# .NET 技術を利用したシステム連携

## System Integration using .NET Technology

稲　葉　　歩

**要　約**　近年，企業の持つ各種システムを柔軟に連携させ，開発を迅速化させるためのアーキテクチャとして，SOA（Service Oriented Architecture）が注目されており，その実現には，Web サービスや BPM（Business Process Management）などに対する検討が重要になってくる．本稿では Web サービスによるシステム連携に対する考え方と，それを Microsoft .NET 技術を利用して構築するための基礎的な知識を紹介する．

**Abstract**　SOA（Service Oriented Architecture）is recently remarkable topic of system developers and the information systems department of the company, for the purpose of flexible system integration and rapid development. In this context, we must consider Web Services, BPM（Business Process Management）, and so on. This paper presents the basic concept of system integration with Web Services, and the realization with the Microsoft .NET technology.

### 1.　は　じ　め　に

システム開発の現場において，新規開発か既存システムの置き換えかに関わらず，外部システムとの連携が必要なケースは非常に多い．ここでいう「外部システム」とは開発対象のシステムが独自に実装しない機能を持つ，既存の（あるいは同時期に開発中の）システムを指す．システム連携は，外部システムの持つ機能を再利用することによって，開発コストを削減し機能の重複を防止することが目的である．たとえ開発プロジェクトにおいて“システム連携”などといったキーワードが明示的に存在しない場合でも，システム連携が必要になってくる場合が多く，開発者が考慮しなければならない領域である．

本稿では，システム連携の代表的な考え方である“EAI”や“BPM”，そこで必須となるシステムの接続に対して現在最も注目が高いと思われる「Web サービス」，それらを実装するための技術としての「.NET」をキーワードに，その考え方や適用箇所などを紹介する．2 章では Web サービスと EAI・BPM について，3 章では .NET およびその関連技術を用いた実現について紹介する．

### 2.　システム連携と Web サービス

本章では，システム連携に関する様々なアプローチの紹介と，Web サービスの適用について紹介する．ここで述べる内容は基本的には .NET テクノロジには依存しないが，一部説明の都合上 .NET テクノロジに限っている部分もある．

#### 2.1　接続のアプローチ

システムが連携するためにはまず，接続する必要がある．そのためにはシステムの「どこに接続するのか」を決めなければならない．図 1 は一般的な論理 3 階層構造をもつ既存システムの各層に対する代表的な手法を示している．もちろん全てのシステムが必ずしもこのような構

造になっているとは限らないが，接続を検討している場所がどの層に相当するか，ある程度帰着させて考えることができるだろう．

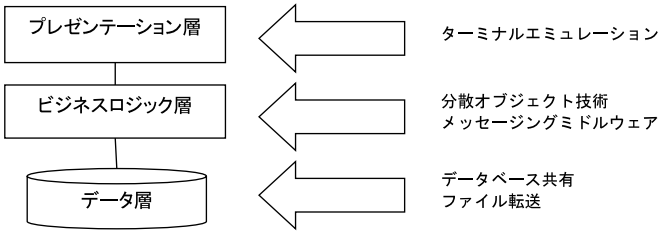



**図 1　システムの接続ポイント**

### 2.1.1　プレゼンテーション層における接続

プレゼンテーション層はユーザに対してデータ入出力機能を提供する．本来ならばユーザが画面上でデータを入力し，出力されたデータを解釈することで業務を行うが，別のアプリケーションが入出力を代行することで，既存システムの機能を再利用するアプローチである．

このアプローチは既存システムに対して変更を加える必要がないというメリットがある．このため変更を加えにくいレガシーシステムとの接続の際によく選択される．しかし，画面に出力されるデータには業務データ以外にも色情報や位置といった表示用のデータも含まれており，その中から必要な業務データのみを取り出すロジックを実装する必要があるため，非効率的で開発生産性が低いというデメリットも持つ．

### 2.1.2　データ層における接続

データ層はデータを永続化して保存・管理する機能を担当する．既存システムが持つデータに対して直接アクセスすることで必要なデータを得ることができる．

このアプローチは非常に自由度が高く，既存システムに対する変更もそれほど大きくならない[*1]というメリットがある．しかし，本来既存システムのために設計されたデータを外部から自由に変更できるという，カプセル化を破壊するアプローチでもあるため，慎重な設計が要求される．最悪の場合はデータ破壊によって既存システムが動かなくなってしまうケースもありうる．

### 2.1.3　ビジネスロジック層における接続

ビジネスロジック層は業務で必要な機能を担当し，業務データに対する様々な処理を行う．具体的には下記のような技術を利用する．

A）　DCOM, CORBA といった分散オブジェクト技術
B）　MSMQ, MQ Series などのメッセージングミドルウェア
C）　Web サービス

このアプローチでは本来システムが提供している機能（インタフェース）を純粋に利用できるため，プレゼンテーション層やデータ層における接続の際に発生するような問題点が発生しにくい．しかし下記のような別の問題が発生することが考えられる．

- システムを開発した段階では外部から接続されることを想定した設計になっていないこ

とが多いため，適切な粒度の機能がない場合がある

- テクノロジへの依存性が強く，接続するシステム同士の実装テクノロジが異なるケースでは，そもそも接続できないケースもありうる．

前者の問題についてはシステム設計方針の問題であるため技術で解決できる問題ではない．後者の問題については C）の選択肢を選ぶことである程度回避できる．

## 2.2 Web サービスによる接続

前節でとりあげた様々なアプローチから，主に Web サービスを取り上げて紹介する．

### 2.2.1 システム連携における Web サービスのメリット

Web サービスは，そのインタフェースが WSDL（Web Service Definition Language）という言語で記述され，SOAP の仕様に則ったメッセージを相互にやり取りすることで実現される．これらは多数のベンダにサポートされた業界標準仕様であり，XML によって記述される文書であるためテクノロジに依存しない．また，現在様々なベンダが Web サービスに対応した製品を発売していることも重要である．Web サービスの持つこれらの特徴から，その最たるメリットは相互運用性であると言える．つまり異種テクノロジ間の相互作用が必要となるシステム連携において，Web サービスは現在最も妥当な選択肢であるといえるだろう．

しかし標準“仕様”であるから実装は各ベンダが提供しており，各ベンダの仕様に対する解釈の差によって，標準であるにもかかわらず相性問題が発生しうる．このため Web サービス仕様の明確化と曖昧さの排除のために WS-I（Web Services Interoperability Organization）が設立されている．

