HITACHI
Inspire the Next

Cosminexus
コズミネクサス
Version 7

SOA解説

既存システムからSOAへのアプローチ

2006.7
July

Contents

uCosminexus Service Platform、uCosminexus Service Architect、uCosminexus Application Server は、経済産業省が 2003 年度から 3 年間実施した「ビジネスグリッドコンピューティングプロジェクト」の技術開発の成果を含みます。

1. はじめに

最近、「情報システムがビジネスの変化に対応できない」という声を良く聞きます。

この背景には、次の2点があるのではないでしょうか。

(1)ビジネス環境の変化が激しくなった

グローバル化、規制緩和、合併や分社化等によってビジネスをとりまく環境の変化は、以前に比べて明らかに激しくなり、企業間の競争も熾烈になってきました。これに伴って、業務プロセス改革や、法令・内規の変更、システムの統廃合、のように情報システムの変更を必要とする機会も非常に多くなっています。

(2)ビジネスとITの関係が密になった

今やITは企業活動のいたるところに入り込んでいます。情報システムの出来・不出来が株主や顧客からの評価に直結する場合もありますし、ラインの生産性や物流、対外取引といった企業活動のコア部分のパフォーマンスを左右する場合もあります。さらに、経営層の意思決定のスピードや確かさにも、ITは深く関わっています。

このような背景から、情報システムをビジネス環境の変化に即して柔軟に対応させていくことが求められる一方で、多くの情報システムには、各部門毎のニーズにしたがって個別最適に形成されてきた経緯があります。その結果、孤立しているシステムがあったり、個別エンハンスの繰返しによって処理が複雑に絡み合ってしまったりしています。このことが、システムを硬直化させ、変化への追従を困難にしていると考えられます。

このような状況を打破する技術として注目されるのが、SOAです。SOAは、サービスの組合せでシステムを構築する技術で、これを上手に利用すれば、硬直化したシステムを、変化に強い柔軟なシステムへと生まれ変わらせることができます。

本冊子では、日立のSOAの考え方に沿って、既存システムを有効に活用したSOA適用のアプローチをご紹介します。

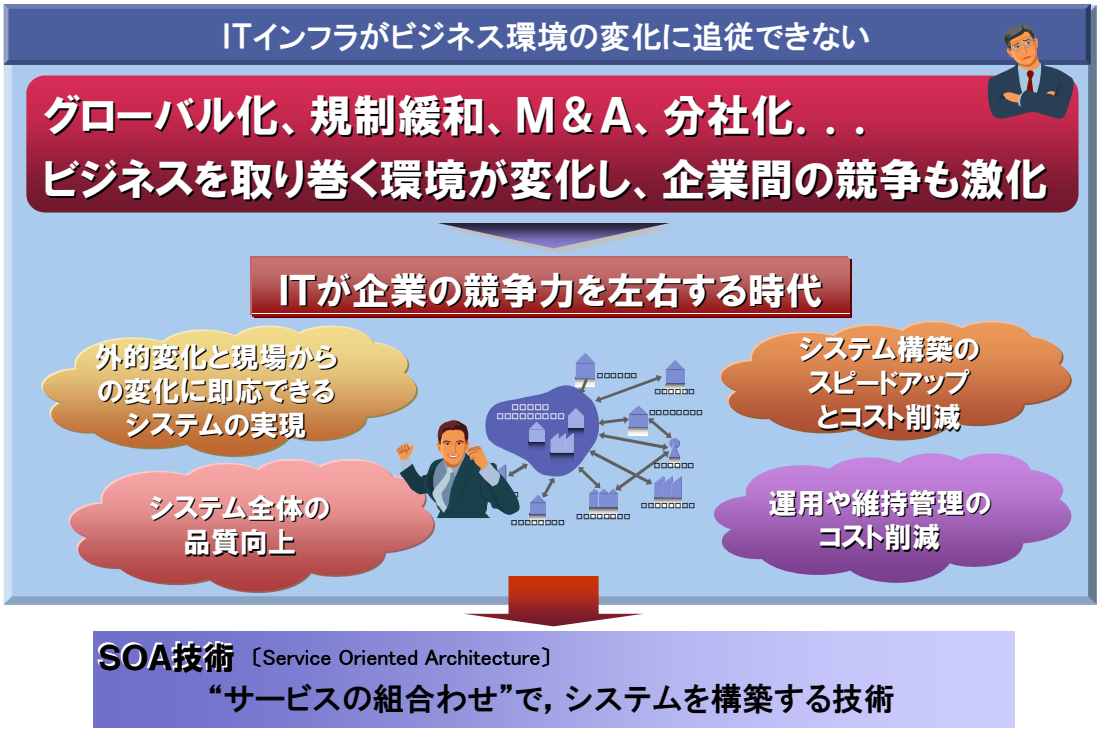

図1－1　SOAへの期待

2. SOAの目指すところ

では、SOAはどのような形で問題を解決することができるのでしょうか。日立では、SOAの目的には、大きく、次の3点があると考えています。

(1)変化への追従を容易に迅速にする

SOAの適用により、サービスという単位で変化を局所化することができます。これによってシステム変更の際の開発部位が限定され、全体への影響も考慮する必要がなくなるため、迅速な変更が行なえるようになるのです。

また、既存のサービスを自由に組合せて新たな開発が行なえるようになることで、システム開発の生産性も向上させることができます。特に、SOAの普及に伴って、ASPとして提供される外部サービスが増えてくれば、必要なときにはすぐにそれを利用し、新たな業務に対応できるようになります。

さらに生産性の面では、コンポーネント指向を上回るSOAのメリットを享受できます。それは、コンポーネントとして開発されたプログラムは、それを利用する各システムに配布する必要があるのに対して、サービスは利用者がネットワーク経由で使いに来るため、配布の必要がないということです。このため、コンポーネントを使用する際の開発環境と実行環境の違いを意識したり、不具合の修正版を再配布したりという煩わしさが無くなり、簡単に部品を組み替え様々な業務処理を実現できるようになるのです。

(2)業務フローを可視化する

迅速な変化への追従のためには、変化を敏感に捉えることも重要です。しかし、従来のシステムでは、業務フローに沿って、顧客や商品毎の処理件数の統計を取ってその推移を調べたり、案件の仕掛り時間やネックとなる部分を調べて業務改善に役立てたりということが、容易には出来ませんでした。これは、業務フローとITを関連付ける標準的な枠組みが無かったことが原因と考えられます。また、業務フローが人手で処理されているため統計情報を得るには再度データ入力が必要だったり、必要な情報が色々な場所に散らばっていて集めるのが大変だったりという現実もあります。

