SOAにおけるフロー制御の実装設計手法

星島 洋一　東郷 巖

Implementation Design Techniques for Flow Controls in SOA

Yohichi Hoshijima　Iwao Tohgoh

本論文ではSOA（Service Oriented Architecture）の主目的の1つであるサービスの組み合わせによる新規サービスの迅速・高品質な提供にフォーカスし，その達成に最も重要で有効な役割を果たすフロー制御に着目する．SOAの視点からエンタープライズシステムのレイヤとフローを整理し，各レイヤに存在するフロー制御に求められる機能要件を明確にする．そしてその各機能要件を満たすためのフロー制御の実装設計手法を提案する．

In this paper, we focus on the rapid and high-quality provision of new services resulting from the combination of services. This is one of the main purposes of Service-Oriented Architecture (SOA). We pay particular attention to flow control, which plays the most important and effective role in achieving such provision. We organize the layers and flow controls of an enterprise system from the viewpoint of SOA, and we clarify any functional requirements needed for the flow controls in each layer. We also suggest implementation design techniques for flow controls to satisfy each of the functional requirements.

Key Words & Phrases： レガシー・トランスフォーメーション，メッセージング，フロー制御，SOA
legacy transformation, messaging, flow control, SOA

1．はじめに

SOA（Service Oriented Architecture）に基づいたシステム構築を模索するプロジェクトにおいて，サービスの粒度の議論が盛んであるが本論文では，適切なサービスが切り出された先に話を進める．サービスとはITシステムが提供する単独で機能可能な再利用性のある業務の塊であり，各企業のシステムユーザに対して価値を与える．本論文ではこの塊をシンプルサービスと呼ぶ．俯瞰（ふかん）するとエンタープライズシステムは外からのトリガーを受けて，**図**1のように内部に隠蔽（いんぺい）された複数のシンプルサービスをつないで実行し結果を返すサービスの集合体であると言える．このシンプルサービスをつなぐ機能がフローであり，アジャイルなフローの設計・開発・運用環境を実装したエンタープライズシステムが企業のアジリティを支えることができると言っても過言ではない．本論文での「フロー」の表現はフロー制御を意図する．

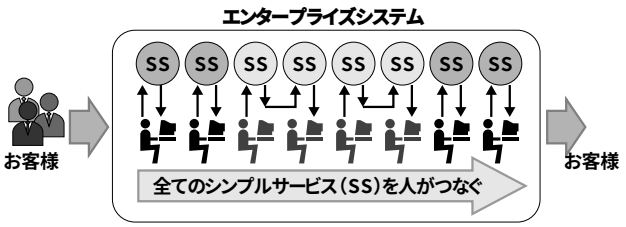

図1.　エンタープライズシステムの俯瞰図

提出日：2005年8月31日　　再提出日：2006年9月15日

本論文ではSOAの視点からIBMが［1］などで提唱している“ SOA Foundation Reference Architecture: Solution view ”を現実実装に射影し，エンタープライズシステムの役割や依存関係などを把握しやすくするためにレイヤ（層）構造で整理する．2章でエンタープライズシステムのレイヤとそこに存在するサービスとフローを整理し，3章で整理したフローの機能要件を明確にした上で，4章で抽出したフローの実装という観点から各機能要件を満たすための設計手法を提案する．最後の5章でフローのまとめと意義について述べる．

2．エンタープライズシステムのレイヤ

エンタープライズシステムを，サービスの要求側から提供側まで役割分担すると，**図2**のようにITシステムの観点からは4層の機能レイヤが存在し，サービスの視点からは5層のサービス関連レイヤが存在する．

本論文ではこのサービス関連レイヤのことを単に「レイヤ」と述べる. 以下に各レイヤとそのフローについて整理する.

図2. エンタープライズシステムのレイヤとフロー

(1) シンプルサービスレイヤ

トランザクション管理／アプリケーションサーバ(以降「TM/AS」と記述) ミドルウェア環境でのトランザクションにより括られた単独のサービスを構成するレイヤである. このレイヤにはソフトウェア部品からなるコンポーネント群であるシンプルサービスが存在する. シンプルサービスの基本はリクエスト/レスポンス(以降「R/R」と記述)型のメッセージ交換パターン[2]となる. ここのシンプルサービス自身はDBアクセス処理とビジネスルール(アルゴリズム)によって構成され, ここにも業務ロジックという末端コードレベルのフローが存在するがこのフローは本論文の対象外とする.

