.NETフレームワーク

萩原 正義
マイクロソフト（株）　masayh@microsoft.com

.NETインフラストラクチャ

2000年6月に発表された.NETはWebサービス，XML対応アプリケーション，3階層型アプリケーション，B2Bメッセージングサービスと複雑な業務プロセスの実行，マルチデバイスGUIクライアント，アプリケーションコンポーネントなど，ソフトウェアサービスとエンタープライズアプリケーションの実行と開発を主目的としている．従来のコンポーネントウェア，分散オブジェクトサービスのフレームワーク技術を発展させることで，信頼性，柔軟性，安全性の高いアプリケーション実行，より生産性の高い開発を提供する．

.NETは図-1にみられるように実行環境としての.NETフレームワークとそのクラスライブラリ，アプリケーションサーバ群，ファンデーションサービス群と，開発環境としてのVisual Studio .NETから構成される．従来のコンポーネントウェアを踏襲し，複数開発言語で開発したソフトウェア部品を再利用し，各種サーバアプリケーションと連携して短期間のアプリケーション開発を可能としている．以下では，実行環境と開発環境に分けて概要を説明する．

■.NET実行環境

.NETフレームワークは開発言語非依存な中間言語，MSIL（Microsoft Intermediate Language）をベースとした仮想マシン.NETランタイムを介して，.NETコンポーネントを動作させる特徴を持つ．仮想マシンを利用して動作する他のプラットフォーム技術としてJavaが有名であるが，Javaと異なり，.NETコンポーネントは実行時はすべて.NETランタイムが動作するマシンのネイティブコードで実行される．すなわち，MSILは実行時にコンパイルされてネイティブコードに変換されてから実行される．これにより，中間言語によるタイプセ

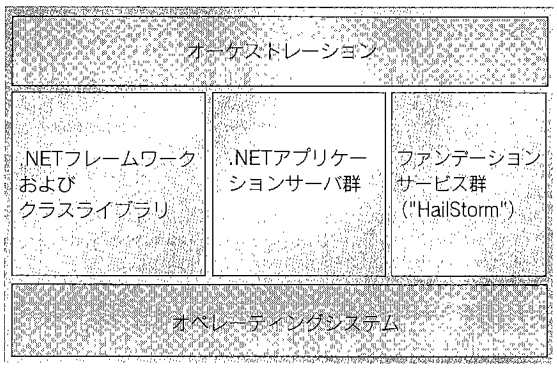

図-1　.NETの構成

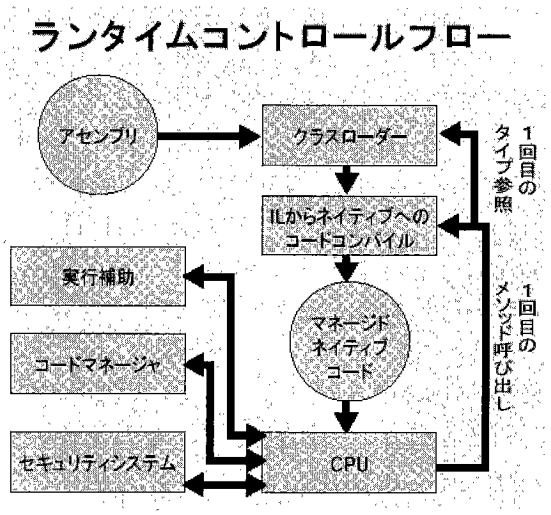

図-2　.NETコンポーネント（アセンブリ）の実行

ーフ，ランタイムサービスを享受しながら，マシンコードレベルの実行性能を得ることが可能となる．**図-2**は.NETコンポーネントが実行時に中間言語を介してネイティブコードに変換されて実行される様子を示す．

また，.NETコンポーネントは一般には汎用クラスライブラリを再利用して開発する．利用クラスを実行時にクラスライブラリから動的にロードする．この点はJavaの仮想マシンと同じ原理である．Javaと最も異なる点は，Common Type Systemと呼ばれる開発言語非依存なタイプシステムの導入により，複数開発言語間で継承，データ授受や型変換，可視性やアクセス制御，例外処理の伝播，メモリ管理の統一，動的バインディング，リフレクションなどが可能となる点である．Common Type Systemは種々の既存開発言語のタイプシステムを調査し，仮想マシンとランタイムサービスを実行する上で必要とされる機能，拡張性などを考慮して設計された．