### 2.2.2 Web サービスを適用したシステム

前述のように，Web サービスで接続するための場所はビジネスロジック層であり，これは Web サービスを提供する側（プロバイダ）となる．一方で，利用する側（コンシューマ）は，ビジネスロジック層にアクセスする可能性があるプレゼンテーション層（図 2 左）もしくは同じくビジネスロジック層（図 2 右）である．

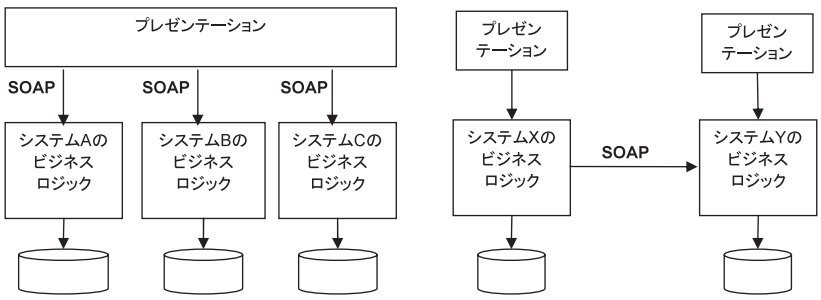



**図 2　Web サービスプロバイダとコンシューマ**

前者は様々なシステムの機能をひとつのユーザインタフェースに統合したポータルの形態である．後者はサーバサイドにおいて外部システムの機能の一部を利用する形態である．

### 2.2.3 Web サービスの構造

Web サービス自体は多くの場合 HTTP プロトコルを利用した SOAP メッセージのやり取りなので，TCP/IP プロトコルとテキスト（XML）の処理に対応するテクノロジであれば Web サービスとして接続できる．しかし自力で Web サービスに対応するのは非常にコストのかかる作業なので，通常の業務システム開発であれば Web サービスに対応した製品を利用することになる．多くの場合，Web サービスを実現する要素は図 3 のような構造になっている．

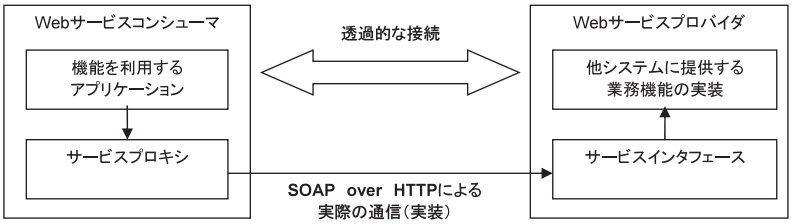



**図 3　Web サービスの構造**

サービスインタフェースは Web サービスが持つ機能を外部に公開し，内部の実装と結びつける役割を担う．サービスプロキシは公開された機能を呼び出すロジックを隠蔽する．この二つは共に，SOAP メッセージの生成と解釈，HTTP プロトコルによる通信といった，Web サービスにおいて必要な低レベルの実装を隠蔽する．こうすることで，コンシューマやプロバイダには Web サービスによる接続であることを意識させない実装が可能になる．

オブジェクト指向プログラミングにおいて参照関係のある二つのオブジェクトでも同じことだが，サービスインタフェースがあって初めてサービスプロキシが設計できる．注意しなければならないのは，Web サービスはシステム間を接続するために採用されているということである．Web サービスに限らず，システムが接続するインタフェースは外部のシステムとメッセージをやり取りする方法に対する契約であり，インタフェースを変更すれば接続できなくなるのが普通である．このためインタフェースの設計には十分な注意を払い，一度決定したら変更しないことを前提に開発を進める必要がある．

### 2.2.4 Web サービスに非対応なシステムとの接続

根本的な問題だが，既存システムは Web サービスとして機能を公開してないケースが普通である．既存システムの開発時期が古ければ Web サービスという技術自体が存在しないため対応しているはずがない．また，最近開発されたシステムであっても，設計当時に外部に Web サービスを提供する理由がなければ Web サービスを提供するはずがない．こういったケースには次のようなアプローチが考えられる．なお前提として，既存システムは Web サービスではないが外部からアクセスできる何らかのインタフェースを提供しているものとする．

#### Web サービスによるラッピング

まず最も簡単なアプローチが，既存システムの持つ機能を呼び出す Web サービスを実装し，公開する方法である．既存システムのテクノロジと親和性が高く，かつ Web サービスの作成が容易な技術を利用して Web サービスを手作りすればよい[*2]．こうすることで外部システムは既存システムのもつ機能を利用できる．また接続部分が既存システム実装に依存

しないため，既存システムの入れ替えが発生した場合にも対応可能となる．

ただし，Web サービス実装・公開が困難なケース，例えばメインフレームのような Web サービスを公開できる基盤を持たないようなケースでは，接続を集計するサーバが必要となるなどの問題が発生する場合がある．このようなケースでは後述のような EAI,BPM の適用を検討すると良いだろう．

**EAI（Enterprise Application Integration），BPM（Business Process Management）の利用**

EAI と銘打った製品は多数存在し，その機能に差があるが，基本的には N 対 N のシステム連携を実現するためのシステム基盤を構築することを目的としている．主要なメインフレーム，パッケージシステム，標準的なプロトコルなどと接続するための実装をあらかじめ備えているか，あるいは接続部分が拡張可能であったり，オプションとして購入することができたりする．EAI 製品はシステム群の中心として捉えられ，そこから放射状に EAI 製品と各システムが接続する“ハブ＆スポーク”と呼ばれるアーキテクチャを構成する（図 4）．

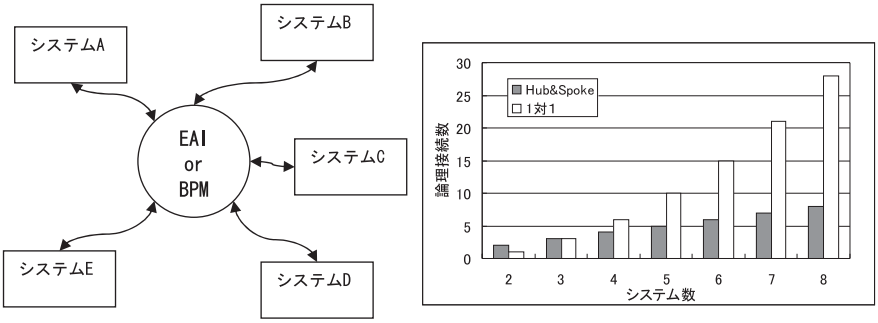



**図 4　ハブ＆スポークアーキテクチャと論理接続数**

EAI 製品はデータ連携の容易な実装と効率的な実現に焦点を当てており，処理を中継するブローカー的な役割を担っている．しかしその中心的な位置づけは，業務システムとそれらが提供する機能を統合・制御・自動化する役割を担うのに最適である．このように複数の業務システムの機能を柔軟に組み合わせることで業務プロセスの最適化に焦点を当てたものが BPM 製品である[*3]．