(2) コンポジットサービスレイヤ

シンプルサービスを複数つなぎ別のサービスを提供するレイヤである. 複数つながれたサービスであるコンポジットサービスの基本もR/R型のメッセージ交換パターンとなり, それ自身もサービスとなる. 技術的な切り口で見るとこのレイヤには3種類のフローが存在する. 照会系サービスを複数並列に動かし結果を効率良く集約するための集約フロー, 業務的な整合性を保つために複数サービスの実行リカバリー単位のアトミック性をミドルウェアインフラ技術で守るマイクロフロー, シンプルサービスの実装インフラの違いによりこのアトミック性をコンペンセーション処理機能などの追加機能で補うR/R型のマクロフローである.

説明を補足するとマイクロフローの場合は, TM/ASミドルウェアにて複数のシンプルサービスのアトミック性を保証することができる. 一方マクロフローの場合はトランザクションが別れるために, " A "と" B "によって構成される" C "サービスを実施時に" A "が完了した後で "B" が異常終了というケースに "A" の実行を業務的にロールバックするコンペンセーション処理が必要になる.

コンポジットサービスは, 通常, マイクロフローとマクロフローに分類され, 集約フローはマクロフローの一部として扱われるが以下の技術的な観点で本論文では別の分類とした. IBMの" SOA Foundation Reference Architecture: Middleware Services view "[3]では, マクロフローはフローエンジンであるWPS(WebSphere® Process Server)製品などを利用して実装される「プロセス・サービス領域(BPELエンジンなどでのフロー制御)」にあり, 上記に説明した集約フローは「ESB (Enterprise Service Bus)領域」に位置付けられたWMB (WebSphere Message Broker)製品などの標準機能として提供されるからである. このリファレンスアーキテクチャーと現実的な実装を考慮し別の分類とした.

(3) イベント通信レイヤ

イベントベースでサービスを実行させるためのイベントメッセージが通過するレイヤである. ここでのイベントとは, CBE (Common Base Event)[4]として定義された運用・監視系を対象としたものではなく, 派生として業務を起動する一方向のメッセージ交換パターン型のものである. このレイヤをシンプルサービスあるいはコンポジットサービスの実行を依頼するためのイベントメッセージが一方向で通過し, このメッセージによって順次実行されるフローをここではイベントフローと呼ぶ. エンタープライズシステム内で実行される処理モデルはパブ・サブモデルやコールバックモデルなど, 多数のモデルがあるが, R/Rモデル(双方向モデル)と一方向モデルの2つのメッセージ交換パターンの組み合わせに集約することができる. サービスは, シンプルサービスもコンポジットサービスもR/R型でありサービスを集中して制御するフローが存在する. 一方, 同一エンタープライズシステム内でも制御が異なるサブシステムとの通信はイベントベースでのプロセスの受け渡しを行う**図3**のようなイベント駆動型のデザイン(Event Driven Architecture [5])がより現実的であり, この機能を提供するイベント通信レイヤが必要である. 下図では, 便宜上イベントとイベントの間を直接接続した表現をしているが現実にはこの間にイベント・サーバやイベント・ブローカーなどが介在することもある.