これに対してSOAでは、業務の流れをビジネスプロセスとして自動化してITと関連付け、さらにそれを1箇所で集中管理できますので、その履歴から、案件毎の業務情報を取り出すことが容易になるのです。これを分析して、販売件数やキャッシュフローなどビジネス活動に直結する様々な情報を抽出し、その動きを監視することをBAMと呼びます。

このように、業務フローを可視化しBAMを実現することが、SOA適用の一つの目的となります。

(3)システムの全体最適化を図る

情報システムは、利用者のニーズに合せてサブシステム毎に個別最適で形成される傾向があります。この場合の問題点として、次のようなことが挙げられます。

- 至るところに類似の処理があり、二重開発による開発・保守のオーバヘッドが大きい
- システム・リソースの配分に無駄や偏りがある
- オペレータが別々のシステムに同じデータを何回も入力しなければならず操作性が悪い

これらの問題は、システムを全体最適化の視点から見直さないと、なかなか解消することは出来ません。SOAは、全体最適化を実現し易い環境作りに効果を発揮します。

まず、SOAのサービスコンポーネントは再利用が容易なので、一度作ったものは、それを必要とするすべての業務から使用可能で、二重開発を抑止することが出来ます。また、サービスコンポーネントの稼動場所も自由に選択できるため、システム・リソースを有効に活用することが出来ます。そして、サービス間の連携も容易ですから、二重入力などの不便さも解消できるのです。

次に、これらを実現するSOAのシステムモデルについて、少し詳しく見て行きましょう。

3. SOAのシステムモデル

SOAは、システムを構築する上での優れた考え方ですが、真に効果を発揮するためには、個々のシステムの特性に応じた柔軟な適用が求められます。このため、まずSOAの考え方を良く理解しておく必要があります。そこで本章では、SOAの基本的な考え方を整理します。

3.1 概念の整理

考え方を整理するためには、まず概念の整理が必要です。SOAでよく使用される概念を、表3－1にまとめました。

表3－1 業務処理に関する概念の整理

観点	処理の単位	業務の流れ	指示の伝達
人間系の処理	業務ステップ	業務フロー	帳票または会話
SOAのモデル	サービス	ビジネスプロセス(BP)	メッセージ
ソフトウェアの実装	サービスコンポーネント（AP）	BPEL スクリプト／BP実行基盤	メッセージ／メッセージング基盤(ESB)

ここでは、人間が行なう業務と、それに対応するSOAのモデル、およびソフトウェアの実装の関係を示しています。

まず一連の業務処理を構成する処理の単位を、ここでは「業務ステップ」と呼びます。これに対応するモデルが「サービス」です。またその実装がサービスコンポーネントということになります。サービスコンポーネントは処理（AP）＋データ（DB）で構成されるのが一般的です。

次に、幾つかの業務ステップを組合せて行なうことによって、一つの業務が完結します。この時の業務の流れを「業務フロー」と呼びます。これに対応するモデルが「ビジネスプロセス」で、その実装がBPELで記述されたスクリプト、及びその実行を制御するBP実行基盤です。

そして、人間系の処理においては業務フローを実現するために、先行の業務ステップを実行する人と後続の業務ステップを実行する人の間で帳票（メモを含む）または口頭（会話）による指示の伝達が必要となります。SOAのモデルではこれを、サービス間のメッセージ通信によって実現します。またその実装は、サービス毎に決められたフォーマットのメッセージと、それを中継するメッセージング基盤ということになります。一般にSOAでは、このメッセージング基盤のことを、エンタープライズ・サービス・バス（ESB）と呼びます。

3.2 サービスの考え方

SOAを適用したシステムでは、開発や運用がすべてサービス単位で考えられるようになるため、まずサービスとは何かを良く理解することが重要です。ここでは、どのようなものをサービスと呼ぶか、そしてSOAのモデルにおけるサービスはどのような構造と性質を持っているかについて説明します。

（1）サービスの大きさ（粒度）

前節でご説明したように、サービスの実装は、一つの業務ステップに対応した処理を実行するサービスコンポーネントに相当します。しかし、その機能の粒度は小さいものから大きなものまであり、その中のどのレベルのものをサービスと見るかによって、システムのパフォーマンスや柔軟性に影響します。

一般には、粒度が大きい方がパフォーマンスはよくなりますが、柔軟性が無くなり、粒度が小さいとその逆になります。しかし、具体的にどの位の粒度にすべきかということは、個別のシステム条件によっても異なるため、簡単には決まりません。そこで日立では、コンポーネントをおおまかに次の三階層に分けて考えて整理しています。

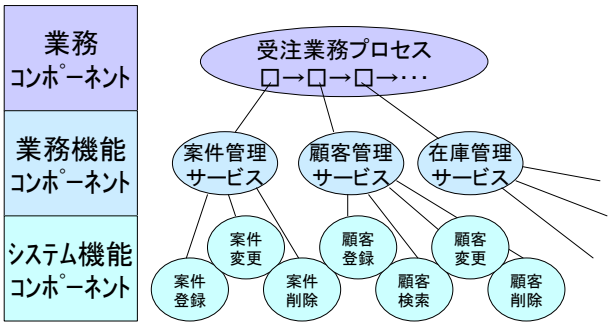

図3－2 コンポーネントの粒度

最も粒度の小さい階層は「システム機能コンポーネント」で、業務色は薄く、様々な業務から共通で使用される部品が多く含まれます。

これらの部品を組合せて作られるのが「業務機能コンポーネント」です。これは一連の業務処理を構成する業務ステップに対応するものが多く、一般的にはこの階層のコンポーネントをサービスと見なします。

さらに業務機能コンポーネントを組合せて作られるのが「業務コンポーネント」で、一つのまとまった業務処理に対応します。既存システムやパッケージの多くはこの階層のコンポーネントに相当します。業務フローの合理化等のよって、この業務コンポーネントをさらに組合せて複合業務を構成する場合があり、この階層のコンポーネントもサービス（又は複合サービス）となります。

(2)サービスの構造

サービス（サービス提供者）は、サービスリクエスタ（サービス利用者）からインタフェースを介して要求を受け、それに応じた処理を行なって結果を返します。この場合のサービスリクエスタは、オペレータまたはプログラム（他のサービス）です。ビジネスプロセスによって幾つかのサービスを組合せた複合サービスの場合も、インタフェースを介してサービス利用者にサービスを提供するという構図は同じです。

また、サービスの処理には、サービスリクエスタからの要求があると即時に実行され、結果が返るものと、要求を受け付けてから処理が完了するまでの時間差が大きいものがあります。後者の場合、サービスリクエスタが処理結果を知るためには、要求とは別の非同期インタフェースが必要になります。

