基幹系システムとSOA環境の双方向インタラクションの実現

― J2EE Connector Architecture（JCA）のInboundとOutboundの連携 ―

服部 洋一　高野 光司

Realization of Bi-directional Interaction between SOA Environment and Enterprise Core System:

- Inbound and Outbound Integration of J2EE Connector Architecture (JCA) -

Yohichi Hattori and Kohji Takano

基幹系システムのSOAによる拡張において，SOA環境から基幹系システムのサービスを利用する方向と，基幹系システムからSOA環境のサービスを利用する方向の双方向のインタラクションが考えられる．本論文では，まず，メッセージの送受信において通信経路の異なる場合の双方向インタラクションを実現するための課題を示す．その上で，J2EE™ Connector Architecture（JCA）仕様の制約の中で，送信と受信の独立したインタラクションを関連付け，非同期に行われる送受信のインタラクションを同期呼び出しに組み込み，一つのWebサービスとしてくくりだすルーティングの技術を提案し，本アプローチの有効性を示す．

In the extension of Enterprise Core System through SOA, two-way interactions are considered: one is the direction from the SOA environment to the Enterprise Core System, and the other is the direction from the Enterprise Core System to the SOA environment. In this paper, restrictions/issues for realizing two-way interactions are first shown, and under the limitations of the J2EE Connector Architecture (JCA) specifications, how the independent interactions of inbound and outbound are co-related, and how asynchronous interaction can be included in a synchronous invocation to wrap the interaction by one web service, are shown, and the effectiveness of this approach is indicated.

Key Words & Phrases：J2EE Connector Architecture（JCA），リソース・アダプター，RER for FSS，Data Systems Environment（DSE），DSE SOA Wrapper，メッセージ駆動ビーン（MDB）
J2EE Connector Architecture (JCA), Resource Adapter, RER for FSS, Data Systems Environment (DSE), DSE SOA Wrapper, Message Driven Bean (MDB)

1. はじめに

企業でSOAを適用するに当たって，これまで構築した膨大なアプリケーション資産をいかに有効に再利用するかは成功への重要な鍵となる．そのため，基幹系システム上に構築したアプリケーション資産をサービスの単位にくくり，SOAのシステム環境から利用可能とするためラッピングの技術が必要不可欠である．ラッピングは，SOAと基幹系システムの接続のための技術と，SOAと基幹系システムのプロトコル変換を行い，サービス・エンドポイントを提供する技術により可能である．ここでSOAと基幹系システムの接続技術の代表的なものには，J2EE Connector Architecture（JCA）［1］と呼ばれるオープン仕様に基づいたリソース・アダプターがある．リソース・アダプターの例としてはIMS（Information Management System）上のアプリケーションとの連携を可能とし，WebSphere® Application Server（WAS）上で稼働するIMS TM Resource Adapter［2］（以前のIMS Connector for Java™）がある．

SOA環境から基幹系システムのサービスを利用するJCAベースのリソース・アダプターでは，JCA Outbound仕様を実装する．SOA環境から基幹系システムへのサービス利用の方向をここではアウトバウンドと呼ぶこととする．さて，SOAによる統合は，何もSOA環境から基幹系システムのサービスを利用するだけには限らない．既存システムである基幹系システム自体も拡張し，SOA環境のアプリケーションと連携を取ることも可能で

提出日：2008年9月9日　再提出日：2009年2月14日

ある．例えば，基幹系システムの証券システムより，指定した銘柄の株価が指定した条件の価格帯に変動した場合に，SOA 環境のアプリケーション・クライアント（例えば携帯端末）に通知メッセージを出力するようなシーンが考えられる．すなわち，基幹系システムから SOA 環境へ非同期メッセージを通知するような場合である．この方向へのサービス利用をここではインバウンドと呼ぶ．あるいは，一歩進めて，SOA 環境のアプリケーションと基幹系システムのアプリケーションが対等に双方向の，すなわち，アウトバウンドとインバウンドのインタラクションを取ることも考えられる．例えば，基幹系システムの銀行勘定系アプリケーション A が他行への振り込みなどの場合に対外接続アプリケーション B とレガシーなインターフェースで連携していたとし，B が老朽化したため，新しいオープンな技術に基づいた SOA 環境のアプリケーション C に移行するといった場合を考える（図 1 に示す）．この場合，基幹系システムのアプリケーション A への影響を抑えるため，基幹系システム側のインターフェースは変更せずに，SOA 環境アプリケーション C と連携するためには，SOAP/HTTP などの標準インターフェースとレガシー・インターフェースとのラッピング機能を作り込むことにより，基幹系システムと SOA 環境のそれぞれのアプリケーションの対等な双方向インタラクションが実現可能である．