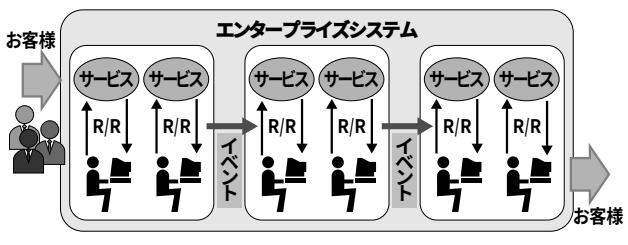

図3. イベント駆動型のサービスの連携

(4) チャネルサーバレイヤ

サービスを要求する末端のエンドユーザインタフェースとなる面をチャネルと表現する. チャネルの具体例として, Webブラウザ, ATM（現金自動預け払い機）, プリンターなどがある. このレイヤはチャネルに依存した機能を提供しサービスを呼び出すレイヤである. このレイヤにはチャネルに依存した入出力制御とサービス呼び出し制御を行うチャネルサーバでのフローが存在する. 前者にはサービスの入力パラメータをユーザとの複数回のインタラクションで収集するためのフローや, サービス呼び出しの結果に応じて出力画面を遷移するフロー機能が相当する. 後者には検索サービス呼び出しの結果で該当者が1名だった場合には1名のリスト画面を出すのではなく詳細検索サービスを自動的に呼び出しその該当者の詳細情報画面を出すといったチャネル側の使い勝手に依存したサービス呼び出しのフロー機能が相当する. よって, サービスを提供するレイヤではない.

(5) チャネルレイヤ

WebブラウザやATMなどのチャネル機能を提供するレイヤである. ここは人がエンタープライズシステムのユーザインタフェースを介して得た出力を基に次の処理を行うというヒューマンフローが存在しサービスは存在しない. シンプルサービスはINとOUTのインタフェースが明確に定義されたものなので, エンタープライズシステムが提供している機能が全てサービス化された場合, 究極の姿として全ての処理（プログラム）をサービスでつなぐことが可能なはずである. しかし, 現実には人手が入らなければならない作業が存在する.

以下にこのレイヤに存在する2つのフローを整理する.

①. スキルフロー

その作業を行うために, または次の作業を進めるために人の能力や判断といった作業が必要になるものでこれに対応するフローをスキルフローと呼ぶ. コールセンターでのオペレータ応対や, 新規顧客の審査処理における与信限度額の判断処理などがこれにあたる.

②. 複雑例外処理フロー

自動化が非現実的な複雑な例外処理への対応処理で, これに対応するフローを複雑例外処理フローと呼ぶ. シンプルサービスがつながったフローは全体として業務的な整合性を確保しなければならないが, サービス提供レイヤが異機種に分散しているエンタープライズシステムにおいてはこの整合性確保をミドルウェアのトランザクション管理機能だけに依存することはできない. 結果として障害ケースごとに既述のコンペンセーションロジックを設定することになる. しかし全ての障害ケースを自動的なコンペンセーションで行えると考えることは[6]でも述べられているように現実的ではない.

以上をレイヤごとのフローという観点で整理すると**表1**のようになる.

表1. サービス関連レイヤとフローの整理

サービス関連レイヤ	フローの種類		サービス
シンプルサービス	最小粒度のサービスを構成する業務ロジックがある（論文の対象外）		単独
コンポジットサービス	集約フロー	(1)	複合
	マイクロフロー	(2)	複合
	マクロフロー	(3)	複合
イベント通信	イベントフロー	(4)	no
チャネサーバ	チャネルサーバフロー	(5)	no
チャネル	スキルフロー	(6)	no
	複雑例外フロー	(7)	no

3. 各フローの機能要件

この章ではエンタープライズシステム内で整理した7種類のフローの機能要件を明確にする.

(1) 集約フロー

複数の照会系のサービスを並列に効率良く呼び出し結果を集約し1つのサービス応答として結果を返す.

(2) マイクロフロー

ミドルウェアインフラのトランザクション制御機能により複数サービスのリカバリースコープを1つにまとめる.

(3) マクロフロー

異機種環境に分散したサービスの呼び出しを制御し, 後続のサービスが障害になった場合には既に先にコミットされた更新系処理をアプリケーション的に戻すためにコンペンセーション処理を行う.

(4) イベントフロー

イベントメッセージによって起動されるプロセスは非同期に実行されるため, その実行を保証しなければならない. 送信側は受信側の状態に依存せずに次の処理に進めるようにする.

(5) チャネルサーバフロー

チャネルとの複数回の入力インタラクションを処理し, サービスを複数回呼ぶことができる. サービスの結果に応じて出力画面を遷移できる. フロー処理内においてセッション情報を持ちチャネルと対面しているユーザに対してステートフルに振る舞うことができる.