さらに、サービスは多くの場合、要求メッセージとして渡されるデータ以外に、処理に必要なデータを持っています。例えば「顧客管理サービス」の場合を考えてみましょう。このサービスでは膨大な顧客データを管理し、要求時に渡された特定顧客のキー情報からその顧客に関わるデータを検索し処理するといった状況が想定されます。このとき顧客情報に対しては、例えば「配送管理サービス」から顧客の送付先を検索したり、「受注管理サービス」から新規顧客を追加したりといった、顧客管理サービス以外のサービスからの参照／更新もあり得ます。このように、サービス間でデータを共有する場合が考えられ、その実現方式には、複数のサービスが一つのDBを共有する方式と、DB間で何らかのデータ連携を行なうことによって仮想的に一つのDBに見せる方式があります。

図3－3に、サービスに関する処理の要求／結果と、サービス間のデータ連携の2通りのデータの流れを図示します。

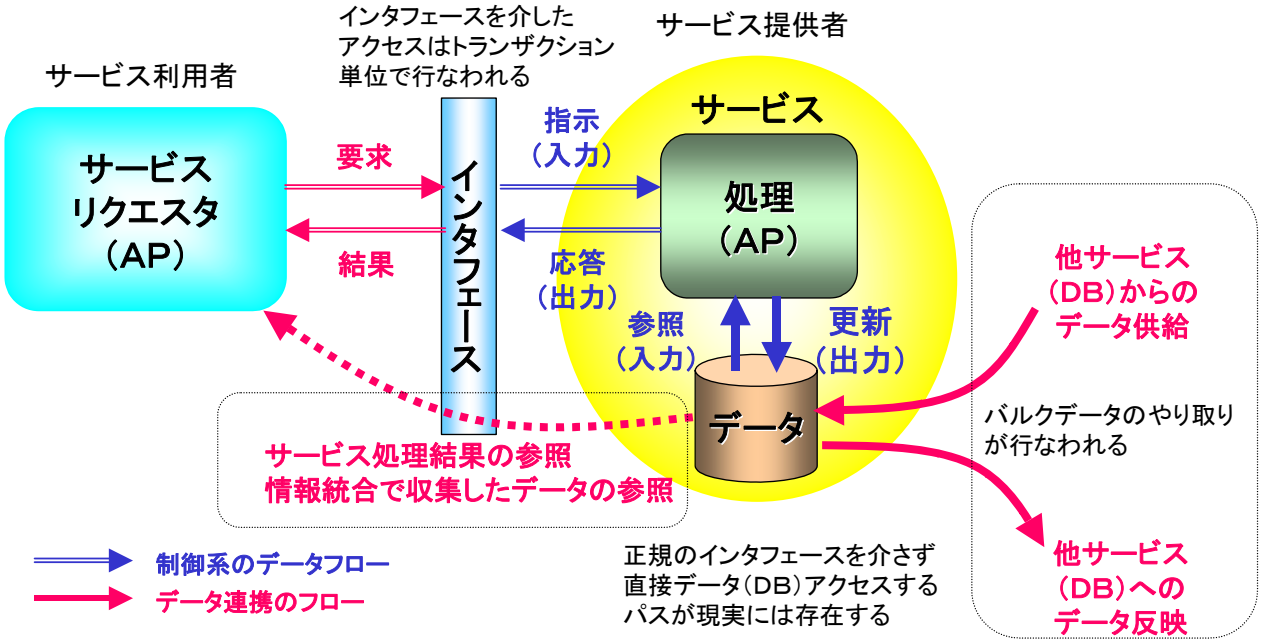

図3－3　サービスの構造

(3)サービスの性質

SOAの目的を実現するためには、サービスは次のような性質を備えている必要があります。

(a)サービスが独立している

サービスの独立性とは、あるサービスの処理内容を変更しても他のサービスには影響が出ないように変化の範囲を局所化するということです。これにより、開発工数だけでなくテスト工数を含めたトータルの工数を削減し、システム変更のスピードアップを実現します。このため、個々のサービスには、次のような性質を持たせる必要があります。

- 一つのサービス内であるまとまった処理が完結している
- 他のサービスと共有しているリソースが無い(データベースを除く)
- ビジネスプロセス上でサービスの実行順序を変えることができる
- 個々のサービスは単独で開発・配備・運用できる

(b)インタフェースが公開されている

様々なサービスリクエスタから、提供されているサービスを自由に利用するためには、機能的に汎用的であるということの他に、定められたインタフェースを経由すれば誰でもこのサービスを利用できるということのために、インタフェースの公開性が必要です。そこでサービスのインタフェースには次のような性質が求められます。

- サービス呼出しと結果確認を行なうためのインタフェースが公開されている
- インタフェースを介して誰からでも利用できる(セキュリティ上の制限を除く)
- ネットワーク上のどこからでも利用できる

(c)ビジネスプロセスによる連携が容易に行なえる

特にビジネスプロセスから利用するサービスの場合、処理のステータスをビジネスプロセスで管理する必要が無く、容易に利用できるという意味で、1回の問い合わせ応答で処理が完結することが必要です。

また、複数の更新系のサービスが一つのビジネスプロセス内にある場合、処理の同期合せが必要になります。そこで、後続のサービスで問題が発生した際に、更新済みの処理を取消すためのインタフェースが必要になります。ただし、これはすべてのサービスで実装可能な性質ではありません。例えば、プリンタ出力のように非可逆的なサービスもあります。このようなサービスについては、他の処理がすべて決着してから実行するようにビジネスプロセスを設計する等の工夫が必要です。

3.3 ビジネスプロセスの考え方

ビジネスプロセスの目的は、個々の業務ステップに対応したサービスを組合せ、一つの業務を実現する複合サービスを作るものということができます。そのために、SOAではサービスをどういう順序で実行するかというシナリオをBPELというXMLベースのスクリプト言語で記述します。

ビジネスプロセスとサービスの関係は、プログラムのメインルーチンとサブルーチンの関係に似ています。必要なサブルーチンが揃っていれば、メインプログラムを書き換えるだけで新しいプログラムを作成できるのと同様に、必要なサービスが揃っていれば、ビジネスプロセスを書き換えるだけで、新しい業務に対応することができます。また、メインルーチンと複数のサブルーチンで構成されるプログラムコンポーネントが、さらに他のメインルーチンからサブルーチンとして呼出されることがあるのと同様に、ビジネスプロセスと複数のサービスで構成される複合サービスを通常のサービスと同様に、他のサービスと組合せてビジネスプロセスを作ることも可能です。

他方、ビジネスプロセスの開発が通常のプログラミングと異なる点は、ビジネスプロセスの開発にはGUI操作で行なえるような開発ツールが提供されていることです。また、前述のようにサービスはサブルーチンと異なり、ビジネスプロセスとは異なるプラットフォーム上で動作させることが出来ます。これによって非常に効率の良い開発・保守が可能となります。