こういった製品を利用することで，多数のシステム連携が迅速に実現でき，柔軟性を維持することが可能である．また製品の性格上，現在提供されているような EAI・BPM 製品は Web サービスに対応していることがほとんどであり，新規システムとの接続も容易である．

### 2.2.5　EAI・BPM 適用の補足・留意事項

前項では説明の都合上 EAI や BPM といったキーワードを製品として扱っているが，これらは本来，システム連携を実現するために検討すべき項目やアーキテクチャなどを包含した概念である．これまで多数の開発者が様々な問題に直面し，その対策として考案されてきたものがこれらのキーワードに集約されていると考える．つまり“システム連携を実現すること”と“EAI や BPM と銘打った製品を導入すること”はイコールではない．これらの製品を導入することによるメリット・デメリットのトレードオフを十分に検討した上で適用して欲しい．簡

単ではあるが筆者の考えるメリット・デメリットを表 1 にまとめた.

**表 1　EAI・BPM 製品のメリット・デメリット**

| メリット | デメリット |
|---|---|
| ● 共通基盤の提供によるシステム連携の開発生産性・品質の向上<br>● 連携対象システムの実装・生存期間に対する非依存性による柔軟性の向上<br>● 接続数増大*の抑制による管理性の向上<br>● データフローの集中化による業務状況の統合監視<br>● 業務プロセスの可視化 | ● ハブがボトルネックになることやプロセス間通信の増加によるパフォーマンスの劣化<br>● 開発者の設計・技術能力に対する要求<br>● 全ての通信がハブ経由でなされるため、全体としての信頼性がハブに依存する。<br>● SW・HWコストや技術リスクの増加 |

*図 4 のとおり接続数は指数関数的に増大する．多数の多様な接続が存在し絡み合ってしまうことで，管理が困難な状態をスパゲッティ状態などと呼ぶ.

## 2.3　Web サービス設計方針

ここでは Web サービスの設計上，考慮する必要のあるいくつかのトピックを紹介する.

### 2.3.1　インタフェースの決定

Web サービス開発においてインタフェースの決定が重要であることを述べたが，ここでは Web サービスのインタフェースを開発するに当たって，どこから着手するかについて記述する．ここでは ASP.NET　Web サービスの開発に焦点をしぼり，代表的なアプローチを表 2 に紹介する.

**表 2　ASP.NET Web サービス開発のアプローチ**

| アプローチ | 概　　要 |
|---|---|
| A)<br>.NETクラスから | 最初にC#やVB.NETなどでサービスを提供する.NETのクラスを設計する。Visual Studio .NETがサポートしているアプローチであり、DataSetなど.NET固有のクラスが利用しやすい。 |
| B)<br>XMLスキーマから | 最初にシステム同士でやり取りされるXMLメッセージのスキーマを作成する。作成したスキーマから.NETのクラスを生成し*、これらを引数や戻り値にもつメソッドを提供するサービスクラスを作成する（→Aへ）。システムがやり取りする業務データを設計することになるため開発者・ユーザにとってわかり易く、XML形式で設計するため開発者の操る言語に依存しない。 |
| C)<br>WSDLから | 最初にWebサービスのインタフェース定義言語（WSDL）を作成する。作成したWSDLからサービスを提供する.NETクラスを生成する**(→Aへ）。メッセージの形式と共にWebサービスの厳密なインタフェースを早期に確定するが、WSDLに対する知識が必要になる。 |

*コマンドラインツール「XSD.exe」を利用する
**コマンドラインツール「WSDL.exe」を利用する

三つのアプローチは一長一短だが，システム開発の状況に応じて適切なアプローチを選択すればよい．例えば以下のようになる.

- コンシューマが Windows フォームを利用したスマートクライアントである
  プロバイダと同じテクノロジ（.NET　Framework）をベースとしており，異種テクノロジとの相互運用性が必要ないのであれば，DataSet など .NET　Framework のクラスライブラリや Visual Studio .NET の機能がフル活用できる A のアプローチ
- Web サービスによる新規の接続
  システムとしての再利用性やテクノロジ間の相互運用性を考慮する場合には，業務要求から設計に落とし易く，各システムの実装テクノロジに依存しない XML から設計する B のアプローチ

- 同じインタフェースを持つ複数の Web サービスを開発する
  既存の Web サービスとの入れ替えや，交換可能な Web サービス群を開発する場合など，同一の WSDL を持つ Web サービスを開発する場合には，インタフェースに誤差が生じない C のアプローチ

最終的にはどのアプローチも A に帰着することで，その後の開発手順に差異はない．サービスを提供するクラスが出来上がってしまえばいくつかの設定を行ったうえで公開すればよい．Visual Studio .NET を利用することで，これらの設定をある程度自動化できる．

### 2.3.2 Web サービスのパターン

Web サービスは SOAP メッセージを HTTP 上でやり取りすることで実現されている．その基本モデルは以下のように分類できる．

- Request-Response の組み合わせによる同期モデル
- Request のみの非同期メッセージングモデル

前者は，Web サービスコンシューマが作成した処理要求メッセージ（Request）と，それに対する Web サービスプロバイダでの処理応答メッセージ（Response）の組み合わせのモデルであり，後者は処理応答を期待しないモデルである．

しかしこの単純な通信モデルでは現実的ではない場合がある．具体的には Request から Response までの時間が保障できないようなケース[*4]である．処理結果が得られない，リクエストの到達保障が得られないなどの理由から，上記の単純な非同期メッセージングモデルで解決できるわけではない．この解決策として図 5 のような 3 パターンの Web サービス接続が考えられる．

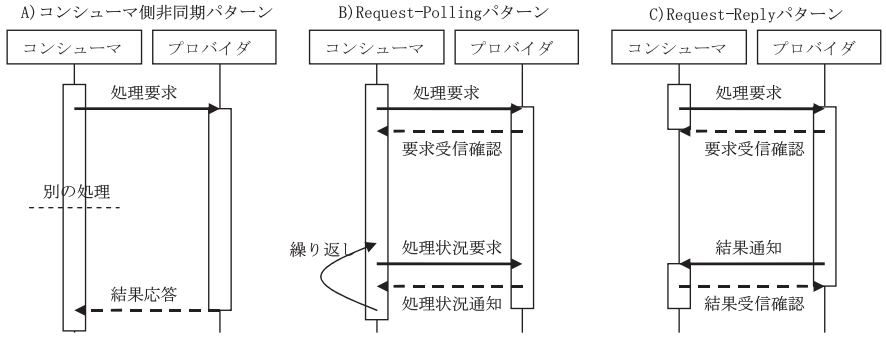



**図 5　Web サービスの非同期接続パターン**

A） コンシューマ側非同期パターン

最も手軽に実現できる手法であり，コンシューマ側で Web サービスのリクエストとレスポンスを非同期化することで，その間に別の処理の実行を可能にする．マルチスレッドプログラミングと同等の考慮が必要になる．