(6) スキルフロー

人の振る舞いであるスキルフローに対してシステムが支援できることは, 適切なスキルを持った人が適切なスキルフロー作業に配置されているかを監視する仕組みを提供することである. 監視の仕組みで提供される情報を基に管理者は人の再配置や作業生産性を向上するための指導を行うことができる.

(7) 複雑例外フロー

サブシステム“ A ”で実行されたサービス“ X ”とサブシステム“ B ”で障害を起こしたサービス“ Y ”が業務的に関連することを示す情報が統合されたログに適切に出力される. システム管理者はそのログを見ることにより個別サブシステム担当者に連絡を仰ぎ, 業務としての整合性をとる作業を指示することができる.

4. 各フローモデルの実装

この章では前章で抽出された要件を実装するための要素技術を提示するとともに過去に経験したプロジェクトでの具体案を示し, より現実的な設計方法を提案する.

(1) 集約フロー

複数の照会系のサービスを並列に効率良く呼び出し, 結果を集約する具体的な実装は2つ考えられる. 図2のESB層のWMBなどが持つアグリゲーション機能を使う方法と, ビジネスアプリケーション層のBPM（ Business Process Management）製品が持つWSBPEL（Web Services Business Process Execution Language）[7]の実装機能を使う方法である.

WSBPELにおいては**図4**のように2つの照会サービスを<flow>アクティビティ内に記述し並列に処理を実行するモデルを例示する.

```
<flow>
     <!--Invoke Inquire Image A web service -->
      <invoke partnerLink="imagesystem"
              portType="ins:FrameImageSystemPT"
              operation="ImageInquireOperation"
              inputVariable="InqImageRequest"
              outputVariable="InqImageResponse" />

     <!-- Invoke Inquire ASXXX B web service -->
       <invoke partnerLink="asxxxsystem"
              portType="ins:FrameAsxxxSystemPT"
              operation="CustomerInquireOperation"
              inputVariable="InqCustomerRequest"
              outputVariable="InqCustomerResponse" />
```

図4. WSBPELでの並列処理記述の例

BPM製品については製品選択も含め検討中である場合にはESB層にWMBなどを配置しそのアグリゲーション機能による実装も可能である. 本論文では, 既に2章で述べた照会系サービスに限定させた集約フローにおいてはLight Weightな実装の前者を選択した. 実サービス提供層ではないESB層には業務色を持たせない設計方針を守るために**図5**のようなメッセージ形式を規定したSOAPメッセージを生成する集約サービスをWSDL（Web Services Description Language）として定義しWMBなどでそれらの複数オペレーションを目的のアプリケーションサーバに分解して応答を集約する単純機能で汎用的に実装することができる. この対応によりESB層の保守性の向上が図られる.

```
<Body>
  <ｵﾍﾟﾚｰｼｮﾝ名 xmlns=“ http://www.xxxx.co.jp/AG ”> ⇒ 複合サービス部
    <ｵﾍﾟﾚｰｼｮﾝ名1 xmlns=“ http://www.xxxx.co.jp/SD ”>input_1
    </ｵﾍﾟﾚｰｼｮﾝ名1>  ⇒  単一サービス(1)部
    <ｵﾍﾟﾚｰｼｮﾝ名2 xmlns=“ http://www.xxxx.co.jp/SI ”>input_2
    </ｵﾍﾟﾚｰｼｮﾝ名2>  ⇒  単一サービス(2)部
  </ｵﾍﾟﾚｰｼｮﾝ名>
</Body>
```

