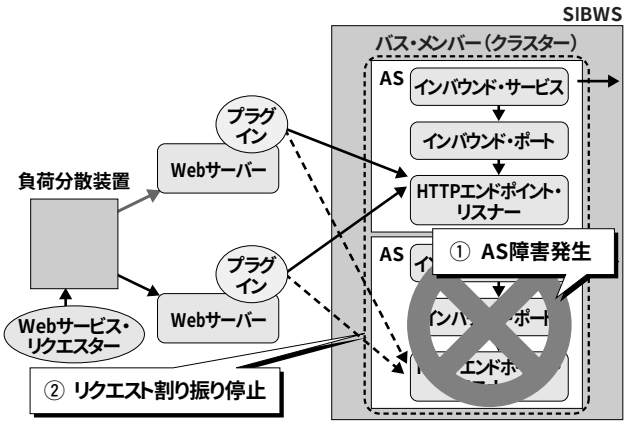


図2.　SOAP/HTTPエンドポイント・リスナー

3.2　SOAP/JMSエンドポイント・リスナー使用の場合

WebサービスリクエスターがSOAP/JMSを使用してリクエストを送信する場合, リクエストはまずキューが受信する. その後キューからメッセージ・ドリブン・

ビーン(以下MDB)[5]で実装されたインバウンド・サービスがメッセージを読み取り, 処理を継続するというフローとなる. インバウンド・サービスの受信キューは, AS上で稼動するメッセージング・エンジン(以下ME)内に作成されるため, 高可用性を高めるための対処としては, 受信キューの稼動するMEの高可用性構成および, インバウンド・サービスの高可用性構成を検討する必要がある.

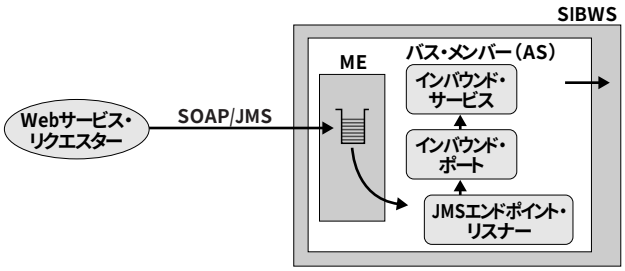


図3. SOAP/JMSエンドポイント・リスナー

MEおよびインバウンド・サービス双方の高可用性実現方法として, 稼動するASをクラスター構成とすることができる. ASクラスター環境に構成されたMEはASクラスター内の一つのAS上で稼動するが, 仮にMEをホストしているASが障害で停止した場合でも継続稼動している別ASに自動的にフェイル・オーバーされるため, 可用性は維持される(「MEのクラスター化機能」). ASクラスターのAS間ではメッセージのパーシスタントDBを共有し, メッセージ・パーシスタント・オプションを「保障パーシスタント」に設定することで代替機でのメッセージの継続処理も可能となる. こうしたフェイル・オーバー機能はWASのHAマネージャー(High Availability Manager)により提供される. その定義情報などはデフォルトで提供されており, 特に必要がない限り設定に手を加える必要はない(**図4**参照).

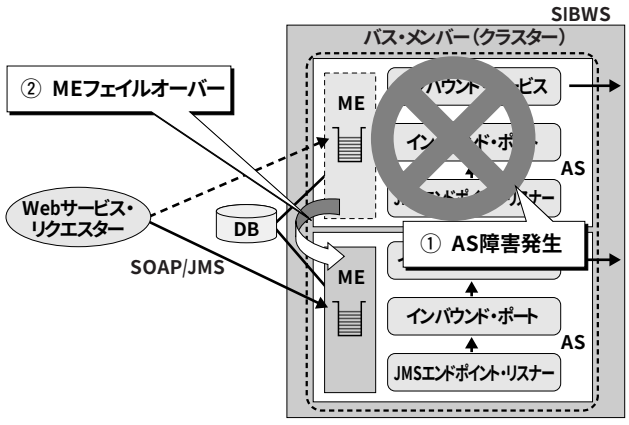


図4. MEのクラスター化機能

MEおよびインバウンド・サービスが稼動するASを単にクラスター構成した場合では, 一時点にASクラスター内の一つのAS上のMEのみがアクティブとなるが, さらなる拡張構成として, ASクラスター内の複数AS上で同時に複数MEを並行稼動させることにより, 可用性に加えスケーラビリティーを実現させる構成も可能である. これは「MEのパーティショニング機能」と呼ばれるが, クライアントとなるWebサービス・リクエスターの配置によって次のような考慮が必要となる.