一般に現在の企業システムでは，フロントエンドは生産性が高く，文字列操作などで有利なスクリプト言語を用い，中間層は業務オブジェクトや業務プロセスルールの記述能力の高く，状態を保持可能な業務向きオブジェクト指向言語を用いている．データアクセス層，中間層キャッシュ，並列ワークフロー処理，コンテキスト管理などの高度で高性能が要求される部位にはC++などの性能と記述能力の高い開発言語が利用される．このように開発スタイル，開発プロジェクトメンバの開発スキルに合わせた開発言語の選択は，現実的に非常に重要である．

さらに，最近のWebサービスなどソフトウェアサービスの開発プロジェクトでは，オブジェクト指向の再利用性を活用した生産性の高い開発と，フレームワーク技術による高度な分散サービスの利用に加えて，XMLを隠蔽しつつ柔軟性，変更に対する強度を持つ疎結合設計とそれを支える新しいアーキテクチャが必要である．

XMLはデータモデル，コンテンツモデル，データ転送構文として利用するが，アプリケーション開発ではXMLアクセスAPIを隠蔽した，より抽象度の高いクラスライブラリと，そのクラスが提供するオブジェクトモデルに基づいたプログラミングモデルを利用し開発を行う．しかし，プログラミングモデルの利用は適切な疎結合設計と組み合わせて行う必要があり，この点では新たなWebサービスに適するデザインパターンやデザインガイドの提供が必要となる．.NETフレームワークは，このようなデザインパターンの一部をランタイム機能，あるいは，.NETアプリケーションサーバ機能の一部として提供することで，開発者の作業を軽減する．

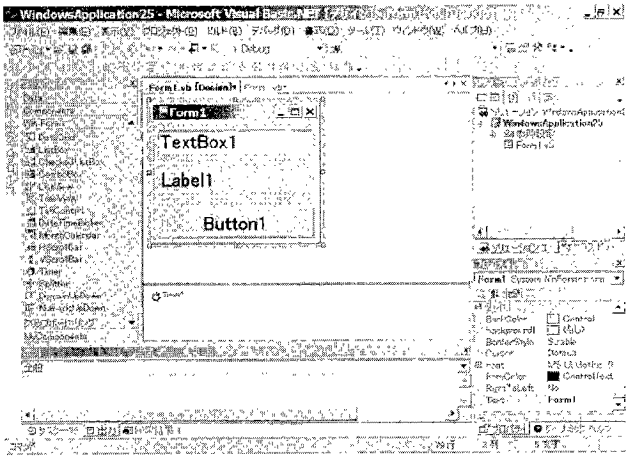

図-3　Visual Basic .NETによるWindowsアプリケーションの作成

■.NET開発環境

.NETアプリケーションやコンポーネントを開発するためには，.NETクラスライブラリの再利用，他の.NETやCOM+カスタムコンポーネントの再利用，Webサービスの発見と再利用などを効率的に行い，分散されたWebサービスや.NETコンポーネントのデバッグ，サービスやコンポーネントのバージョン管理などを行う必要がある．また，UMLによるオブジェクト設計，データベーススキーマ定義，XMLスキーマ定義，コンポーネントの署名，コンポーネント配布やセキュリティ設定などの支援機能の提供も必要である．

この.NET開発環境として，Visual Studio .NETが提供される．Visual Studio .NETは開発言語として，Visual Basic .NET，Visual C++ .NET，C#をサポートする．また，UML設計ツール，XML設計ツール，デバッガ，インストーラなどを統合している．Visual Studio .NETは従来のVisual Basicなどで多くの開発者が使い慣れているWizardによるプロジェクト作成，ポイントアンドクリックによるツールボックスの利用によるWebアプリケーションのGUI作成，オブジェクト参照によるXML Webサービスの再利用などが可能となっている．バージョン管理，セキュリティ管理，アプリケーション配布ツールも統合されている．そのため，従来の開発スタイルのままで，XML Webサービスやその他の高度な開発への対応が可能である．**図-3**はVisual Studio .NETの開発例を示している．

.NETにおけるWebサービスの実装

Webサービスを.NETで実装するには，Webサービス提供サーバでは，(1) Webサービスの属性ベースプログラミングを利用する方法，(2) .NETランタイムリモーティングクラスライブラリを利用する方法がある．その他に，従来のCOM+環境では，(3) SOAP Toolkit V2.0を利用する方法，(4) Windows XPで提供されるCOM+1.5のWebサービス機能を利用する方法，がある．