図5. 集約サービスのSOAPメッセージ構成の例

(2) マイクロフロー

現時点において筐体をまたがった複数サービスのリカバリースコープをミドルウェアインフラのトランザクション制御機能により守っているシステムは日本ではほとんど実績がない. しかし要素技術であるWS-Transaction（WS-CoordinationとWS-AtomicTransaction）はTM/ASミドルウェアであるCICS® Transaction Server V3.1やWebSphere Application Server V6に実装されてきており異機種に分散された複数サービスをこれらの仕様に基づき保てるようになってきている. 筐体をまたがってトランザクションスコープを広げた場合の最大の考慮点はロック時間が長くなることによるシステム全体のパフォーマンスダウンと2フェーズコミット処理障害時のIndoubt（未確定トランザクション）ステータス解消の運用が挙げられる. 前者のロック制御が把握できるクローズされた分散システム間ではWS-Transactionの技術によりマイクロフローの制御範囲を広げるべきと考えている. ほとんどのサービスが特定のTM/ASミドルウェア環境（例えばWAS, CICS）に存在しているような場合は“ 複数の更新サービスを束ねるマイクロフローは対象のTM/ASミドルウェア上に持つ ”という制約を設ける設計ができる.

一方で異機種にまたがり同一リカバリースコープでDBを更新するという業務要件が存在するケースもある. その解決案として以下の2案がある.

案1）パーシステントなメッセージを派生させて非同期に別サブシステムで更新処理を行う（**図6**）

案2）分散DBの機能を使い片側のプラットフォームにDBを配置し共有させる（**図7**）

この選択はデータをサービスに隠蔽するべきという考え方と, 一方で現実を考慮した実装をするべきという考え方の双方をアーキテクチャー, パフォーマンス, 可用性, 運用, 開発という側面で十分に比較し

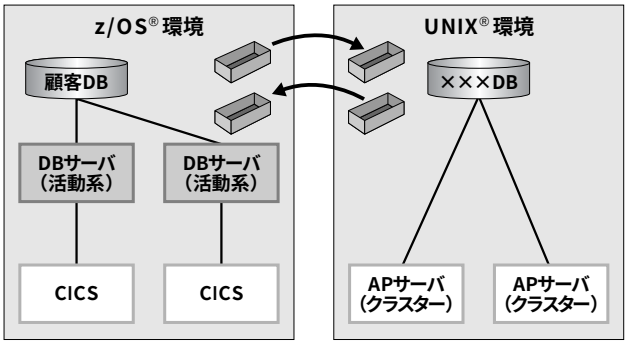

図6.　メッセージング利用の構成

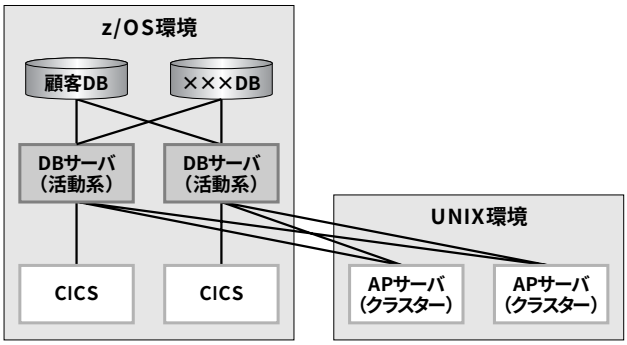

図7.　DBの共有配置の構成

検討すべき項目である.

(3) マクロフロー

BPM製品の機能としてコンペンセーション機能が提供されている. WPSにおいては「Undo」オペレーションが1つのアクティビティとして定義され, ユーザプロセス実行時にハンドリングされなかったFault（例外）が発生した時に当該「Undo」オペレーションを実行することができる.「Undo」オペレーションは正常時処理順序の逆での処理となる.「Undo」の内容と処理のタイミングに依存して正常処理は成功するがコンペンセーション処理は失敗するケースが想定される. 例えば先行入金処理により即座にペンディングの引き落としが行われ, その後先行の入金をコンペンセーションで戻そうとしても既に入金額がないというケースである. こういう場合は人手による運用に頼らざるを得ない. このように考えるとBPMによるコンペンセーションは対応が明白なサービスフローのシナリオに絞るべきであり, 障害シナリオにより派生するコンペンセーションシナリオが多数必要になる場合には, 障害時の処理を打ち切りヒューマンフローである複雑例外フローに任せることが望まれる.