現実の業務フローを見た場合、審査・承認のように人間が介入する処理が含まれる場合があります。業種によっては、むしろこの形の業務の方が多いかもしれません。そこで日立のSOAでは、人間が行なう業務ステップも一種のサービスと見なし、ビジネスプロセスとして連携していくという考え方をしています。

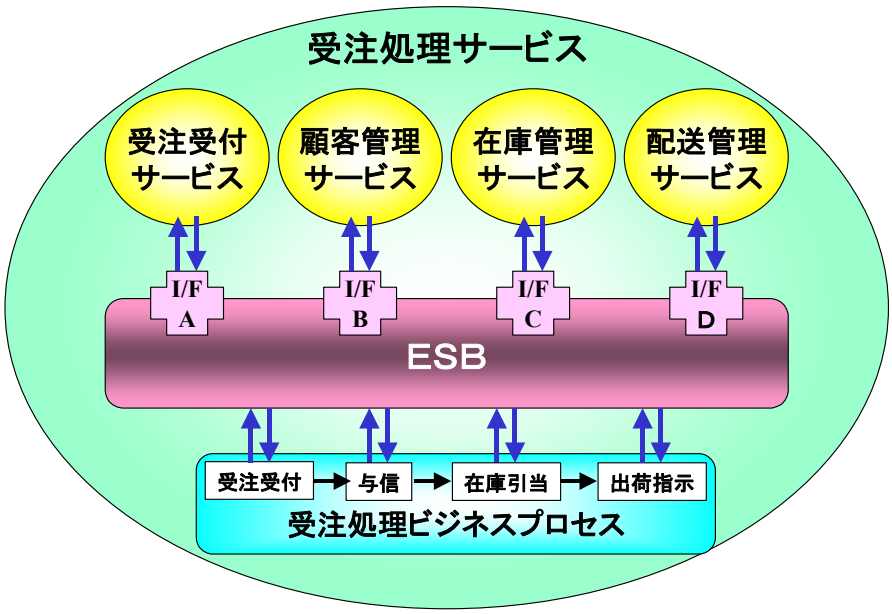

図3－4　複合サービスの例

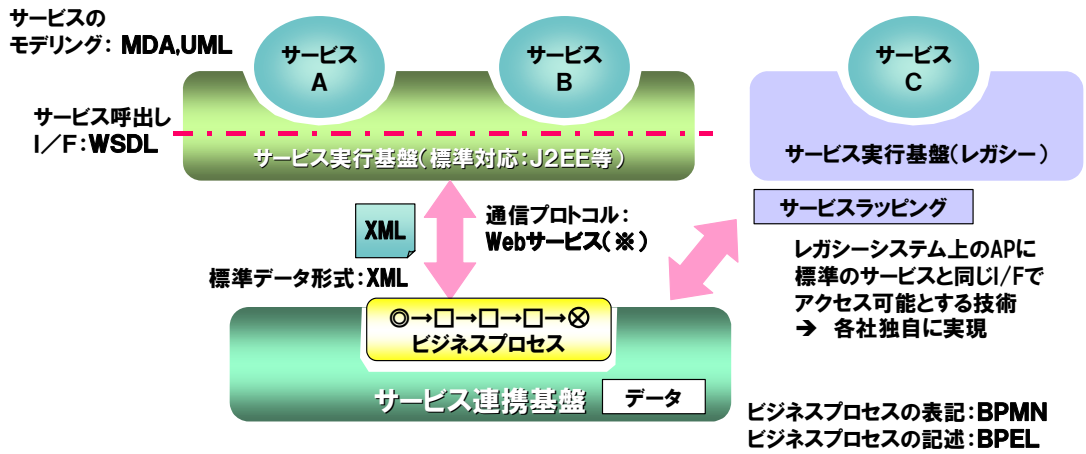

図3－5　SOAで使用される標準

3.4　メッセージ通信の考え方

サービス間の連携には、可能な限りWebサービスの標準プロトコルを使用することが、SOAの基本的な考え方です。標準を利用することによって、別々に開発された様々なサービス間をスムーズに連携することが出来、前述のサービスの性質を満たすことも容易になります。図3－5に主な標準を示します。

Javaや.NET といった新しい基盤を用いてサービスを新規に構築する場合は、特に問題なくこの標準に対応することが可能です。そして、すべてのサービスが標準に準拠しているような環境では、サービスリクエスタとサービスを直接連携されるシステム構成も可能です。しかし、この場合でも、個々のサービスが使用するメッセージをすべて一つのフォーマット（XML Schema）に統一することは困難な場合があります。またサービスを複数のノードに分散しておき、それらを運用スケジュールや負荷状況に合せて効率良く使い分けたいといった場合も考えられます。このようなときに、ESBを介してサービス間のメッセージを中継し、そこでメッセージ変換やルーティングを行なわせることで、容易に問題を解決することが出来ます。

さらに、メインフレームやTPモニタ等の上で動く既存アプリケーションをサービスとして利用したい場合、既存システムで直接Webサービスの標準プロトコルをサポートすることは困難です。このような場合には、既存アプリケーションを、ラッピング技術を利用してWebサービスに変換して連携します。また、この変換をESB側で行なえるようなアダプタを備えた製品もあります。

また、ESBの物理的な配置は、一つのESBサーバを中心として、複数のサービスリクエスタとサービスがスター状に接続されるHub＆Spoke型である必要はありません。サービスリクエスタ、サービス、ESBといった構成要素の、サーバ・ノードに対する配置は自由です。そして各サーバ上のESB間が連結されることにより、論理的なバスが構成されます。

次に、既存システムを活かして、このSOAのシステムモデルを適用するアプローチを見て行きましょう。

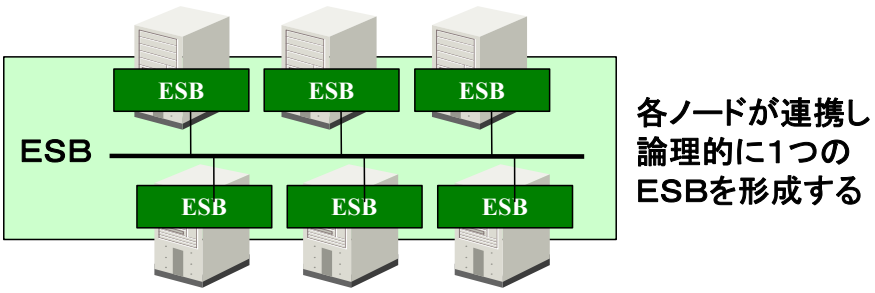

図3－6　バス型の連携

4. SOA適用のアプローチ

実際のSOA適用にあたって十分な効果を得るためには、事前に業務分析やシステム分析を行い、戦略を明確化することが必要です。日立では、そのためのコンサルテーションサービスや構築支援サービスもご用意しています。本章では、日立のSOAミドルウェア製品を用いた実際のシステム構築という観点から、SOA適用のアプローチをご紹介致します。