Webサービスを利用するクライアントの実装は，(1) WebサービスをVisual Studio .NETプロジェクト内で参照して利用する方法，(2) WSDLからC#のソースコードを生成しプロキシを作成して.NETランタイムリモーティングの起動メカニズムを利用する方法，(3) SOAP Toolkitで実行時にWSDLを読み取り，動的にプロキシを作成して利用する方法，(4) COM+1.5のWebサービスクライアント機能（SOAPモニカー）を利用する方法，がある．このうち，最も容易で一般的Webサービスのサーバ実装方法はWebサービスの属性ベースプログラミングを利用するものである．**図-4**は，Webサービスの属性ベースプログラミングのコード例である．

.NETにおける属性ベースプログラミングは，.NET対応の開発言語のクラス，メソッドやプロパティなどクラスメンバ，メソッドの引数などに，[...]で示される標準あるいは開発者がカスタムに定義したキーワードの属性を指定することで，クラスやメソッドの実行時にキーワードが定義する実行コード（とメタデータ）が再利用される技術である．また，このWebサービスの実装コードに対して，他のカスタム属性として従来のトランザクション属性やセキュリティ属性を追加することが可能であるので，Webサービスで要求されるメソッド実行がトランザクションスコープで実行され，実行クライアントのロール（役割）ベースセキュリティによるアクセス制御が可能となる．

```
[Help("http://SomeUrl/SomeClass")]
public class SomeClass{

   [WebMethod]
   [SoapMethod(MessageStyle=SoapMessageStyle.Rpc)]
   [TraceExtension()]
   public void SomeOldMethod()
   { ... }
   [Obsolete("Use SomeNewMethod instead")]
   public string Test( [SomeAttr] string param1 )
   { ... }
}
```

図-4　Webサービスの属性ベースプログラミングの例

XML対応

従来のXMLを利用したアプリケーション開発は，XMLパーサを利用したXMLドキュメントの解析と，ドキュメントのコンテンツに応じてアプリケーションを実行するための，DOM (Document Object Model) やSAX (Simple API for XML) といったXML APIの利用を主とした．このような開発では，XMLドキュメントの構文を意識したAPIの呼び出しを開発コードが持つため，XML構文に依存した開発コードになりやすく，XML本来の持つ柔軟性の高いドキュメント定義の変更機能に開発コードが対応しきれていなかった．

しかし，XMLスキーマ定義言語が標準化され，XMLスキーマ定義をオブジェクト指向のクラス定義と同等に扱うことで，データの処理からの分離，動的なオブジェクトの生成による開発コードの柔軟性，スキーマ定義の再利用と実行時の型識別による多態的処理，タイプセーフな処理による実行時エラーの減少などが可能となった．XMLスキーマ定義言語の標準化に合わせて，開発環境と実行環境は，上記のメリットを提供すべく多くの改善が現在も行われている．たとえば，.NETのXML対応ではクラスライブラリとして，XMLスキーマの作成，読み込みと検証，DOM/SAX APIを隠蔽したコンテナやナビゲータ，XPathによるクエリーによる，スキーマ定義，XMLドキュメントやRDBデータモデルへの操作を可能としている．また，SOAPのXMLプロトコルメッセージやXMLデータベースへの操作も明示的なXML構文を意識したプログラミングなしで開発が可能となっている．

このようなXMLの対応機能の中で，最も.NETが従来の技術から飛躍している機能がADO .NETである．ADO .NETはデータアクセスオブジェクトの一種である．その点では従来のADO (ActiveX Data Object)，あるいは，Java APIのJDO (Java Data Objects) やEJB (Enterprise Java Beans) のエンティティビーンと類似する機能を持つ．しかし，それらと根本的に異なる点は，ADO .NET自体がXMLに基づく論理的な木構造データモデルを持つ点である．