Webサービス・リクエスターが, SIBWSが稼動するASと同じSIBusに属するAS上で稼動している場合, 複数MEにあるキューへのリクエストはラウンド・ロビンで処理される. しかし, リクエスターがSIBus外部に居る場合, リクエトは分散されず, 固定的に一つのMEのみが使われてしまう. これを避けるためには, リクエスターとSIBWSの間にTCPレベルの負荷分散装置を配置し, キュー接続ファクトリー定義に負荷分散装置が持つクラスターアドレスを定義することで複数MEに対してリクエストを分散する必要がある.

4. SIBWS内部のSIBus処理に対する高可用設計

次にSIBWS内部のSIBusに対する対処法を検討する. インバウンド・サービスとアウトバウンド・サービスの間を結ぶのは「あて先」, つまりこれもキューである. 従ってこの領域における高可用性対応の第一歩も, 3.2で述べたSOAP/JMS エンドポイント・リスナーと同様MEのクラスター化によって実現される.

SIBWSで構成されるあて先はアウトバウンド・サービス側ME上の「サービスあて先」および「ポートあて先」と, インバウンド・サービス側ME上の「リプライあて先」の三種類となる(図1参照). 従って一般的なインバウンド・サービスとアウトバウンド・サービスでASを分けた構成の場合, 高可用性を維持するためには二つのMEに関してクラスター構成を検討する必要がある.

また, 3.2で述べたようにMEのパーティショニング構成を行うことも可能だが, インバウンド・サービス側を複数ME構成とした場合, リプライあて先が複数存在することになるため, リクエストを投げた特定インバウンド・サービスのリプライあて先に対してアウトバウンド・サービスが正しくレスポンスを返せなくなるという問題がある. これにはインバウンド・サービスとアウトバウンド・サービスを同一のプロセス上に構成することで対処可能である(**図5**参照).

ME以外のコンポーネントでは, パーシタントDBがME稼動中にアクセスされるため, 可用性の検討をする必要がある. ただしDBの障害に関してはHAマネージャーの監視範囲外なので, 他の高可用ミドルウェアなどを使用した対応が必要である. その他, SIBWSコンポーネントはWSDL(Web Services Description

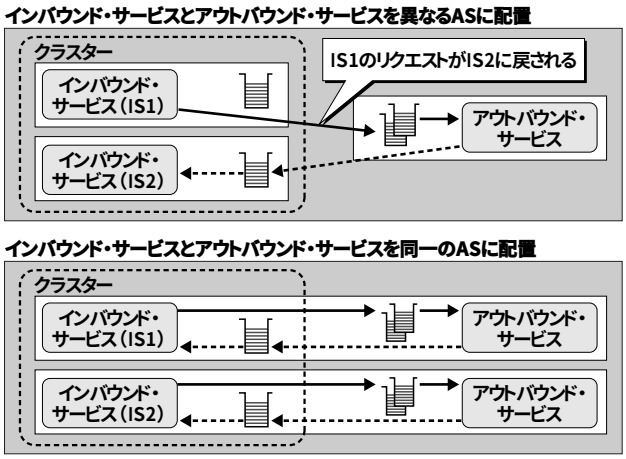


図5. リプライあて先の配置

Language）を保管するために，内部でSDO（Service Data Objects）リポジトリ・アプリケーションを介してSDOリポジトリ・データベースを使用する．従ってSIBWS内部の対処法としてSDOリポジトリ・データベース（セルに一つ存在）の二重化とSDOリポジトリ・アプリケーション（各ASおよび管理サーバーのDMに存在）の監視・障害対応も忘れてはならない．

また，MEのパーティショニングは，インバウンド・サービス，アウトバウンド・サービスが分かれている構成の場合，構成するMEの数も多くなる．さらに，ASクラスター内のMEの数の分，通常構成ではパーシスタントDBが必要となる．これはMEごとにスキーマを分けることでパーシスタントDBの共有が可能であるが，こうしたリソースの必要量なども考えた上で最適な高可用性対応構成を検討すべきである．

5．SIBWSからWebサービス・プロバイダーにおける高可用設計

SIBWS機能を使用してWebサービスを呼び出す際，最終的にWebサービス・プロバイダーを呼び出すのはSIBWSのアウトバンド・サービスである．いわばアウトバンド・サービスがWebサービス・プロバイダーを呼び出すWebサービス・リクエスターのように振る舞う．そのアクセスプロトコルはSOAP/HTTPだけではなく，SOAP/JMSプロトコルも使用される可能性がある．SIBWSからWebサービス・プロバイダーへの高可用性を考える時，これらのプロトコル別に検討する必要がある．従って当節では，まずそれぞれのバインディング・プロトコルにおける高可用性の考慮点について論じ，さらにメディエーションでの高可用性機能実現の一案を提示する．