4.1 システムの構成法

SOAを適用したシステムの構成は、論理的には、まずESBを構築し、そこに個々のサービスをプラグインして行く形となります。

(1) 日立のESB製品

ＥＳＢを実現する日立のミドルウェアは、uCosminexus Service Platform です。これは、uCosminexus Application Server を含む最上位のスウィート製品で、J2EE をベースとしたメッセージング、ビジネスプロセス制御、サービス実行などの機能を、オールインワンで提供します。

標準インタフェースとして、同期的にサービス連携を行なうためにSOAP、非同期的にサービス連携を行なうためにWS－Reliability、ローカルにEJB呼出しを行なうためのRMI－IIOP、リレーショナルデータベースにアクセスするためのSQLをサポートしています。

図4－1は、日立のESBを用いた受注処理業務の一例を示しています。

まずESBがサービス利用者(オペレータ)からの要求を受けて、ビジネスプロセスを起動し、その中で「受注受付」「与信」「在庫引当」「出荷指示」といった業務ステップに対応するサービスを順次呼出して処理を行ないます。

ビジネスプロセスからサービスを呼出す際には、ESBの内部に持っているサービスディレクトリに問合せて目的とするサービスの位置情報を取得します。これは、サービス利用者からビジネスプロセスを経由せず、直接サービス呼出しを行なう場合にも同様です。この機能によって、サービスの変更を、呼出し側で意識する必要がなくなり、疎な連携を実現します。サービスディレクトリの内容は、ESBにサービスを追加した際に自動的に更新されます。

また、ビジネスプロセスから次に呼出すサービスに合せてデータ変換を行なうことができます。ここでは、XML Schema の変換だけでなく、既存システムとの連携を考慮してバイナリデータの変換もサポートしています。

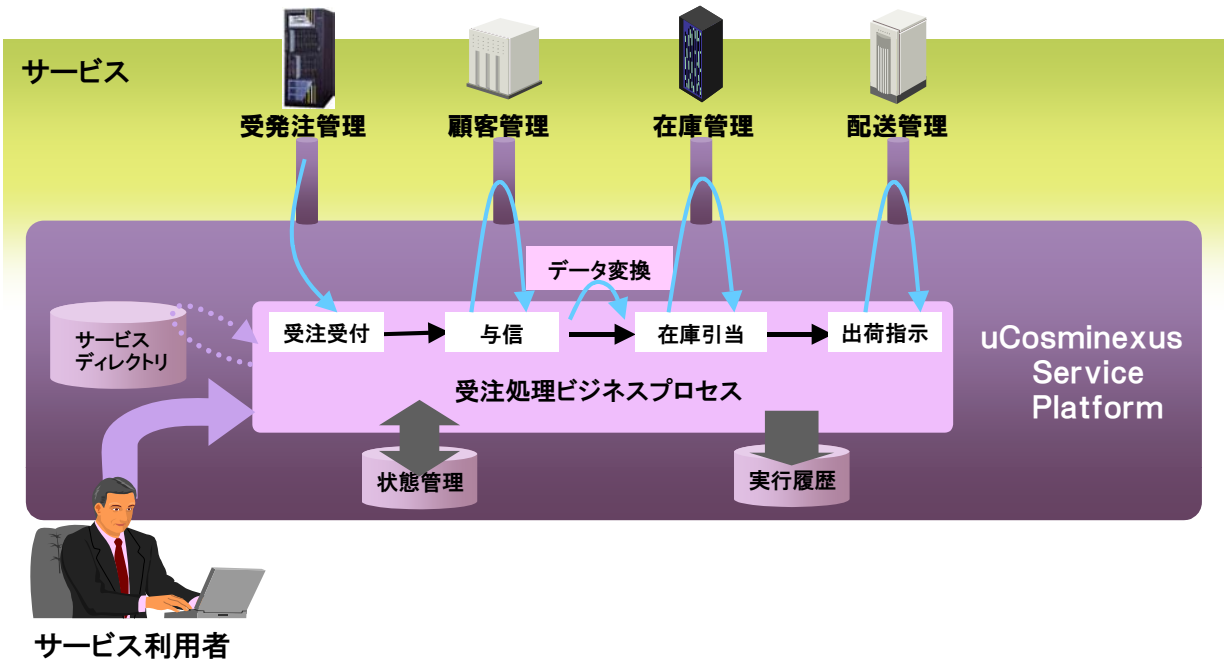

図4－1　日立のESB製品

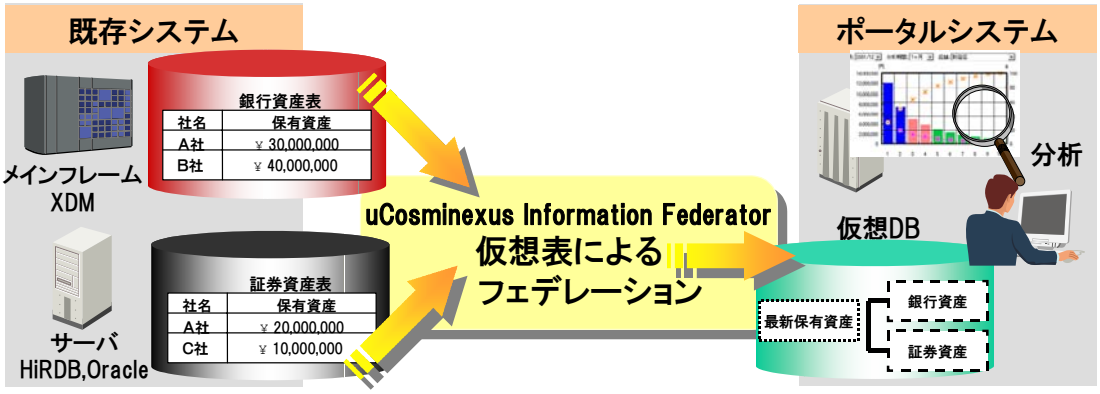

図4－2　フェデレーションによるデータ活用

(2)データ統合基盤

実際のシステムでは、3．2節で述べたように、サービス呼出しや応答受信のためのメッセージ通信以外に、サービス間のデータ(DB)共有のための連携が必要な場合があります。

そこで、このようなデータ連携のために、日立ではデータ統合基盤をご用意しています。日立のデータ統合基盤は、連携のリアルタイム性(データ鮮度)に応じて、フェデレーション、レプリケーション、ETLという三種類の連携機能があります。