B） Request-Polling パターン

初回のリクエスト時にプロバイダは業務処理を実行せずに要求を受信した確認通知（Ack）を返し，その後業務処理を実行する．一方コンシューマは処理がどこまで進んでいるのかを確認するためのメッセージを送信し，プロバイダは処理状況や処理結果を

返す．コンシューマはプロバイダ側の処理が終わっているか否か把握できないので，処理状況要求は随時送信し，プロバイダは随時それを受け付けられる必要がある．

C） Request-Reply パターン

初回のリクエスト時にプロバイダは業務処理を実行せずに要求を受信した確認通知（Ack）を返し，その後業務処理を実行する．業務処理の終了後，要求元に関して結果の通知を返す手法である．このケースでは双方がコンシューマかつプロバイダとなる．

これらの 3 パターンをどのような局面で適用するかについて，代表的なケースを以下に紹介する．

#### ユーザインタフェースにおけるパターンの適用

ユーザインタフェースを実行するのはユーザの PC であり，そこで Web サービスプロバイダを起動すること（パターン C）は現実的ではないため考慮しない．パターン A は，処理時間が数秒から 1,2 分のオーダーの処理をバックグラウンドで実行するケースが考えられる．パターン B は，処理時間が数時間から数日のオーダーで，ユーザインタフェースの起動中に処理の終了が保証できないようなケースが考えられる．

#### サーバサイドビジネスロジックにおけるパターンの適用

パターン A は最終的には処理が終了するまでコンシューマが結果を待つ必要がある．つまり処理が長時間に及んだ場合には待ち状態になり最終的にはサーバリソースをロックしてしまうため，数秒程度のオーダー処理を複数並行に実行するケースが考えられる．パターン B は繰り返し処理状況要求を実行するための仕組みを実装する必要があり，また繰り返し間隔のチューニングも必要である．パターン C は SOAP メッセージングも必要最小限の回数で抑えられ，プロバイダ兼コンシューマとなる実装も現実的である．

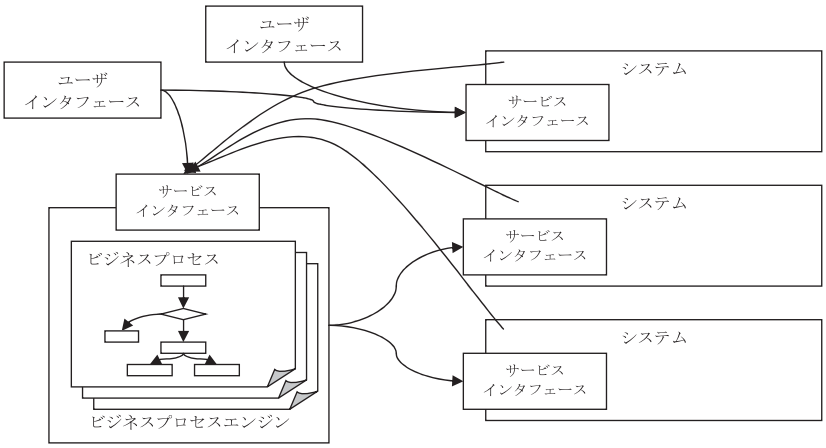
パターン B・C は長期間実行されるプロセスの状態を保持する必要があるため特殊な作りこみが必要になる．しかし一般的な BPM 機能をもった製品であれば，こういった長期間実行されるビジネスプロセスを実現する為のフレームワークを提供している．このことからサーバサイドのシステム同士が連携する場合には, BPM 製品の導入を考慮すべきだと考える．

### 2.3.3　EAI・BPM を適用したシステム連携と Web サービスの位置づけ

2.2.1 項ではシステム間の接続に Web サービスを採用することのメリットを，2.2.2 項ではシステムにおける Web サービスの適用箇所を，2.2.3 項では Web サービスのパターンとその対象箇所について，2.2.4, 2.2.5 項では EAI や BPM の適用について検討した．これらの検討から，BPM の概念や Web サービスを適用したシステムの構造は図 6 のようになる．

図 6 では各システムが自らのもつ機能をサービスとして外部に提供し，それらが全体として協調動作することでひとつの大きな業務システムを構成している．図 4 と異なる点はユーザインタフェースが登場している点，ビジネスプロセスを管理・実行するエンジンが登場している点である．

ユーザインタフェースはシステムの利用者が必要とする業務機能を実際に提供し，高い利便性や応答性が要求される．またユーザインタフェースが提供する機能[*5] によっては単一のシステムだけで実現しうることも十分ありうる．このため図 6 ではユーザインタフェースはビジネ




*BPM は EAI のより進化した形であると考えられるため, ここでは BPM として記述している.

**図 6　BPM*によるシステム連携**

スプロセスエンジンだけではなく直接システムとの接続も含んでいる．ビジネスプロセスエンジンは各システムの持つ機能を利用・協調動作させることで様々な業務プロセスを実現する．あるシステム（あるいはユーザインタフェース）から受け取ったメッセージを確実に別のシステムに届ける必要があるため，信頼性の高いメッセージング機構が必要となる．

#### Web サービスの位置づけ

図 6 において Web サービスの適用箇所はユーザインタフェース，各システム，ビジネスプロセスエンジンといった各要素を接続する矢印の部分となる．このように複数のシステムが連携する場合，各要素の実装が単一のテクノロジによって統一され，また今後もそれが継続されることは非常に考えにくい．このため Web サービスの持つ相互運用性が大きなメリットをもたらす．しかしすべての接続が Web サービスである必要はない．例えば既存システムがメッセージングミドルウェアを利用した他システム連携を既に実現している場合である．BPM 製品を適用しているならば，既に動いているインタフェースを無理に Web サービス化せずに，既存のインタフェースをそのまま利用すればよい．接続に Web サービスを用いるか，別のテクノロジを利用するかについては，そのテクノロジが要求を満たすかどうか十分に検討して欲しい．