5.1 基本的な高可用設計

SIBWSのアウトバンド・サービスからSOAP/HTTPでWebサービス・プロバイダーにアクセスする場合，2.2.1.1のSOAP/HTTPエンドポイント・リスナーで検討したHTTPの負荷分散構成と同等の検討を，アウトバウンド・サービスをWebサービス・リクエスターと置き換えて行う．

SOAP/JMSを使用する場合も，3.2のSOAP/JMSエンドポイント・リスナーで検討したMEのクラスター構成と同じ内容を検討すればよいのだが，ここで扱うべきWebサービス・プロバイダーはSIBusであるとは限らない．WebSphere MQ（以下WMQ）などの他のメッセージング・ミドルウェアを使用する場合は，メッセージング・ミドルウェアが提供する高可用性機能（MQクラスター機能など）を検討する必要があるが，本論文では対象外とする．

5.2 メディエーションを利用した高度な高可用性実現

アウトバウンド・サービスからWebサービス・プロバイダーの高可用性に関し，これまでバインディング・プロトコルごとに述べてきたが，これらは基本的に製品の機能によって実現できる．しかし，システムによっては，障害をアプリケーションでハンドリングし，より柔軟な障害対応を実現したいというケースがありうる．このようなエラーハンドリング・ロジックはWebサービス・リクエスター内に埋め込むこともできるが，冒頭でも述べたようにESBの狙いの一つである「関心の分離」を実現するため，SIBusのメディエーション機能を利用して，SIBusのあて先にプラグインする形で付加することもできる．本論文ではこの非機能要件をESB内のメディエーションという機能で実装する方法を提案する．

SIBusが提供するメディエーションとは，あて先にJavaコードを関連付けることで，メッセージの加工ができる機能である．これにより，データ変換やコンテンツベースのルーティングなどの機能を実現できる．この機能を活用して，Webサービス・プロバイダーで障害が発生しSIBWSに例外が戻された場合に，メッセージ内のリターンコードを解析し，その内容に応じてメッセージを正常に稼動している代替Webサービス・プロバイダーへ転送する仕組みを実現できる．

SIBWSを利用する場合，最終的にリプライのメッセージがインバウンド・サービスのリプライあて先を経由してWebサービス・リクエスターに戻される．従ってメディエーションを付加すべきあて先はリプライあて先となる．ただ例外が発生してリクエストを再送する場合でも，リプライあて先にはオリジナル・メッセージは戻されないため，何らかの仕組みで元のメッセージのコピーを保持する必要がある．

例えばWebサービス・プロバイダーを呼び出すポートあて先に別のメディエーションを付加し，そのメディ

エーションでWebサービス・プロバイダーを呼び出す前にリクエストメッセージを外部ファイルに出力，リプライあて先に付加するメディエーションがそのファイルを参照し元のリクエストメッセージを取得するという方法が考えられる．

具体的に作成した障害対応の流れの解説を行う．まずリクエストメッセージのコピーを保持するため，ポートあて先にメディエーション1を付加する．メディエーション1の中でメッセージをコピーし，コピーしたオブジェクトをシリアライズして外部ファイルに書き出す．次にリプライあて先に障害を処理するメディエーション2を付加する．メディエーション2の中でSOAPメッセージにエラー情報が含まれているSOAPFaultなどのエレメントをアクセスし，その内容を解析する．エラーの内容に応じて代替Webサービス・プロバイダーへ転送すると判別した場合，先ほどの外部ファイルからメッセージ読み込み，そのメッセージに次に転送するあて先をセットするという処理を行う．**図6**に簡単に処理の流れを示している．