(a)フェデレーション

仮想表によるデータリソースの仮想化を行い、必要なデータを複数のDBからオンデマンドで取得することができます。この機能を利用することによって、次のようなメリットを享受することができます。

- ヘテロ(異種・分散)なDBへの透過的アクセス
- 高鮮度データのオンデマンド参照
- 仮想表へのDBマッピング定義と仮想表への問合せのみ
- 新たなDBの構築は不要
- 新たな連携プログラムの開発は不要
- 既存の業務や運用への変更なし

(b)レプリケーション

連携元のデータベースから更新差分を抽出し、連携先のデータベースに反映する、ディレード連携機構を提供します。この機能を利用することによって、次のようなメリットを享受することができます。

- 更新情報のみを定期的に反映するので、処理負荷を抑制
- ログからの更新情報抽出により、抽出側オンライン業務への影響なし
- 更新情報の反映は定義のみで可能、プログラミングは不要
- データ変換用プログラムのアドオンも可能

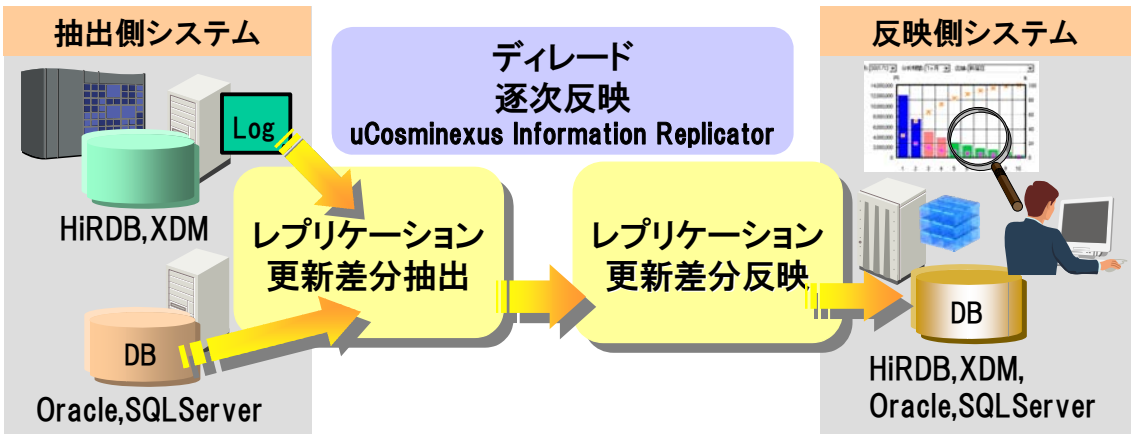

図4－3　レプリケーションによるデータ同期

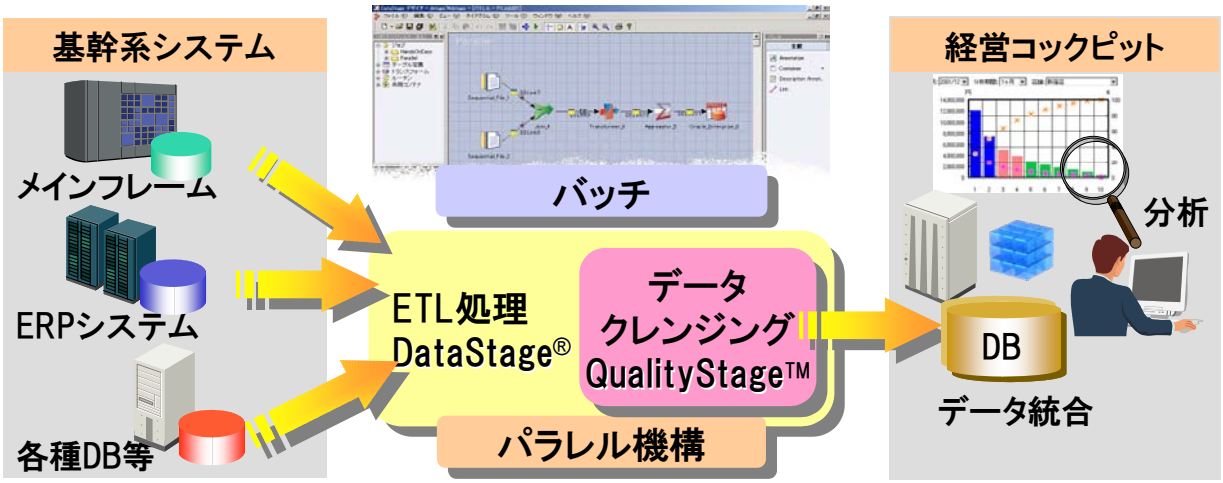

図4－4　ETLによるデータ収集

(c)ETL

一括抽出したバルクデータをバッチ処理で加工・変換し、統合データベースに反映します。この機能を利用することによって、次のようなメリットを享受することができます。

- 各種データソースのサポート
- 多彩な変換・集約機能による高い加工自由度
- 加工ロジックをプログラミングレスで作成
- 使いやすい開発環境
- 名寄せ等のデータクレンジングもサポートビジネスプロセスの開発

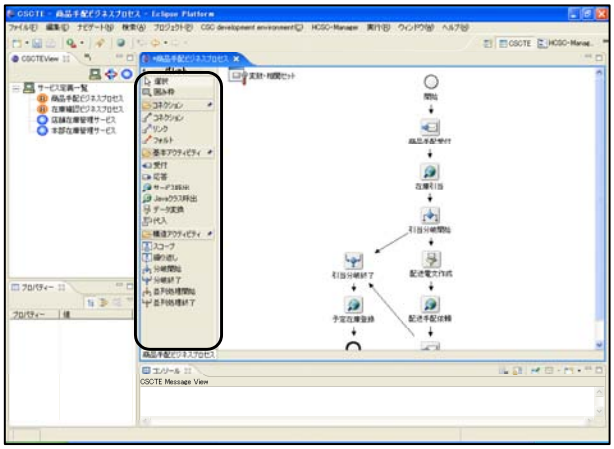

図4－5　ビジネスプロセス定義画面

4.2　ビジネスプロセスの開発

ビジネスプロセスの開発は、統合開発ツール(uCosminexus Service Architect)のGUIを使用した、以下に示す一連の操作により、コーディングレスで行なうことができます。このため、非常に効率良く開発を行なうことができ、システム開発の生産性を飛躍的に向上させることが可能です。

(1)ビジネスプロセス定義

最初に、ビジネスプロセス定義画面上に、アイコンを選んでドラッグ&ドロップで貼り付け、それを矢印で結んで行くという操作でビジネスプロセスを定義します。すべてGUI操作なので、BPELの言語仕様を知っている必要はありません。