著者らは IBM Data Systems Environment（DSE）バンキング・システム上に構築された銀行勘定系の資産を SOA のシステム環境で活用するためのラッパー DSE SOA Wrapper（アセット登録名：SOA message transformation adapter for IBM banking service system）を開発し，SOA 環境と DSE 環境の双方向インタラクションを実現した．これは，金融機関の既存勘定系システム資産を活用しながら SOA によって基幹系システムを拡張するためのソリューション体系 Rapid Enterprise Renovation for Financial Services Systems（RER for FSS，通称 SOA RER）の一コンポーネントに位置付けられる［3］［4］．

本論文では，基幹系システムと SOA 環境の双方向インタラクションを，JCA の技術の拡張により実現する手法を提案する．以下，2 章で前提となる JCA 仕様について述べ，3 章で SOA 環境と基幹系システム／ DSE 環境のインタラクションに関する課題を整理し，4 章でその課題を解決するアーキテクチャーを提案し，5 章でその実装の核となる JCA Inbound-Outbound の連携について詳述し，6 章で実装を評価し，応用案を提示する．

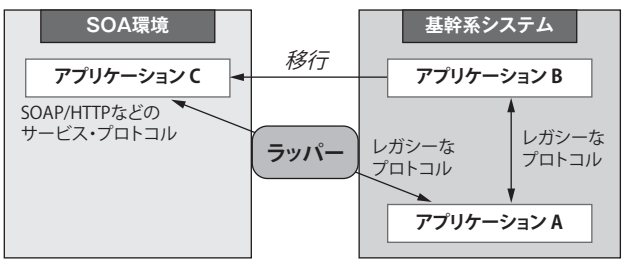

図1. 双方向インタラクション

2. JCA仕様

JCA とは，アプリケーション・サーバーと基幹系システムとを Java の技術で接続するための規約であり，リソース・アダプターがアプリケーション・サーバーと協調して動作し，アプリケーションに使用する通信経路を隠ぺいしたインターフェースを提供する．Java Community Process で Java Specification Request（JSR）16 として定義された仕様が JCA 1.0 であり，リソース・アダプターから基幹系システムへの要求／応答モデル（JCA Outbound）を規定した．JSR 112 として定義された JCA 1.5 では新たに基幹系システムからリソース・アダプターへのメッセージ駆動モデル（JCA Inbound）が追加された．

JCA Outbound と JCA Inbound 仕様の概念図を図 2 に示す．まず，JCA Outbound の場合，Connection（接続）オブジェクトが基幹系システムとの通信路の論理表現であり，Interaction（インタラクション）オブジェクトが基幹系システムのアプリケーションとの対話を実施する．アプリケーションとの対話は Record と呼ばれるメッセージ（電文とも呼ぶ）を介して行われる．この JCA Outbound 仕様に基づくリソース・アダプターにより Java アプリケーションから基幹系システムへの連携が可能となる．なお，リソース・アダプターの JCA Outbound を利用する Java オブジェクトを J2EE Connector（J2C）Bean と呼ぶ．