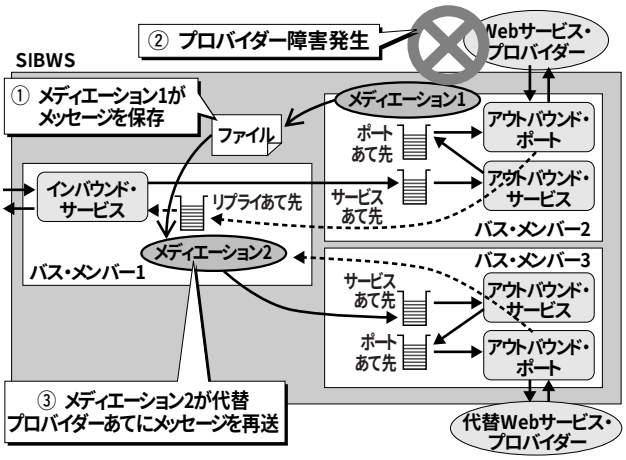


図6.　メディエーションによる障害対応

ここで注意しなくてはならないことが二点ある．一つ目は，書き出したメッセージの識別である．複数のクライアントからリクエストがある場合，保管した元のメッセージを一意に識別しなくてはならない．SIBWS内部でリクエストメッセージが作成される際，ApiMessageIdヘッダーに自動的にユニークなIDが付与されるが，リプライ・メッセージではCorrelationIdヘッダーに同じ値がセットされ，リクエストとリプライのメッセージが紐付けられる．従ってメディエーション1がリクエスト・メッセージをファイルに書き出す時にApiMessageIdも一緒に記録すれば，メディエーション2ではリプライ・メッセージのCorrelationIdヘッダーの値をキーに元のメッセージを検索できる．

二つ目は、障害判別の精度である．JAX-RPCの仕様では各ベンダー間の相互接続性を高めるため，システムエラーが発生した場合，ベンダー固有の例外ではなく，例外情報をRemoteExceptionまたはSOAPFaultExceptionマッピングしてリクエスターに返すので，システムエラーの種類に応じて細かく対応できない場合がある[6]．ただしユーザーが独自に定義した例外であれば例外情報がそのままがリクエスターにも返されるため，必要な場合ユーザー例外を定義することで対応する．

メディエーションはメッセージの加工やルーティングだけではなく，このような障害対応の一ソリューションとしても活用できる非常に有用な機能である．

6．拡張ESB構成での考慮点

これまでは，Webサービス・リクエスターからSIBWSを介して，Webサービスへアクセスするという経路で高可用性を検討してきた．拡張として，SIBWSと既存のWMQシステムを連携させる場合においての高可用性実現を検討する．

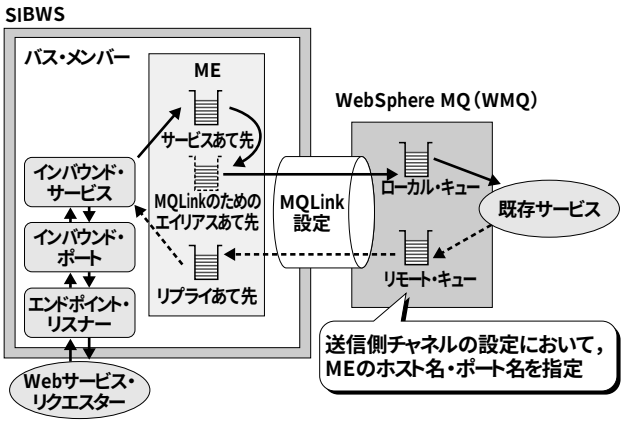


図7.　MQLink設定によるWMQとの連携

図7では，SIBusのMQLink設定を使用し，既存のWMQシステムにSIBWSを接続している例を示している．

MQLink設定のためには，メッセージ受信のために，WMQ側のローカル・キューに紐付くエイリアスあて先がSIBus側で必要となる．このエイリアスあて先にリクエストを送信する必要があるが，現在のバージョンでは，インバウンド・サービスは受け取ったリクエストをエイリアスあて先に直接送信することができない．この問題は，インバウンド・サービスとエイリアスあて先を結ぶためのあて先を用意し，メッセージの転送をすることで解決可能である．

MQLink設定により，リクエストを受け付けるエイリアスあて先に送信されたメッセージは，ローカル・キューに自動的に送信され，レスポンスを受け付けるリモート・キュー（WMQ側）に送信されたメッセージ

は, リプライあて先(SIBus側)に自動的に送信される. この仕組みにより, SIBWSとWMQシステムとの相互連携が実現する.

このように, WMQと連携するシステムにおいて, 高可用を考慮しなければならない場合が二つある. 一つはWMQに障害が発生した場合, もう一つはSIBus上のMEに障害が発生した場合である.

WMQに障害が発生した場合, WMQが提供するクラスター機能を使用したいところであるが, 残念ながら現在のところMQLink設定とWMQのクラスター機能は連携することはできない. WMQ障害に対してはHACMPなどの高可用製品を使用し, WMQの高可用性を確保しなければならない.

次に, SIBus上のMEに障害が発生した場合であるが, MEのクラスター化機能により対応する. WMQでは送信側チャネルの設定において, 接続するMEのホスト名・ポートを指定するが, この指定は一組のホスト名・ポートのみの指定となるため, MEがフェイル・オーバーし別のホスト名・ポート番号を使用して起動した場合, 接続できないという事態が発生する.