(2)サービス接続定義

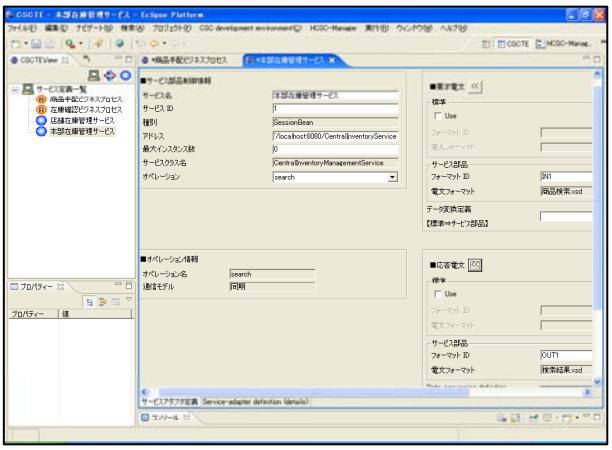

図4－6　サービス接続定義画面

次にサービス接続定義画面で、ビジネスプロセスから呼出される各サービスとの接続定義を行ないます。サービスのインタフェースとしてWSDLのファイルが提供されていれば、それを取込み、名前付け等の必要な設定を行ないます。また既存のサービスで、WSDLが提供されていない場合も、画面から接続先等の必要な情報を入力して定義を行ないます。ここでも、WSDLの仕様などの専門的な知識は必要ありません。

(3)データ変換定義

最後にデータ変換定義画面で、サービス間で受け渡すデータのフォーマット変換定義を行ないます。変換前のデータ構造と変換後のデータ構造を定義し、データ要素毎にマッピングを定義していくだけの簡単な操作です。XMLであれば、XML Schema を読込むだけで、データ構造の定義も必要ありません。

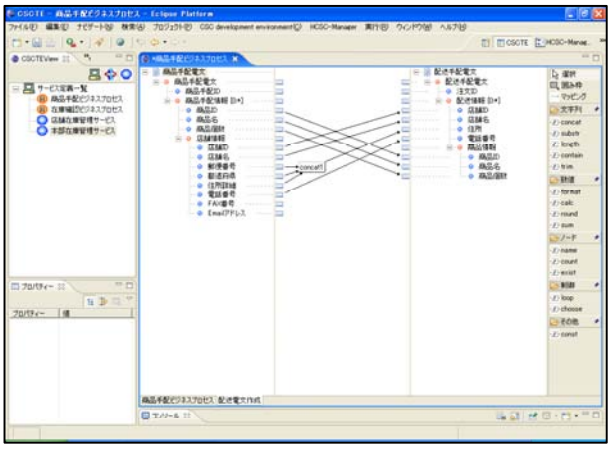

図4－7　データ変換定義画面

4.3　既存システムの連携

既存システムをサービスとして利用する利点として、次のようなことがあります。

- 既存資産の有効活用によって、新規開発を最小限に抑えることができる
- 実績ある資産の継続利用により、システムの信頼性や性能を維持できる
- アプリケーションをその特性に応じた最適なプラットフォーム上で稼動させることができる

しかし反面、次のような課題があります。

- サービスを必要な粒度まで分割することが困難な場合、その組合せを自由に変更して新しい業務を開発することは難しい
- サービスの条件を満たすことが難しい場合、特に処理の同期合せ等について、ビジネスプロセスの組立時に工夫が必要

そこで、これらの課題を踏まえ、目的に応じた既存システムの利用方法を、幾つかのパターンに分けてご説明します。

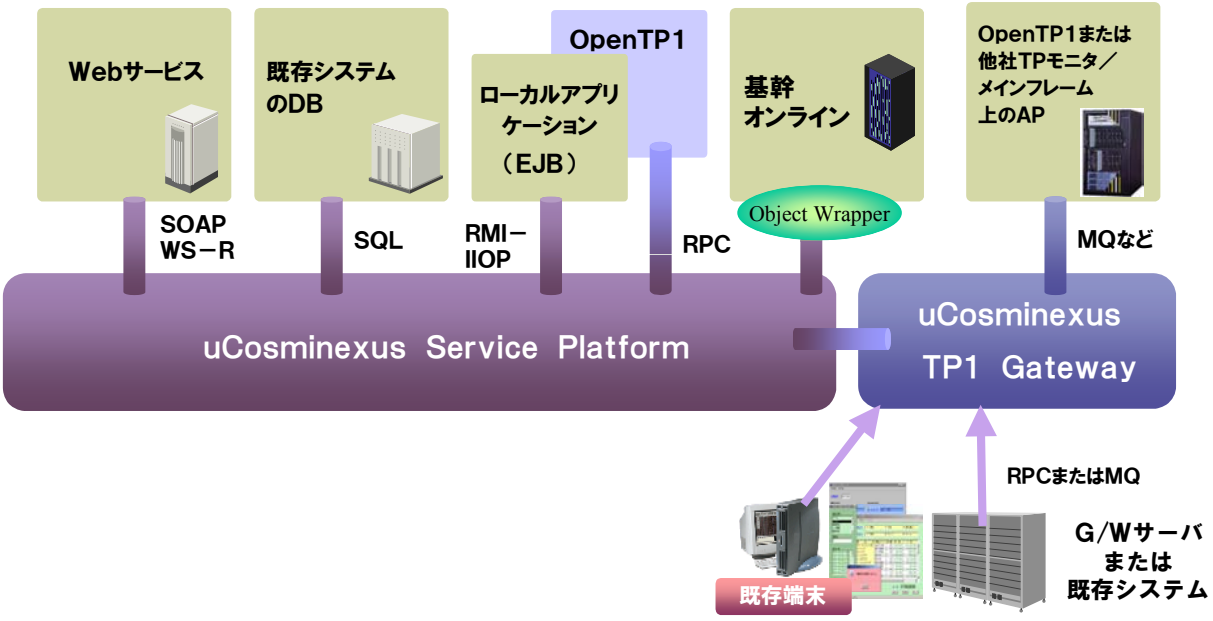

図4－8　既存システムとの連携パターン

(1)オンライン画面のラッピングによる連携

メインフレームなどのオンライン画面を持ったアプリケーションは、画面単位での処理内容を、ラッピング技術によって比較的容易にサービスとして連携することが出来ます。

日立では、ObjectWrapper と呼ばれる製品群をご用意しており、日立やIBMメインフレームのオンライン画面上のデータを解析して必要な情報を取り出し、それをサービスとして見せることが出来ます。

ただし、システム構築に当たって、次の点について注意が必要です。

- 画面解析等のオーバヘッドが大きく、特に複数サービスを連携させたリアルタイム処理では性能上問題となる場合がある
- サービスの内容を変更したい場合には、メインフレーム上のアプリケーションの改造が必要となる