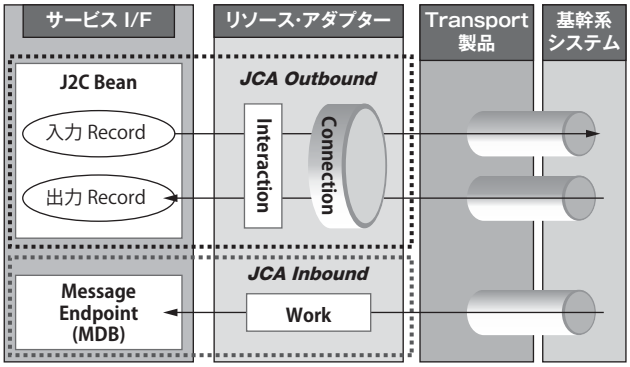

図2. JCA仕様

次に，JCA Inbound では非同期に受信したメッセージをエンドポイントであるメッセージ駆動ビーン（Message Driven Bean:MDB）に配信する仕様を定める．基幹系システムからメッセージを受信したリソース・アダプターは，MDB のインスタンスを作成し，メッセージを配信する．なお，JCA ではリソース・アダプター専用のスレッドである Work およびその制御用マネージャー WorkManager が提供され，基幹系システムからのメッセージの受信，MDB へのメッセージの配信処理などに使用可能である［1］［5］．

なお，両者に共通する基幹系システムとの通信制御は，リソース・アダプターで独自に通信制御プログラムを実装する方法と，基幹系システムとの通信制御に特化した製品（Transport 製品）を使用し，その通信制御プログラムが提供するインターフェースを使って，セッション管理やメッセージの送受信を実装する方法がある．

3. SOA環境と基幹系システムのインタラクションに関する課題

ここでは，SOA 環境と基幹系システムのインタラクションに関する課題を，アウトバウンド／インバウンド双方について示す．

まず，アウトバウンドの場合，JCA Outbound 仕様を実装したリソース・アダプターでは，サービス・インターフェースを介し，基幹系システムでの処理をサービスの形でサービス要求元に公開可能である．SOA 環境のサービス要求元はリソース・アダプターに対する要求を発行し，リソース・アダプターが基幹系システムとのインタラクションを行い，結果をサービス応答としてサービス要求元に返す．ここで，基幹系システムへの送信と受信のチャネルが共用されている場合など，一つのインターフェースで送受信が可能な場合は，リソース・アダプターでサービスとの対応付けも難しくはない．しかしながら，基幹系システムへの送信と受信のチャネルが別で，送信と受信が非同期に行われるような場合に，SOA 環境から要求／応答で呼ばれる同期サービス要求が必要な場合がある．例えば，DSE のシステム間連携と呼ばれる取引形態にこのようなケースがある．この場合，送信に対して受信メッセージをどのように対応付けるかといった困難が伴う（対応付けをコリレーションと呼ぶ）．すなわち，同期プログラミング・モデルに非同期処理を埋め込み，非同期性をサービス呼び出しから遮へいする必要がある．このような場合に対して JCA Outbound では何も規定がなく，リソース・アダプターの開発者に委ねられているのが現状である．この課題を図 3 に示す．

次に，インバウンドの場合について取り上げる．この場合のアプリケーションの作成方法として，主に以下の 2 通りの方法がある．

1. アプリケーションが受信要求を発行し，基幹系システムからのメッセージが到着するまで受信を待ち続けるという方式（RECEIVE-ONLY 方式と呼ぶ）
2. メッセージを受信した場合に，必要なアプリケーションが呼び出される方式（メッセージ駆動方式と呼ぶ）

RECEIVE-ONLY 方式では，アプリケーションは JCA Outbound を使って，同期型のプログラミング・モデルを使用し，最初にリソース・アダプターに対して要求を行う．リソース・アダプターでは，基幹系システムへの電文送信は行わず，基幹系システムから送られる電文の受信のみを行う．電文を受信すると，アプリケーションに返す．前述の IMS TM Resource Adapter は，この方式をサポートしている．一方，メッセージ駆動方式は，JCA Inbound が規定する方式である．RECEIVE-ONLY 方式と異なり，アプリケーションは最初にリソース・アダプターへの要求を行う必要がなく，電文を受信したタイミングでの処理のみを実装すればよい．さて，これらの方式は一長一短であり，両方式のサポートが望まれるが，異なるプログラミング・モデルに対応する必要があり，両方式