(4) イベントフロー

非同期の一方向の要件を実装するミドルウェアはMOM（Message Oriented Middleware）製品が主役となる. 過去MOM製品によって構築されているシステムは多くある. その中でイベント型メッセージ伝播というソリューションは多数の企業で実装されてきた. 多くの場合は単純にデータの更新結果を伝播するという考えに基づいて設計が行われてきているが, SOAをベースにした新システムでの設計のポイントはプロセスを伝播することである. データ伝播という概念に基づいた場合はそのデータを処理するために送受信の両側に同じような業務のデータ処理が存在することになり, サービスの再利用性というSOAの視点からずれる可能性が高い. SOAPメッセージに包まれたプロセスコンテキストを伝播するという発想に立つと, **図8**のように受信側リスナーから起動されたファサードインスタンスがサービスクライアント（要求側）となりイベント駆動とサービス指向の橋渡しの役割を担うという設計ができる.

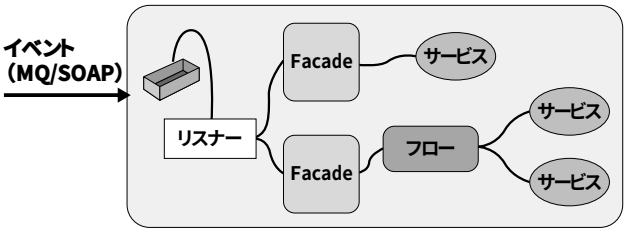

図8.　イベント駆動でのサービス呼び出し構成

ファサードの機能としては, R/R型のサービス要求を受け取るだけでなく結果に応じて次のフロー遷移を制御する機能が必要になる. BPM製品が当該サブシステムに導入されている場合はBPM製品がこの機能部分を実装可能である.

経験したプロジェクトにおいてはCICS内にSOA用R/R対応のファサードとイベント用のファサードを用意した. またイベントメッセージの属性はパーシステントになるためノンパーシステントメッセージで通信されるR/Rモデルとは障害時の運用が異なる. 従ってR/Rメッセージが通るパスをフロントエンドハブ（ESB層）, イベントメッセージが通るパスをフローハブ（プロセスサービス領域[3]となるサービス層）と名付け論理的にも物理的にも異なる要素としてアーキテクチャー上定義した.

(5) チャネルサーバフロー

チャネルサーバは対応チャネルに依存して種々のタイプのものが想定される. Webサーバにおける主な処理フローは以下のようである.

① 入力条件（Submitボタンなど）に応じて特定処理フローが実行される
② フローでは, 画面の入出力項目, サービスの入出力項目, セッション情報項目間でデータの移動を行う