(2)クライアント・サーバ型システムの利用

クライアント・サーバ型システムの場合、そのサーバ・アプリケーションも、比較的容易にサービスとして利用することができます。この場合必要となるのは、プロトコル変換とメッセージ・フォーマットの変換です。

日立では、ESBからOpenTP1上のアプリケーションを呼出す機能をオプションとしてご用意しております。

また、OpenTP1クライアントからESBを呼出したり、OpenTP1以外のアプリケーションと連携したりする場合には、uCosminexus TP1 Gateway という製品をご用意しております。

(3)データベースへの直接アクセス

メインフレームやERPパッケージを利用した基幹システムで、現在のシステムそのものには不満はないものの、新しい業務で基幹システムのデータだけを利用したいというケースがあります。この時、データベースを「情報提供サービス」と見なしてアプリケーションを経由せずに直接アクセスすることが考えられます。

日立のESBは、この目的のためにSQLインタフェースをサポートしており、HiRDBやOracle等のオープン系リレーショナルデータベースと連携することが出来ます。また、メインフレーム上のデータベースにも情報統合基盤を経由してアクセスすることが可能です。

尚、このケースでは、基幹システム上のデータの整合性を保つため、更新処理は既存アプリケーション経由のみで行い、新規システムからはデータ参照のみを行なうようにすることが必要です。

4.4 各処理形態への適応

SOAを適用したシステムで、特に既存システムを流用したケースでの、処理形態に応じた留意点には次のようなものがあります。

(1)オンライン処理

基本的には、サービス利用者から要求を受け応答を返すまで、同期的に処理を行ないます。幾つかのサービスを連携する場合もありますが、レスポンスタイムを考慮すると、あまり複雑なビジネスプロセスの処理を行なわせることはできません。

日立では、このような処理形態に対応するため、通常はDBで行なうビジネスプロセスのステータス管理を、すべてメモリ上で行なう高速処理モードをサポートしています。

(2)ワークフロー処理

多くの場合、サービス利用者から要求を受け付けた後、非同期で処理を行ないます。例えば、各種申請業務や、ユーザ登録、初期設定などの業務がこれに該当します。

このような処理形態では、間に審査・承認といった人間の判断を必要とする処理がはさまる場合がしばしばあります。基本的に、BPELではサービスを連携させて処理の自動化やワンストップ化を実現するための機能を備えていますが、人間系の処理はあまり考慮されていません。そこで日立では、ESB上でBPELによる自動的なサービス連携と、対人型ワークフローの両方をサポートしており、両者を組合せた処理を行なうことができます。

(3)バッチ処理

SOAにおけるサービスは、基本的にはオンラインのアプリケーションを指していますが、実際の既存システムでは、バッチ処理の方が多いのが実態です。また、バッチ処理についても、変更を容易に行ないたい、部品の再利用によって生産性を高めたいといったニーズは変りません。そこで、バッチ処理にSOAをどのように適用して行くかということが課題となります。

これには、次の二つのアプローチが考えられます。

(a)ロットを細分化してオンラインで処理する

現状はバッチ処理だが将来はオンライン化したいといった部分では、ビジネスプロセスの入口に専用のアダプタを設け、ファイルやデータベース経由で渡される入力データをレコード単位に分解して投入することによってオンライン処理に変換する方式が考えられます。このようにすることで、将来入力データがオンラインで渡されるようになっても、ビジネスプロセスとサービスはそのまま使用することが可能になります。

(b)バッチ処理をビジネスプロセスでコントロールする

将来的にもバッチ処理は変らない部分でSOAのメリットを活かしたい場合は、バッチ処理をサービスに相当する単位に分解し、ビジネスプロセスからそれをコントロールする方法が考えられます。この場合注意すべきことは、ビジネスプロセスやサービス・バスはバッチ処理で扱う大量データを流すのには適していないということです。そこで、データはサービス間でファイルやデータベースを経由して渡し、ビジネスプロセスからは処理フローのコントロールのみを行なうようにするということです。例えば、大量データの移動(ファイル転送)が必要な場合には、「ファイル転送サービス」を作り、ビジネスプロセスからはこれを起動するという方法が考えられます。

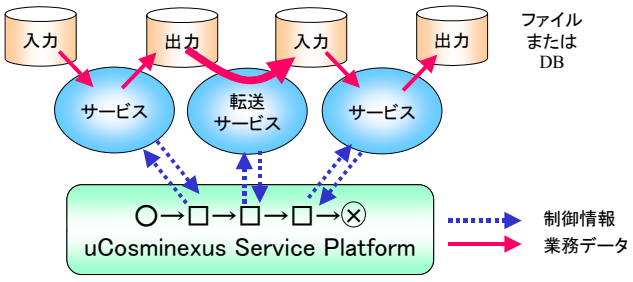

図4－9　バッチ処理パターン

(4)更新処理の同期合せ

ビジネスプロセスでデータの更新処理を行なう複数のサービスを連携させた場合、障害発生等によって、更新済みのデータを元に戻さなくてはならない場合があります。これに対してBPELには「補償」という考え方があり、更新系のサービスについて、予めその更新の取消しインタフェースを登録しておくと、障害発生時に自動的に更新処理を取消すことができます。このため、サービスは取消しインタフェースを備えていることが必要ですが、すべてのサービスでそれが可能な訳ではありません。

また取消しによる回復方法は、2フェーズ・コミットによる厳密な同期処理に比べ、次に示すような幾つかの問題点があります。

- 更新が完了してから取消し要求を受けるまでの間に、別の更新処理が挟まる可能性がある
- かなり遅れて取消し要求が到着する可能性があり、取消し可能な状態をいつ解除して良いかわからない

そこで、ビジネスプロセスを設計する際に、次の点に注意する必要があります。

- 2フェーズ・コミットによる厳密な更新処理の同期が必要なリソースは、一つのサービス内に閉じ込めるように設計する
- 取消しが不可能なサービスは、すべての更新処理が終ってから最後に実行させる
- 取消し処理の有効期限を決め、それ以降に問題が発覚した場合の運用対処を予め検討しておく

＜用語＞

SOA:　Service Oriented Architecture
BPEL:　Business Process Execution Language
BPMN:　Business Process Modeling Notation
BAM:　Business Activity Monitoring
ESB:　Enterprise Service Bus
XML:　eXtensible Markup Language
WSDL:　Web Service Definition Language
GUI:　Graphical User Interface
ETL:　Extract, Transform and Load
ERP:　Enterprise Resource Planning

Cosminexus

2006年7月5日発行
お問い合わせ先：株式会社 日立製作所 ソフトウェア事業部　販売推進部
cosminexus-s@itg.hitachi.co.jp

インターネットで製品情報をご覧いただけます。

http://www.hitachi.co.jp/cosminexus/
http://www.cosminexus.com/