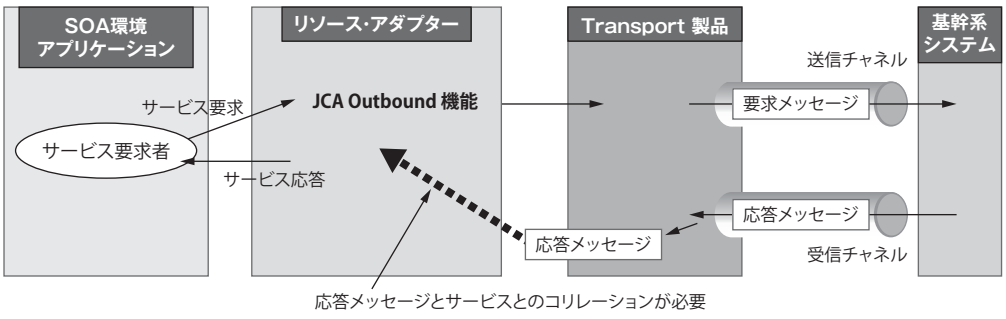

図3. アウトバウンドの課題

ともに対応することは困難である.（なお，最新の IMS V10 では IMS TM Resource Adapter がインバウンドの課題，すなわち，両方式に対応した［2].）

最後に，SOA 環境と DSE 環境間のインタラクションに特化した課題を紹介する．DSE バンキング・システムは IMS 上に構築される．勘定系アプリケーションは，大量のトランザクションを高速に処理することが求められるため，IMS のデータベースの一種であり，メモリーを使用する高速な MSDB（Main Storage Databases）を利用することが多い．しかしながら，IMS TM Resource Adapter では，MSDB の更新がサポートされない．IMS TM Resource Adapter は OTMA（Open Transaction Management Access）インターフェースにより IMS においてトランザクション処理が行われる．この OTMA 自体が MSDB の更新に対応していない．従って，IMS との接続を可能にするほかの製品，例えば IMS SOAP Gateway なども OTMA を使用するため，MSDB 更新ができない．これは SOA 環境と DSE 環境のインタラクションの上での非常に大きな制約である．

4. SOA環境と基幹系システムのインタラクションのためのアーキテクチャー

前述した課題に対して，解決のアプローチを示した上で，実装のアーキテクチャーを示す．

4.1 SOA環境とDSE環境間のインタラクションにおける課題の解決

MSDB の更新を可能とするためには，OTMA を使用しない接続形態を取る必要があるが，この場合は SNA（Systems Network Architecture）/VTAM（Virtual Telecommunications Access Method）を経由して，直接 IMS のコントロール・リージョンとやりとりを行う．この SNA/VTAM を経由して，IMS 上の System Development Aid for IMS/ESA On-Line Applications（SAIL）/Common Application Control Package for Advanced Banking System（CAP-A）アプリケーションとの通信を実現する製品に，金融対外ネットワーク・インターフェース制御サーバー AIX®/6000 版 SAIL/CAP-A 接続フィーチャー（以下，FINEACE/6000）があり，営業店システムなどのチャネル・システムと DSE 環境の接続に使用されてきた実績を持つ．そこで，通信チャネルの制御に FINEACE/6000 を使用することにした．これにより，MSDB 更新も可能である．

4.2 SOA環境と基幹系システム間のインタラクションにおける課題の解決

インバウンドの課題は RECEIVE-ONLY 方式とメッセージ駆動方式という 2 つの異なるプログラミング・モデルに対応するのが困難という課題であった．アウトバウンドの課題は，基幹系システムへの送受信で異なる通信路を通るメッセージ間のコリレーションが困難という課題であった．そこでこのコリレーションに着目し，コリレーションの解決手法として，インバウンドにメッセージ駆動方式を採用した上でインバウンドとアウトバウンドを内部的に結び付ける手法（“Inbound-Outbound 振替方式”）を考えた．これによって，インバウンドのメッセージ駆動方式による非同期メッセージ受信も可能となり，かつ，アウトバウンドへのコリレーションによって，アウトバウンドの課題も解決する．さらに，コリレーションの応用により，RECEIVE-ONLY 方式による受信も実現可能となり，インバウンドの課題にも対応する．

もし，仮にインバウンドに RECEIVE-ONLY 方式を採用し，その応用のみで上記課題すべてに対応しようとした場合，RECEIVE-ONLY 方式によって受信したメッセージの仕分けや，不要メッセージの処理が必要となるが，この仕分けや不要の判断は業務的判断を必要とするため，汎化できない．一方，メッセージ駆動方式で受信すれば，これらの判断を業務処理を行う MDB に委ねることが可能である．従って上記アプローチを取った．採用したアプローチを以下に整理する．