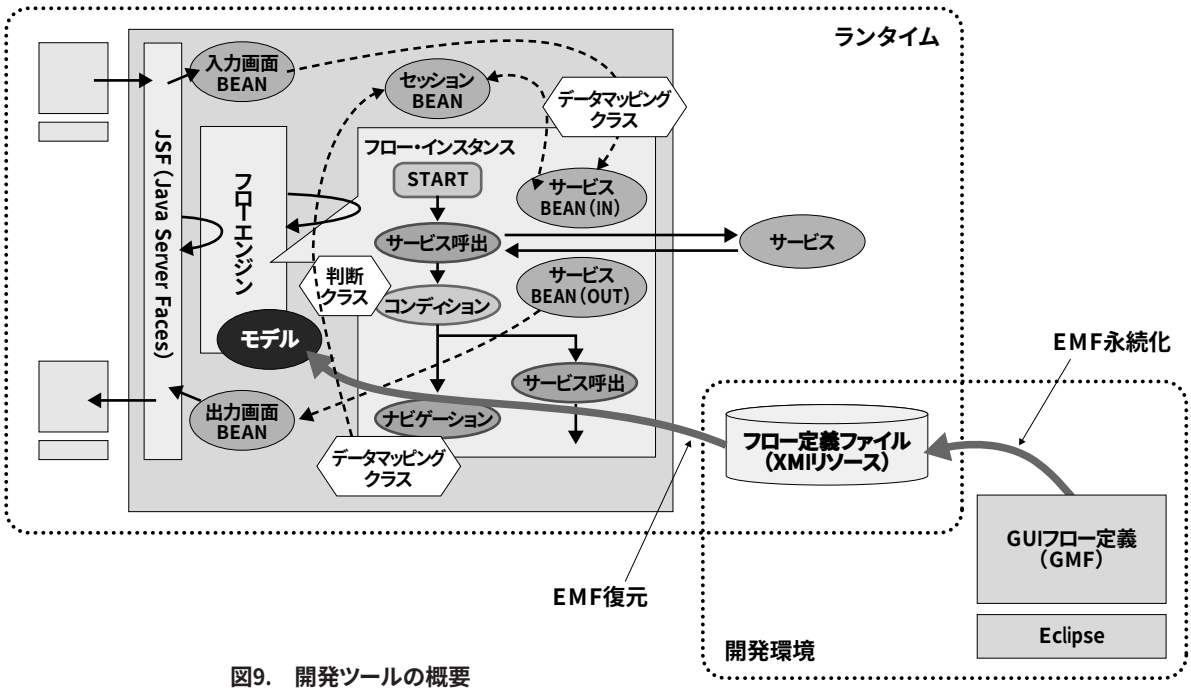

図9. 開発ツールの概要

③ サービス(複数可)を呼び出す
④ 処理結果(例外を含む)に応じて出力画面遷移を変える

①はJSF(Java™ Server Faces)フレームワークの機能により実装される. 残る3つの機能についてモデル化によるツール実装が可能であるがここでは経験したプロジェクトにおける実例を**図9**に示す.

フローをEclipse/GEF (Graphical Editing Framework) 開発環境でモデル化しEMF(Eclipse Modeling Framework)によりXMI(XML Metadata Interchange)リソースとしてフロー定義ファイルに永続化する. ランタイム環境においてこのXMIリソースを読み込み, モデルオブジェクトを復元し, そのモデル情報に従ってフローエンジンは決められたフローを実行する. ②の実装のためにフローエンジンから呼び出されたユーザJavaクラス(“ データマッピングクラス ”)はフローコンテキスト経由で画面ビーン, サービスビーン, セッションビーンをアクセス可能である. ③についてはフローエンジンはWSDLから自動生成されたスタブコードを呼び出し, ④については“ 判断クラス ”の戻りのtrue/falseによってフローエンジンは次のアクティビティを決定する.

SOA対応により再利用性の高いサービスが抽出できるとこれらサービスの保守頻度は低くなる. 一方ビジネスの変革スピードは従来にも増して加速するが, それはチャネルサーバ側のフローにより吸収できる可能性が高い. このチャネルサーバ側のフロー開発機能をEclipse/GEF/EMF上に持つことにより, 整合性のとれた成果物を迅速に開発でき, ビジネス変革を強力に推進できると考えている.

(6) スキルフロー

統一されたXML形式のログフォーマットがCBE (Common Base Event)としてOASISに提出されており, [8]にもあるようにWESB (WebSphere Enterprise Service Bus), WPS V6においてはCBEフォーマットでのログ書き出しとイベントサーバとなる機能も提供されている. ビジネスプロセスの適切なポイントでこのCBEメッセージを非同期に書き出し, イベントサーバに集めることによりビジネスモニタリングが可能となる. このイベント管理のためのインフラ機能部分をCEI (Common Event Infrastructure)と呼ぶが新ITシステムの設計おいて検討しておくべき基盤機能である.

(7) 複雑例外フロー

上記で記述したCBEはログフォーマットとしても使用される. WESB, WPSやCBE対応したアプリケーションのログは直接, CBE対応していないログは汎用(はんよう)ログアダプターを経由してログトレースアナライザーに送ることにより筐体(きょうたい)をまたがる複数サブシステムのログを統合ログとして表示することが可能となる. SOA対応されたシステムにおいては, 一連の業務処理は仮想化されて配置されたサービスを組み合わせて開発される. 従って従来のように特定サブシステムのログファイルによって問題を判別し障害に対応する運用処理では迅速性に欠け不十分である. 統合ログ環境においては関連するアプリケーション, ミドルウェアなどが全て同一相関IDを持ちまわりその相関IDによって関連ログを抽出・表示できる必要もある. 現時点では製品によって未対応の部分もあるがAutonomic対応製品の浸透に合わせてより充実した

統合ログ環境が期待できる.