その解決策としては, ASクラスターの配下にある全てのMEは, 稼動する際に同じホスト名・ポートを使用するという手段が挙げられる.

次の二つの仕組みにより, この解決策が実現する.

- 全てのMEに同じポートを使用させるために、ASクラスター配下のMEは全て別々の筐体上で稼動させる
- MEが稼動する筐体において, MEのフェイル・オーバーに合わせて, WMQで指定しているIPアドレスを引き継ぐ

以上の仕組みを使用することで, 高可用性を確保した, SIBWS-WMQ連携システムが実現する.

7. おわりに

ESB機能としてSIBWSを使用したシステムの高可用性実現手段ついて述べた. 以下に, その要点をまとめる.

SIBWS高可用性実現の核はAS／MEのクラスター化である. SOAP/HTTP使用時のリクエストは通常のASのクラスター化と同様に, 負荷分散装置の配置, Webサーバー・プラグインによる障害検知・リルートといった対処を実施する.

また, SIBWSにおける全てのあて先はクラスター化されたME上に配置することでどのMEに障害が発生した場合においてもサービスの継続を実現する. この場合, スケーラビリティーの同時実現のために, MEのパーティショニングが考えられるが, 構成の自由度の高いインバウンドおよびアウトバウンド・サービスの分離構成を使用する場合は, リプライあて先のパーティショニングは実施するべきでない. リプライあて先も含めたパーティショニングを実施したい場合は, インバウンドおよびアウトバウンド・サービスを同一プロセス上に載せる構成を選択する.

システム・トポロジーとして高可用構成を行うことに加え, より細かい障害対応を求められるケースでは, 本論文で提案しているような形でESBのメディエーション機能を使用する. メディエーションに障害対応を配置することにより, ESBを利用する個々のアプリケーションは耐障害ロジックの実装から開放される[7].

以上の手法を適用することにより, ESB環境における高可用性を効果的に実現することができる. なお, 本論文で述べた設計手法の有効性は, 実装して確認している.

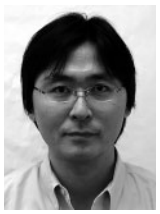
参考文献

[1] David A.Chappell：*Enterprise Service Bus*, O'REILLY, Sebastopol, ISBN0-596-00675-6 (2004).
[2] Martin Keen：*SOA with an Enterprise Service Bus in WebSphere Application Server V6*, IBM, 073849058X (2005).
[3] Ueli Wahi, et al.：*WebSphere Version 6 Web Services Handbook Development and Deployment*, IBM, SG24-6461-00 (2005).
[4] 本 俊也：最新Webサービス, 秀和システム, ISBN4-7980-0706-4 (2004).
[5] Shaun Terry：Enterprise JMS Programming, M&T Books, New York, ISBN0-7645-4897-2 (2002).
[6]“ JAX-RPCによる例外処理, ”http://www.ibm.com/jp/developerworks/webservices/040611/j_ws-tip-jaxrpc.html (2005.8).
[7] Mark Endrei：*Service-Oriented Architecture and Web Services*, IBM, SG24-6303-00 (2004).

**日本アイ・ビー・エム
　システムズ・エンジニアリング株式会社
Webインフラストラクチャー
主任ITスペシャリスト**

丸田 康正 Yasumasa Maruta

［プロフィール］
1997年，ISE入社．Web技術のスペシャリストとしてフレームワーク設計，複数の基幹系Webシステムのデリバリー・リーダーを担当．
現在は，WESBを採用した初の商用SOAシステムのテクニカル・リーダーとしてリファレンス・モデル確立に励んでいる．



**日本アイ・ビー・エム
　システムズ・エンジニアリング株式会社
ワークプレース
主任ITスペシャリスト**

山口 崇 Takashi Yamaguchi

［プロフィール］
1994年，IBM入社．アジア太平洋地域を対象としたWebSphere製品の技術サポート・チームの一員として，主にWebSphere XDなど高トランザクション・高可用性の求められるエンタープライズ・システムのデザインおよび構築を担当．

**日本アイ・ビー・エム
　システムズ・エンジニアリング株式会社
Webインフラストラクチャー
主任ITスペシャリスト**

趙 京揚 Kyohyoh Choh

［プロフィール］
1995年，IBM入社．ISE出向以来，ミッション・クリティカルなWebシステムの設計，構築を担当，主に金融・流通のお客様を担当．
現在は，WESBを採用したSOA基盤のアーキテクトとして，複数のお客様への展開を遂行している．