JCA Inbound によるメッセージ駆動方式（インバウンド課題の一部に対応）

→ Inbound-Outbound 振替方式によるコリレーション（アウトバウンド課題に対応）

→ Inbound-Outbound 振替方式の応用による RECEIVE-ONLY 方式（インバウンド課題に対応）

このように，上記アプローチが 3 章で述べた課題を解決し，SOA 環境と基幹系システムの双方向インタラクションを可能とすることが分かる．なお，“Inbound-Outbound 振替方式”の詳細は 5 章で述べる．

4.3 DSE SOA Wrapperの実装アーキテクチャー

機能拡張版の DSE SOA Wrapper では，従来のアウトバウンド機能に加え，DSE 主導の非同期メッセージの単方向インタラクション，また SOA 環境と DSE（SAIL）環境の双方向インタラクションを実現した．図 4 に，これらのアーキテクチャー概要を示す．

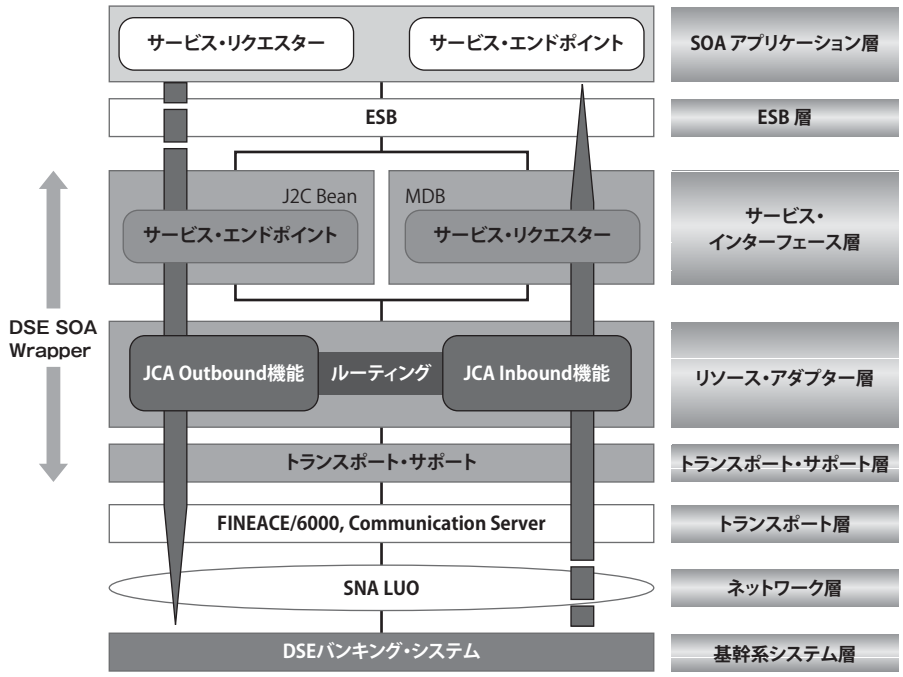

図4. DSE SOA Wrapperのアーキテクチャー

DSE SOA Wrapper は 3 層から構成される．SOA 環境に対するのがサービス・インターフェース層である．これはアウトバウンドに対しては，基幹系システム層の処理をサービスとして公開するサービス・エンドポイントであり，J2C Bean として実装される．インバウンドに対しては，基幹系システムから送られるメッセージに対するエンドポイント（実装は MDB）であり，また SOA 環境のサービスを呼び出すサービス・リクエスターとしても機能する．SOA と基幹系システムのプロトコル変換が行われる層であり，ユーザー実装が必要な部分である．次のリソース・アダプター層は DSE SOA Wrapper の根幹の部分であり，JCA Outbound と JCA Inbound 仕様を実装する．また Inbound-Outbound 振替を行うルーティング機能も実装される．次のトランスポート・サポート層は，通信制御プログラムとやりとりし，セッション管理，メッセージの送受信を行う．トランスポート層の通信制御プログラムは前述した FINEACE/6000 とその前提製品 Communication Server for AIX を使用し，SNA LU0 プロトコルで DSE バンキング・システムに接続する．SOA と基幹系システムの接続をリソース・アダプター層とトランスポート・サポート層で実現する．