## 3.　.NET 技術を利用したシステム連携

本章では, 前章で検討したシステム連携を実現するために利用できる. NET 関連テクノロジ, 主に Web サービス，BizTalk Server 2004 を紹介する．

### 3. 1　ASP.NET による Web サービスプロバイダ開発

. NET　Framework を利用したシステム開発においては ASP.NET で Web サービスを提供することが基本となる．後述の BizTalk Server 2004 においても，実際には ASP.NET の機能を利用して Web サービスを構築することになるので，これが全ての基本となる．

### 3.1.1　ASP.NET アーキテクチャと Web サービス

ASP.NET 上で動作する Web サービスの動作の仕組みは図 7 のようになり，その大まかな手順は下記のようになる．

1) Web サービスコンシューマ・アプリケーションがプロキシクラスに対して処理要求を行う．
2) プロキシクラスは処理要求を SOAP メッセージの形に変換し，HTTP プロトコルを利用して Web サービスプロバイダに送信する．
3) Web サービスプロバイダ側の IIS（Internet Information Service）が HTTP リクエストを受信し，ASP.NET ワーカプロセスに転送する．
4) ASP.NET ワーカプロセスはリクエストの内容を元に該当する Web サービスを実行する．

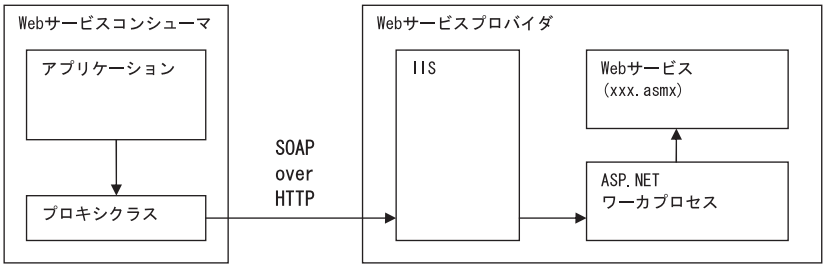



**図 7　ASP.NET アーキテクチャと Web サービスの概要**

#### Web サービスの動作

Web サービスの実行ファイル[*6] 自体はテキストファイルであるため，特殊なツール等がなくともテキストエディタで開発することが可能であり，いくつかの設定（仮想ディレクトリ設定とアプリケーション構成）をすれば Web サービスを公開することができる．二つの 64 ビット浮動少数点数の足し算（Add）のサービスを提供する簡単な Web サービスプロバイダと，それを利用するコンシューマを構築し，実際にやり取りされた HTTP プロトコルの内容をキャプチャしたものが表 3 である．

表 3 のとおり実際にやり取りされているのは SOAP 形式を持った XML フォーマットのデータである．SOAP メッセージ全体はエンベロープ（封筒）と呼ばれるルートノードの下に，ヘッダー[*7] とボディの二つのノードが存在する構造になっている．ボディには業務データそのものや処理内容が，ヘッダーには SOAP の拡張要素が含まれ，SOAP メッセージ全体で自己記述的な構造をとっている．

Web サービス自体はこのような標準の技術を用いて構築されているため，実装テクノロジに依存していないことがわかる．また，Web サービスの仕様は上記のようにシンプルであり，SOAP ヘッダーによる拡張性も考慮されている．しかしそれらを各ベンダやソリューションごとに独自に実装していては，Web サービスのメリットである相互運用性が満たせなくなり，従来のテクノロジと同様の問題が発生してしまうため，OASIS などの標準団体で様々な拡張仕様が検討されている．