従来のデータアクセスオブジェクトはデータモデルを持たず，標準インタフェース定義とオブジェクトモ

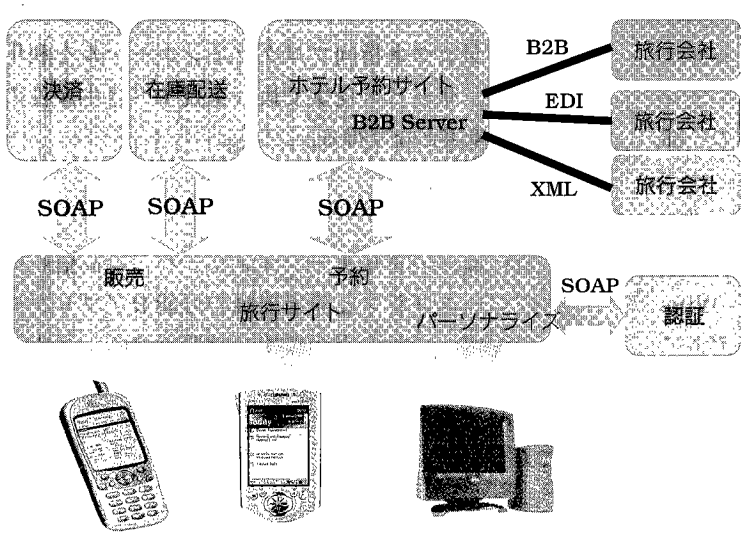

図-5　複数のWebサービス提供者の集約者としてのサービス代行者（旅行サイト）

デルのみを持っていた．そのため，アクセスされるデータソースのデータモデルからオブジェクトモデルへの構造変換が必要で，複雑なアクセスナビゲーションが伴うと同時に，データソースのデータモデルを忠実に表現できない欠点を持っていた．たとえば，従来のADO Recordsetオブジェクトで階層型のデータモデルを表現するためには特殊なプログラミングをアクセスクライアントが行う必要があった．その結果，アクセスクライアントはデータソースのデータモデルにかなり依存した開発コードとなり疎結合設計という観点では望ましいコードではなかった．

ADO .NETの第2の利点は，共通データモデルの提供により，RDBの関係データモデルとXMLデータモデルを統一的に扱うことが可能な点である．さらに，第3の利点はRDBの複数テーブル間の制約関係を位置に非依存にオブジェクト内で扱うことが可能な点である．第4の利点は，このデータモデルをフレームワークの各階層間で転送可能であり，データソースに対して楽観的ロックに基づくトランザクション処理が可能，という点である．もちろん，従来のようにフレームワークの各階層間で，XML文字列，XML DOM形式，値渡しの永続化オブジェクト，コレクションオブジェクトによる転送，オブジェクト参照渡しは可能であるが，ADO .NETはこの中では最も優れたデータ転送，キャッシュ技術として.NETで利用されている．

Webサービスのファンデーションサービス：HailStorm

Webサービスの本格的普及には，SOAPのRPCバインドのようなクライアントからサービスへの単純な要求と応答の連携に加え，複数のWebサービスが複雑な連携をするシナリオをサポートするプラットフォームの提供が必要である．たとえば，図-5のようにWebサービスのクライアントが複数のWebサービス提供者の集約者としてのサービス代行者にアクセスし，このサービス代行者がバックエンドの複数Webサービス提供者にアクセスし，複合的で高度なサービスをクライアントに提供するシナリオがある．このようなシナリオを可能とするには，Webサービスの集約者がクライアントから権限の委譲を受けて，クライアントの権限でバックエンドのWebサービス提供者にアクセスする必要がある．このため，マルチホップのセキュリティ委譲のプロトコルスキームがプラットフォームに求められる．クライアントは特定サービス提供者に対しては，権限の委譲や個人識別情報の提供を許すが，他のサービス提供者には，より制限をつけたアクセスだけを許す制御が求められる．

従来のイントラネット，あるいは，マルチドメインのエクストラネットではKerberosのようなセキュリティプロトコルスキームがあったが，Webサービスを提供する非集中型システム間連携のインターネットではKerberosの適用は適切ではない．このようなサービスインフラの支援機能は他にも多く存在する．たとえば，UDDIはサービス提供者のサービス定義，アクセスポイントなどサービスレベル契約に関する情報は提供するが，すでに提供中のサービス状態など動的な情報の提供はしない．しかし，たとえば，利用中のサービス状態に応じてサービスを利用したいアプリケーションは，UDDI以外にこの動的情報を必要とする．

Webサービスは現在，SOAP，WSDL，UDDIなどのサービス記述階層における低レベルな基本技術の提供が開始されたばかりである．W3CのXML関連技術の標準，インターネット標準，OASISなどの業界標準ではこれ以外に，SOAPのルーティング機能，信頼性の高いメッセージング機能，複数サービスポート間の連携とフロー記述定義，サービスレベル契約やTPA（Trading Partner Agreement）定義，UDDIのサービスタクソノミ