5. 双方向インタラクションの実現

ここでは双方向インタラクションを実現する上で，キーとなる機能である Inbound-Outbound 振替機能について述べる．前述したように，これは，JCA Inbound 仕様にはない，著者が独自に考えた機能であり，3 章で述べたアウトバウンドの課題を解決する．DSE の双方向インタラクション（システム間連携）では，DSE への送信と受信では異なるチャネルが使用されるため，課題で指摘した場合に相当するが，すべてのメッセージにコリレーションが必要とは限らない．コリレーションが必要ないものは，DSE より受信したメッセージを JCA Inbound の仕様通りに配信するので，特に問題とはならない．しかしながら，メッセージに含まれる連動処理回数やターミナル・ファイル・キーなどメッセージのキーとして使われる情報の制御のため，コリレーションが必要なケースもある．例えば，

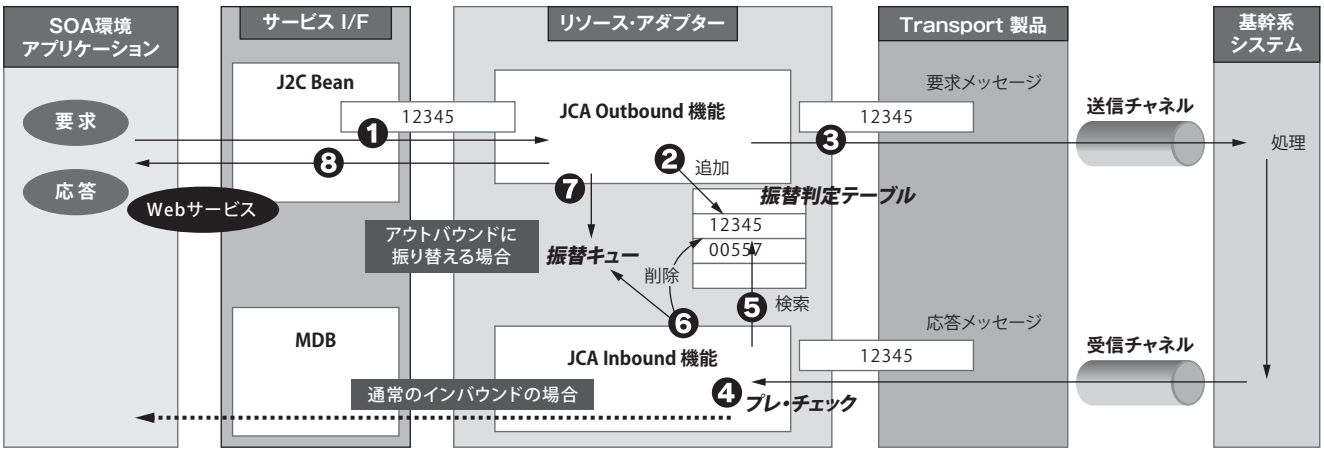

図5. Inbound-Outbound振替

SOA 環境のアプリケーションが DSE（SAIL）に対して発行した連動メッセージに対する SAIL からの連動戻りメッセージの場合である．そこで JCA Outbound と JCA Inbound を内部的に関連付け，JCA Inbound で受信したメッセージを JCA Outbound にルーティングし，振替を行う．図５に JCA Inbound が MDB に配信する通常のケースと，Outbound への振替を実施する場合の概要を示す．

この振替機能の詳細を述べる．以下は図５に記した番号に対応する．この実装手法は，DSE SOA Wrapper に限定したものではなく，基幹系システムとの接続を行うさまざまなリソース･アダプターの実装に応用可能である．

1. サービス・インターフェースはサービス要求を基幹系システムへの電文へ変換した後，Inbound-Outbound が必要な場合にひも付け（コリレーション）用キーを電文内にセットし，リソース・アダプターに渡す．
2. JCA Outbound は，振替判定テーブルにひも付けキーによりエントリーを作成する．
3. JCA Outbound は，電文を基幹系システムへ送信する．
4. JCA Inbound では基幹系システムからの電文を受信後，電文をチェックする（プレ・チェック）．
5. JCA Inbound は電文が振替対象であれば，電文に含まれるキー情報を使って，振替判定テーブルを検索する．
6. 振替判定テーブルに該当エントリーが存在すれば，JCA Inbound は振替キューに電文を Put し，振替判定テーブルからエントリーを削除する（5 ～ 6 は同一同期点内で実施）．
7. JCA Outbound は振替キューから電文を Get し，サービス･インターフェースに渡す．
8. サービス・インターフェースは通常のアウトバウンド処理と同様の方法で電文をサービスの応答に変換して要求元に返す．