**表 3　Web サービスの通信内容（抜粋）**

HTTPリクエスト

```
POST /SimpleWS/CalcService.asmx HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/Add"
Host: localhost:8080

<?xml version="1.0" encoding="utf-8"?>                          SOAPエンベロープ
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"   …>
  <soap:Header>                                                 SOAPヘッダー
    …..
  </soap:Header>
  <soap:Body>                                                   SOAPボディ
    <Add xmlns="http://tempuri.org/">
      <x>10</x>
      <y>20</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

HTTPレスポンス

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>                          SOAPエンベロープ
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"   … >
  <soap:Header>                                                 SOAPヘッダー
    ……
  </soap:Header>
  <soap:Body>                                                   SOAPボディ
    <AddResponse xmlns="http://tempuri.org/">
      <AddResult>30</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

### 3.1.2　Web サービスのセキュリティ

Web サービスの抱える代表的な問題がセキュリティである．上記のとおり HTTP プロトコルによって実際の通信を実現することから，通常の Web アプリケーションと似たような問題が当然発生する．まず認証の仕組みを持たないためコンシューマの特定ができず，アクセス制御が必要となるような業務機能が公開できない．また，表 3 で見られるように通信はテキストであるため，個人情報などの重要な業務データが漏洩する危険性が高い．さらに通信経路においてメッセージが改竄されても検知できないといった問題もある．これらの問題に対応するため，WS-Security という SOAP の拡張仕様が提唱されている．WS-Security には SOAP メッセージに対する認証データの付与，暗号化，デジタル署名といった仕様が定義されている．

#### Windows&ASP.NET Web サービスにおけるセキュリティ機構

WS-Security だけが問題に対応するための唯一の解ではなく，これ以外の仕組みも様々存在する．WS-Security の機能で過不足があるような場合にはそれらの方針も検討すべきである．図 7 のような構造をもった Windows&ASP.NET による Web サービスはいくつかのセキュリティを提供する仕組みを持つ．その仕組みは図 8 のようなレイヤ構造になっており，各階層において様々なセキュリティの仕組みが提供されている．

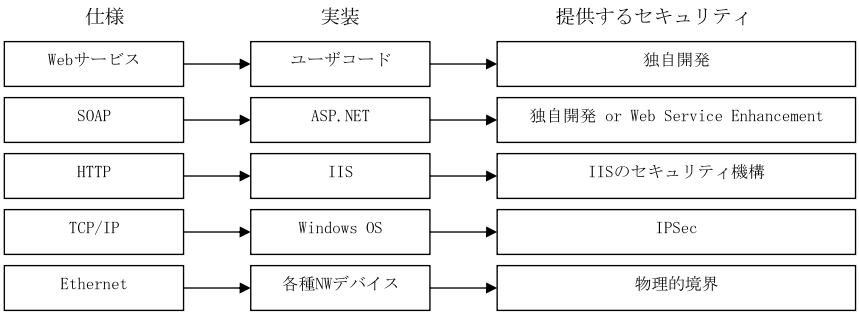



**図 8 Web サービスの仕様と実装とセキュリティ機能**

各レイヤにおけるセキュリティの仕組みは以下のようになる．

- Ethernet
  最も原始的な方法だが確実でもある．Web サービスに対して物理的にアクセスできないようにネットワークを構築する．
- TCP/IP
  IPSec を利用して通信プロトコルレベルでの暗号化を実現する．
- HTTP（1）
  IIS の機能を利用する．認証の仕組みとして Windows 統合認証，基本認証，ダイジェスト認証，匿名認証などが利用できる．
- HTTP（2）
  IIS に加えて PKI の仕組みを利用する．サーバ証明書を用いた SSL 暗号化，クライアント証明書を利用した証明書認証が利用できる．
- SOAP（1）
  WS-Security 仕様の Microsoft 実装である Web Service Enhancement（WSE）を利用する．標準仕様をベースにしているため相互運用性が期待できる．
- SOAP（2），Web サービスレベル
  SOAP ヘッダーもしくはボディに認証の仕組みを独自に組み込む．

これからわかるように上位レイヤになるほど自由度が高くなる反面，開発コストは大きくなる．万能な仕組みというものが存在しないため，要求される内容や各種仕組みの持つ技術的制約などに応じて，適切に選択し組み合わせる必要がある．

#### Windows&ASP.NET Web サービスにおけるセキュリティストラテジ

図 8 に示される仕組みの組み合わせは複数あるが，その中には無駄な組み合わせというものも存在する．極端な例だが，クロスケーブルで直結されたマシン同士の Web サービスに WS-Security による認証や SSL の暗号化が必要だろうか．また独自実装は開発コストの面からお勧めできない．現実的な組み合わせはいくつかにしぼられるので，代表的な例を表 4 に紹介する．

### 3.2　Web サービスコンシューマの開発

Web サービスコンシューマが Web サービスプロバイダに対して表 3 のような SOAP メッ

**表 4 Web サービスのセキュリティストラテジ**

| パターン | 適用に適切な状況 |
|---|---|
| IPSecによる暗号化 | 社内イントラネット環境などの限定された範囲でのサーバ間Webサービスなど。 |
| 証明書認証とSSL暗号化 | インターネット経由でのWebサービスによるB2Bなど。Webサービス以外でも既にインターネット上の通信として多くのケースで利用されており実績がある。 |
| WS-Security認証とSSL暗号化 | クライアントPCからのWebサービスアクセスなど。多数のユーザに証明書を配布し管理する手間がない。またSSLはIISではなくハードウェアアクセラレータの利用によってパフォーマンスの向上が可能。 |
| WS-Securityによる認証・署名・暗号化 | ひとつのメッセージが複数のWebサービスによって処理される場合など。WS-Securityは通信レベルではなくメッセージレベルで実現されるため、SOAPメッセージの部分的な暗号化など、End-to-Endのセキュリティが確立できる。 |

セージングを実行することで Web サービスによる接続が成立する．本節ではコンシューマ開発の方法を紹介する．

### 3.2.1 .NET Framework テクノロジの利用

図 7 図 3 に示されるように，コンシューマ側はプロバイダのプロキシを作成することで，提供されるサービスを利用する．.NET 開発におけるプロキシの作成方法は以下の 2 種類であるが，基本的には A）の方が操作も管理も容易である．どちらを利用してもプロキシクラスが作成され，全ての.NET アプリケーションから利用可能である．

A） Visual Studio .NET の「Web 参照の追加」
ウィザードに従って WSDL を指定すればプロキシクラスが作成され利用可能になる．

B） コマンドラインツール「WSDL.exe」