5. おわりに

以上筆者らが経験してきたシステムの具体例を含めて述べてきたが以下にSOA対応したシステム構築に重要な役割を果たすフロー実装の要点を述べる.

1. サービスという切り口で既存基幹システムを論理的に5つにレイヤ化する(表1)
2. システム計画から判断してレイヤに必要となる7つのフロー機能の優先順位付けを行う
3. 短中期に必要となるフロー機能についてはそのフローの開発・実装も含めて4章を参考にして具体的な製品あるいは開発方法を検討する
4. 長期に必要となる機能についてシステム基盤として拡張性を考慮して設計を行う

SOAの主目的であるサービスの組み合わせによる新規サービスの迅速・高品質な提供には, フロー制御の技術が最も重要な役割を果たす. その意味でも今回のフローモデルの切り出しとその実装手法を効果的に利用することがアジャイルに変ぼうした新システムへの成功要因となる. このためにはエンタープライズシステム全体を俯瞰しレイヤ化し必要なフローを明確化し最終システムのビジョンをITシステムの視点から描いて各機能レイヤを実装しておかなければ達成しえない. SOAで紹介される技術以外では, [5]でも紹介されているイベント駆動型モデルも採用したグランドデザインが必要である. そしてこれらの作業はサービスの粒度の議論と並行してしっかりと行う必要がありその巧拙によって新エンタープライズシステムのアジリティが決すると言っても過言ではない.

参考文献

[1] John Ganci : "Patterns: SOA Foundation Service Creation Scenario," IBM redbooks, http://www.redbooks.ibm.com/redbooks/pdfs/sg247240.pdf P30 (2006.9).
[2] Christopher Ferris : "Basic B2B Profileを理解する," IBM developerWorks, http://www.ibm.com/jp/developerworks/webservices/050422/j_ws-b2bpaper.html (2005.4.4).
[3] John Ganci : "Patterns: SOA Foundation Service Creation Scenario," IBM redbooks, http://www.redbooks.ibm.com/redbooks/pdfs/sg247240.pdf P31 (2006.9).
[4] John Dinger : "Common Event Infrastructureによるイベントの統合管理," IBM developerWorks, http://www.ibm.com/jp/developerworks/autonomic/041119/j_ac-cei.html (2004.6.20).
[5] Jean-Louis Marechanx : "ESBを使用したSOAとEDAの統合," IBM developerWorks, http://www.ibm.com/jp/developerworks/webservices/060412/j_ws-soa-eda-esb.shtml (2006.3.28).
[6] Dirk Krafzig, Karl Banke, and Dirk Slama : *Enterprise SOA*, PrenticeHall, ISBN 0-13-146575 -9 (2004.11.9).
[7] OASIS : WSBPEL, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (2005.11).
[8] 近藤 仁, 佐藤なみえ : "ESBの実態を探る," ISE TECHNICAL CONFERENCE 2005, http://www.ibm.com/jp/software/websphere/developer/websphere/ise/techcon2005/pdf/satokondo.pdf (2005.7).

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
アーキテクチャー・イノベーティブ・ソリューション
エンタープライズ・デザイン部
ICP-コンサルティング ITスペシャリスト

星島 洋一 Yohichi Hoshijima

[プロフィール]
フィールド部門での都銀の第三次オンラインシステムの構築，開発部門での金融ミドルウェア・パッケージの開発，本社部門での製品サポート（IMS，CICS，MQ，WMB）を経験し，現在は多くのプロジェクトにITSとして参画している．

HOSSY@jp.ibm.com

日本アイ・ビー・エム株式会社
テクニカル・セールス・サポート
ディスティングイッシュト・エンジニア（技術理事）

東郷 巖 Iwao Tohgoh

[プロフィール]
CICS，MQ，Webアプリケーション・サーバー関連のミドルウェアを中心に製品サポートを実施すると共に，多くの実プロジェクトのシステム設計からサービスインまでの実装を支援している．

MQOSHO@jp.ibm.com