振替判定テーブル，振替キューは，リソース・アダプターの用途に応じて最適なものを選択し，実装すればよい．このように JCA Outbound と Inbound をつなぐ機能を追加することにより，リソース・アダプターは送信と受信で独立した通信経路を取る場合のそれぞれの電文を一つのインタラクションとしてくくり，一つの Web サービスとして SOA ワールド内のアプリケーションに提供可能となった．

6. 実装の評価および発展

最後に DSE SOA Wrapper における実装を評価し，実装の発展案について考える．

DSE SOA Wrapper の機能拡張版は 2008 年 10 月に日本 IBM のアセットとして登録された．機能拡張版の実装状況は以下の通りである．

- JCA Outbound 機能 → 実装済み
- JCA Inbound 機能 → 実装済み
- Inbound-Outbound 振替機能 → 実装済み

3 章で示した課題に対する状況は以下の通りである．

- SOA 環境から基幹系システム（アウトバウンド）に対する課題 → Inbound-Outbound 振替機能により解決
- 基幹系システムから SOA 環境（インバウンド）に対する課題 → JCA Inbound 機能により，メッセージ駆動方式には対応．RECEIVE-ONLY 方式は未対応だが，実装の可能性は検討済み．この拡張についてはこの後述べる．

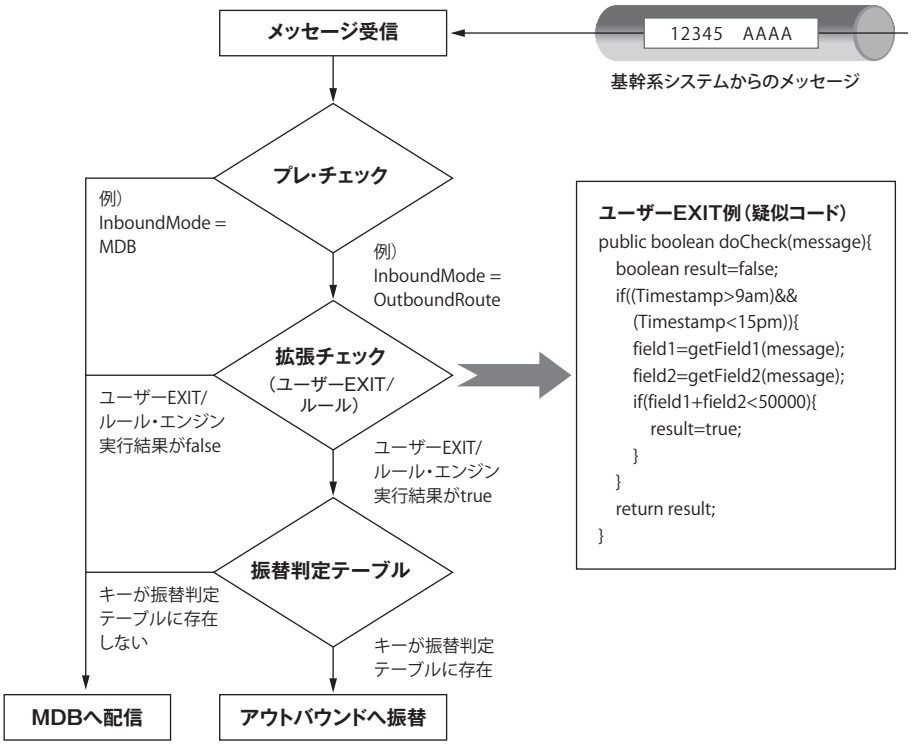

図6. Inbound-Outbound振替のアルゴリズム

• DSE 環境特有の課題（MSDB 更新不可）→ FINEACE/6000 を使用した SNA LU0 接続により解決

以上により，一部を除いて課題は解決し，SOA 環境と基幹系システムの双方向インタラクションに本アプローチが有効であることを示せたと考える.