処理自体は上記の「Web 参照の追加」とほぼ同様であるが，より柔軟なオプションが指定できる．

### 3.2.2 その他の Microsoft テクノロジの利用

.NET Framework 以外のユーザインタフェースにおける Microsoft テクノロジの代表は Microsoft Office 製品であり，その選択肢は表 5 のようになる．

**表 5 ユーザインタフェースの種類と特徴**

| 種　類 | 特　　徴 |
|---|---|
| Office 2000、XP、2003 | ● Office Web Services Toolkitを使用する<br>● VBAから利用するプロキシクラスの生成 |
| Excel 2003、Word 2003 | ● Visual Studio .NET＋Visual Studio Tool for Officeによる開発<br>● .NETマネージドコードが実行可能<br>● .NETクラスライブラリが利用可能 |
| InfoPath 2003 | ● Webサービスにネイティブ対応 |

このようなリッチクライアントテクノロジを選択した場合には，ユーザインタフェースの実行に何らかのソフトウェアのインストールが必要になるので，管理や配布のコストを考慮して選択する必要がある．

## 3.3 BizTalk Server 2004 における Web サービス

Microsoft が提供するシステム連携製品が BizTalk Server である．本節では BizTalk Server 2004 における Web サービスの位置づけについて紹介する．BizTalk Server 2004 における具

体的な開発については紙面の都合上紹介しきれないため，簡単な紹介に止める．

### 3.3.1 BizTalk Server 2004 概要

二つのシステムと BizTalk Server 2004 によるシステム連携の概要図は図 9 のようになる．

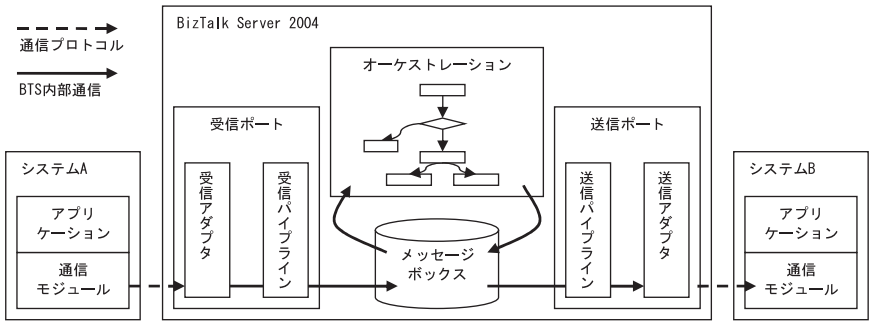



**図 9　BizTalk Server 2004 によるシステム連携フレームワーク**

図中の破線は各システム固有の仕様によるメッセージング，実線は BizTalk Server 2004 内部での XML メッセージングである．図中に表れる各モジュールは表 6 のような役割を担う．

**表 6　BizTalk Server 2004 の主要な構成モジュール**

| モジュール | 役　　割 |
|---|---|
| 受信ポート | 外部システムからメッセージを受信する。複数のプロトコルによるメッセージ受信を実現するために、受信アダプタと受信パイプラインを取りまとめた“受信場所”を１つ以上取りまとめたもの。 |
| 受信アダプタ | 実際の接続プロトコルの受信処理を実装する。既定ではSOAP、HTTP、FILE、MSMQT、EDI、FTP、SQLといったプロトコルに対応する。独自実装、サードパーティ製品による実装の追加が可能。 |
| 受信パイプライン | アダプタが受信したメッセージに対して後処理を行う。後処理にはデコード、逆アセンブル、検証といった複数の“ステージ”に分割されており、これらが順次処理される。各ステージの処理は自由に組み合わせ可能であり、処理の独自実装も可能である。 |
| メッセージボックス | 受信ポートやオーケストレーションによって処理されたメッセージを、確実にオーケストレーションや送信ポートに渡すために格納する場所で、BizTalk Serverの心臓部分。実体はSQL Server データベースであり、MSCS（Microsoft Clustering Service）などを利用してメッセージングの信頼性を確保する。複数のBizTalk Serverのプロセスからメッセージボックスを共有することで、BizTalk Serverはスケールアウトが可能となっている。 |
| オーケストレーション | 受信ポート（あるいはオーケストレーション）が処理したメッセージに対して、変換、条件分岐、ループ、並行処理などといった様々な処理を組み合わせ、ビジネスプロセスを実現する。仕掛かり中のオーケストレーションの状態も随時SQL Serverに永続化され、途中で待ち状態に入るような、長時間実行されるビジネスプロセスをサポートする。開発はVisual Studio .NET上でのGUI操作によって行われ、ビジネスプロセスの可視化を実現している。 |
| 送信ポート | オーケストレーション（あるいは受信ポート）が処理したメッセージを外部システムに送信する。送信パイプラインと送信アダプタを取りまとめたもの。複数の送信先に同時に送るような場合には、送信ポートを複数取りまとめる“送信ポートグループ”を使用する。 |
| 送信パイプライン | 受信パイプラインと逆にアダプタが送信するメッセージに対して前処理を行う。前処にはプリアセンブル、アセンブル、エンコードといった複数の“ステージ”に分割されており、これらが順次処理される。各ステージの処理は自由に組み合わせ可能であり、処理の独自実装も可能である。 |
| 送信アダプタ | 実際の接続プロトコルの送信処理を実装する。既定ではSOAP、HTTP、FILE、MSMQT、EDI、FTP、SQL、SMTPといったプロトコルに対応する。独自実装、サードパーティ製品による実装の追加が可能。 |

### 3.3.2 BizTalk Server 2004 によるシステム連携の開発

BizTalk Server 2004 の開発環境は Visual Studio .NET に統合されており，.NET 技術者には馴染みの開発環境を利用した開発が可能である．以下に Visual Studio .NET 上で利用でき

る代表的な BizTalk Server 2004 の開発ツールを紹介する．

- XML スキーマをグラフィカルに開発する“ BizTalk エディタ ”
- ある XML メッセージを別のスキーマを持つ XML に変換するマップをグラフィカルに開発する“ BizTalk マッパー ”
- 受信，送信，変換，条件分岐といった“ シェイプ ”と呼ばれる様々な図形を配置・組み合わせることで，グラフィカルにビジネスプロセスを開発する“ オーケストレーションデザイナ ”
- 受信ポートや送信ポートの作成を行う“ BizTalk エクスプローラ ”
- パイプラインの各ステージで行う処理をグラフィカルに配置して組み合わせ，カスタムパイプラインを作成する“ パイプラインエディタ ”
- オーケストレーションを Web サービスとして公開する“ Web サービス公開ウィザード ”
- 外部 Web サービスの WSDL を読み込んで，接続に必要な XML メッセージのスキーマを生成する“ Web 参照の追加 ”

### BizTalk Server 2004 における Web サービスの位置づけ

BizTalk Server 2004 はオーケストレーションを Web サービスとして公開する（プロバイダ）ことも，オーケストレーションから Web サービスを利用する（コンシューマ）ことも可能であり，どちらもウィザード形式で操作を行う．BizTalk Server 2004 において，Web サービスは外部システムとの接続方法の 1 種（図 9 の点線部分に相当）という位置づけであり，SOAP アダプタによって実現されている．これ以外にも標準で用意されたアダプタや，サードパーティ製のアダプタ，あるいは独自に開発したアダプタを利用して各種の業務システムと接続可能である．このため必ずしも接続が Web サービスである必要はなく，既存の業務システムが持つインタフェースと最も親和性の高いアダプタを購入するか，場合によっては自作して接続すればよい．

### ASP.NET Web サービスとの比較

BizTalk Server 2004 はその実行基盤に.NET Framework を採用しており，最終的に作成された Web サービスは ASP.NET フレームワーク上で動作する[*8]．このため実行レベルでは本質的な差異はないが，その開発の進め方が大きく異なる．