ーなどが検討されている．これらの標準化と合わせて，より上位のサービスアプリケーションを支援するプラットフォームの提供が必要であり，.NETファンデーションサービス（コード名"HailStorm"）がこの役割の一部を担おうとしている．

現在提供が予定されている.NETファンデーションサービスには，passportによる複数Webサービス間の共通認証サービスと個人識別情報の再利用技術，Windows messengerによるサービス通知，マルチメディアデータ転送，プレゼンス，カレンダーによる予定管理，アドレスによる連絡先管理，ロケーションによる地理情報に依存するサービス提供などがある．こういったサービスインフラの支援機能はB2Cを中心として発展し，第2段階としてB2B分野にも拡張される予定である．

まとめ：.NETとサービス指向アーキテクチャ（SOA）の今後

.NETは開発言語としてVisual Basic .NET，C#などオブジェクト指向言語とクラスライブラリの採用，また，.NETランタイムがJava仮想マシンと類似の機能を提供するため，従来のオブジェクト指向フレームワークの実行環境と大差ないと認識されている．しかし，.NETは以下の点でサービス指向アーキテクチャと考えることができる．

(1) OSに代わる実行環境としてのアプリケーションドメインとプロセスリサイクル
(2) 非Webサーバ，非アプリケーションサーバによるサービス実行
(3) メタデータによるコンポーネントバージョン管理とバインディングポリシー制御
(4) SOAP，WSDLのネイティブサポート
(5) サービス利用のリース管理

SOAP通信がWebサーバを前提とし，コンポーネント実行がアプリケーションサーバに限定され，SOAPやWSDLが追加ライブラリで提供されるこれまでのフレームワーク技術は，サービスよりも特定のコンポーネントモデルを前提としたオブジェクトを単位とした実行と，そのオブジェクト間連携を前提として設計されたフレームワークであるので，サービスの実行は必ずしも効率的ではない．また，サービス提供のために特別なデザインパターンの考慮が必要であるとか，特定のオブジェクト指向プログラミングモデルを利用する開発は，サービス提供で最も重視すべき生産性の高い開発の障害となり得る．

たとえば，並列処理ワークフローが並列実行インスタンス相互の信頼性を確保しつつ高性能な実行を制御するには，アプリケーションサーバには高信頼なワークフローエンジンと障害隔離の可能なアプリケーションコンポーネント実行環境が必要である．従来の実行環境は複数サーバマシン間で障害隔離のためのクラスタ技術を導入することで対応してきたが，最新の.NETフレームワーク実行環境では，上記のアプリケーションドメイン，Web gardenのようなマルチインスタンスサーバ，アプリケーションキャッシュとリサイクルなどをクラスタと併用するフレームワーク技術を導入している．

アプリケーションドメインはオペレーティングシステムのプロセスモデルに代わり，中間言語とタイプセーフを利用したより安価な実行環境であり，これまで商用Javaでは提供できていない．こういった実行環境が，Webサーバ，データベース，アプリケーションサーバ，OSシェル，ブラウザ，オフィスアプリケーションに透過に提供され，いずれもSOAPなどのWebサービス機能がネイティブに利用できるのが.NETの特長である．提供サービスのQoS（Quality Of Service），セキュリティ信頼モデルが適切に管理され，クライアントごとの課金レベルに応じて利用リソースに制約を設けるには，従来のOS／ファイルシステム，データアクセスコンポーネント／アダプタなどのリソース管理では不十分であり，.NETのようなサービス指向を考慮し，オブジェクト指向とバランスをとったプラットフォーム機能の提供をしなければならない．

Webサービスはまだ発展途上にあり，多くの技術進化が今後も継続するであろうが，.NETはその中で常に1つの“参照”実装として位置づけられると思われる．

参考文献
1) Microsoft .NET完全理解，アスキームック（2000）．
2) Microsoft .NET完全入門，アスキームック，MSDN Magazine日本語版別冊，アスキー（2001）．
3) C#で学ぶ.NETプログラミング，Windowsプログラミング特別編集，技術評論社（2001）．
4) MSDN Library, http://msdn.microsoft.com/library/default.asp

（平成13年8月8日受付）