さて，4.3 で紹介した Inbound-Outbound 振替手法を発展させ，よりフレキシブルな振替のためのアルゴリズムを提案する. 図 6 がその例である.

JCA Inbound 機能を拡張して，基幹系システムからのメッセージを受信後，3 段階のチェックを実施する（このチェック自体の ON/OFF も構成定義で設定可）.

1. プレ・チェック…ここでは，メッセージ内のフィールド値，あるいはリソース・アダプターが保持しているステート，構成定義値などに応じたチェックを行う. チェックはハードコードされる.
2. 拡張チェック…ここでは振替判定のルールをリソース・アダプターの開発者からアプリケーション開発者に解放する. ユーザー EXIT やルール・エンジン（例えば，Drools［6］や ILOG［7］など）にて，追加の判定を行う.
3. 振替判定テーブル…JCA Outbound と JCA Inbound からアクセスされる共有リソースであり，前述のような動的にコリレーションを行う方式以外に，事前にコリレーション用の情報（メッセージ内のどのフィールドをキーに使用するか，また振替を行うキーの値，あるいは範囲など）を設定しておく静的な方式も考えられる.

著者は，この振替判定テーブルの静的方式を応用し，リソース・アダプターの構成時に振替に必要となる情報を登録しておき，基幹系システムから受信したメッセージに対して常に振替を実施することにより，未対応の RECEIVE-ONLY 方式を実現可能と考えている.

7. おわりに

基幹系システムと SOA 環境の双方向インタラクションを実現する上での課題を整理し，テクニカル・イネーブラーである JCA ベースのリソース・アダプターにおいて，JCA 仕様の拡張により課題を解決する実装アプローチを提案した. 当アプローチは，DSE 環境にとどまらず，広く基幹系システムと SOA 環境の双方向インタラクションの実現に応用可能なものである.

謝辞

本論文の執筆に当たっては兼康健氏，山下健司氏より有益なご助言をいただきました. アセット開発にかかわった皆様も含め、この場を借りてあらためて深謝いたします.

参考文献

［1］Sun Microsystems, Inc.：“J2EE Connector Architecture Specification Version1.5,” http://java.sun.com/j2ee/connector/（2009.1.27）.
［2］IBM：“IMS TM Resource Adapter User's Guide and Reference Version 9, Version 10（SC19-1211-03）,” http://publibfp.boulder.ibm.com/epubs/pdf/ico1030a.pdf（2009.1.30）.
［3］日本 IBM：“2006 年 12 月 6 日 SOA による金融機関向け現行基幹系システム活用ソリューションの発表 ,” http://www.ibm.com/jp/finance/solutions/fsn/dec2006_soa.html（2009.1.23）.
［4］山下眞澄：“SOA を活用した Rapid Enterprize Renovation（SOA-RER）のアーキテクチャー ,” ProVISION, No.54, pp.72-79（2007）.
［5］David Currie：“JCA 1.5, Part 2: Work management and transaction inflow,” developerWorks®, http://www.ibm.com/developerworks/java/library/j-jca2/（2008.7.2）.
［6］Drools：http://jboss.org/drools/（2009.2.3）.
［7］ILOG：http://www.ilog.com/（2009.2.3）.

日本アイ・ビー・エム株式会社
グローバル・ビジネス・サービス
金融ソリューション
金融システム開発、金融第一システム開発
コンサルティングITスペシャリスト

服部 洋一 Yohichi Hattori

［プロフィール］
1989 年日本 IBM 入社. 大和研究所にて，通信制御ソフトウェア，Smalltalk によるテレフォン・バンキング・システムなど，主に金融インダストリーのミドルウェア製品開発に従事. 2008 年以降はサービス部門において，本論文で紹介した DSE SOA Wrapper アセットの設計・開発を担当する.

日本アイ・ビー・エム株式会社
グローバル・ビジネス・サービス
金融ソリューション
金融システム開発、金融第一システム開発
アドバイザリーITスペシャリスト

高野 光司 Kohji Takano

［プロフィール］
1998 年日本 IBM 入社. 大和研究所にて，IBM DSE バンキング・システム製品の開発に従事. 2007 年より，本論文で紹介した DSE SOA Wrapper アセットの設計・開発を担当する.