表 2 に示されるように，ASP.NET そのものを利用した Web サービスの開発は，クラスや XML の作成から始まり，その処理の実装（コーディング）までが“ テキストベース ”で進められる．つまり XML スキーマ，WSDL，C#（あるいは VB.NET）などといった技術的な専門知識が要求される．

一方で，BizTalk Server 2004 を利用した場合，XML スキーマの開発や，その内部処理（オーケストレーション，マップ等）の多くがグラフィカルに進められる．専門知識の不足をツールでサポートし，開発の敷居を下げることが可能である．このようにして開発コストを下げることで，開発者はビジネスプロセスの設計・改善などといった，どちらかというと業務知識が要求される作業に注力することが可能である．また技術的な専門知識をもたない業務アナリストでも，ビジネスプロセス（オーケストレーション）やメッセージ構造（XML スキーマ）が可視化されるため，実装に対するある程度の理解が可能となり，業務アナリス

トと開発者のギャップを埋める一助となるだろう．

### 3.3.3　ASP.NET か BizTalk Server 2004 か

Web サービスに限った話ではないが，BizTalk Server 2004 は以下のような処理をフレームワーク的に実現している．

- 受信送信を構成するモジュールの分割
- メッセージを永続化することによる，メッセージングの信頼性向上
- オーケストレーションの永続化による状態保持
- 処理のトラッキング，等

一般的に BizTalk Server 2004 で実行される Web サービスは，ASP.NET そのものを利用して開発された Web サービスに比べて，上記のようなリカバリ対応などの多くの処理を実施しているためオーバーヘッドが大きい．もちろんこれらの処理を自力実装することは非常にコストが高いので，BizTalk Server 2004 は状況を見極めての導入が必要となる．

ASP.NET は粒度が細かく，ビジネスプロセスを構成する細かい業務処理を実装するのに適していると言えるだろう．細かい業務処理のパフォーマンスが悪くてはビジネスプロセス全体に悪影響を与える．また開発者にとって細かい業務処理は，C# や VB.NET といったプログラミング言語が最も表現しやすいと考える．

BizTalk Server 2004 はより粒度が大きく，長期間実行されるようなビジネスプロセスの実行に適しているといえるだろう．このような場合には，図 6 に示されるようなアーキテクチャが必要になってくるため，図 9 に示されるようなフレームワークが提供されることは大きなメリットとなる．

## 3.4　.NET 技術を利用したシステム連携

2.3.3 項で紹介したシステム連携のアーキテクチャと，本章で紹介してきた.NET 関連技術をマッピングすると図 10 のようになる．このように.NET 関連技術のみを利用しても，システム連携のほぼ全てを実装することは可能である．

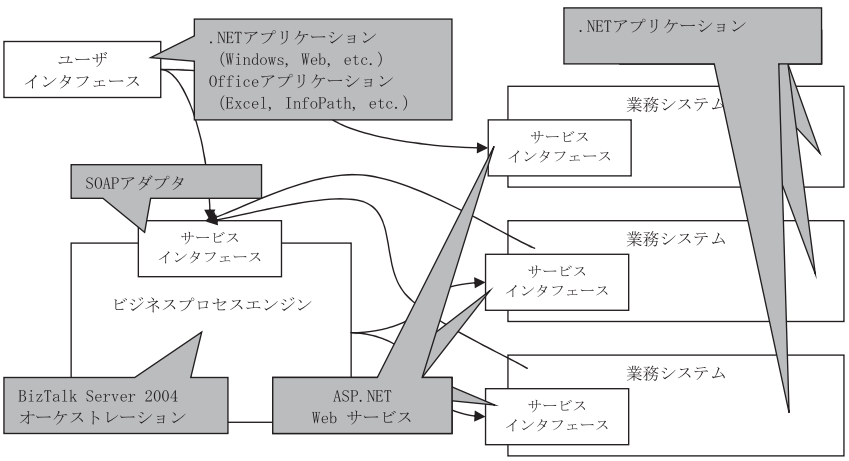



**図 10　.NET 技術で実現するシステム連携**

図 10 はこれまで紹介してきたシステム連携の構造と .NET 技術の対応について，その全てではないが総まとめしたものである．しかし「.NET でシステム連携を実現」といった場合に，「必ずしもこうしなければならない」といったものではないことは注意していただきたい．2 章で示したように，この構造にはメリットとデメリットがあり，さまざまな検討事項に対するトレードオフの上で成り立つべきものである．システム連携を検討する場合には，システムに対する要求と，それを満たすためのアーキテクチャに対する十分な検討をお願いしたい．また，Web サービス以外，.NET 以外，BizTalk Server 2004 以外の技術や製品も存在し，考慮しなければならないことを常に意識しておいてほしい．

## 4. おわりに

本稿では，2 章でシステム連携に対する検討の指針を，3 章ではシステム連携における .NET 関連技術の適用を，主に Web サービスに焦点をおいて紹介してきた．しかし，ある企業のシステムが全て .NET 技術で構築されているケースは非常に稀であり，その全ての接続が Web サービスでなければならない理由はない．.NET Framework だけで実現できるシステム接続に限っても，その接続方法には Remoting,MSMQ,HTTP,Socket など様々な選択肢があり，どれが良い・悪いという評価は状況に応じてすべきであり，技術だけを根拠に優劣はつけられない．また，BizTalk Server 2004 以外にも EAI 製品は多数存在し，これも優劣がつけがたい．システム連携に限った話ではないが，実装技術以外にも，企業のポリシー，予算，開発・運用のコスト，開発者のスキル，政治的制約など様々な変数を考慮した上で，システムの最適な姿をデザインし，適切なテクノロジやアーキテクチャを選択するべきである．

問題となるのは，目先の課題だけに焦点を当てて局所最適解を求めてしまうことである．システム連携を細かく分割していけば最終的には 1 対 1 の接続に帰着される．こうすることで複雑に見える問題を簡略化することができる．しかし，その寄せ集めがシステム連携における「スパゲッティ」状態を作り出し，結果的には管理コストの増大や保守性の低下につながってしまう．これは「最適」を評価するための指標を「開発の容易さ」においてしまった例であり，これが企業システム全体の最適な姿と一致しないことが問題である．

システム全体の最適な姿とはどんなものか，全体最適な解にたどり着くためにはどうするべきか．そのために考案され，議論されているのが EA（Enterprise Architecture）や SOA（Service Oriented Architecture）であり，本稿がそれらのアーキテクチャを元にしたシステム開発の一助となれば幸いである．

---

＊ 1　例えば RDBMS を共有してしまうと，既存システムはまったく変更する必要がない．
＊ 2　例えば Windows の COM アプリケーションがある場合には，.NET Framework の機能を利用して，内部的には COM を呼び出し，外部に対しては ASP.NET で Web サービスを公開する．
＊ 3　ここでは EAI 製品と BPM 製品を区別しているが現実にはその境界線は曖昧である．製品導入の際には必要な仕様を満たしているかを十分に検討していただきたい．
＊ 4　プロバイダ側での処理に非常に時間がかかる，通信速度の遅い回線を使用しているなど
＊ 5　マスタデータのメンテナンス画面などが考えられる．
＊ 6　ASP.NET の既定の状態では拡張子 asmx のファイルが Web サービスに関連付けられる．ここで言う実行ファイルとはエンドポイントとなる asmx ファイルとそれが利用するアセンブリを指す．
＊ 7　シンプルな Web サービスを構築したため SOAP ヘッダーを利用しておらず，本来はキャプ

チャ内容に含まれていないが，ここでは参考のために位置だけを示しておく．

＊ 8 オーケストレーションは.NET のアセンブリにコンパイルされ，Web サービス公開ウィザードはオーケストレーションを呼び出すためのロジックを持った ASP.NET　Web サービスを作成する．

**執筆者紹介**　稲 葉　歩（Ayumu Inaba）

2002 年東京大学大学院工学系研究科航空宇宙工学修士課程修了．同年日本ユニシス(株)入社．.NET ビジネス専任組織「.NET ビジネスディベロプメント」でシステム開発支援および提案支援に従事．現在，日本ユニシス・ソリューション(株)AD CoE コンピテンスセンタ .NET テクノロジコンサルティングに所属．