.NETの本来の姿と現状，そして日本ユニシスの.NET

Goal and Current Status of .NET and Our Strategy

山 岸 重 雄

要 約 .NETのコンセプトが発表されてから4年半が経過し，今やJ2EEに匹敵するシステムのプラットフォームとなった．しかし，これは.NET本来の姿ではない．.NETはマイクロソフト社が考えるソフトウェアの未来像であり，「ソフトウェアをサービスとして利用できるようにすること」を目指している．現時点での.NETは，.NET FrameworkとVisual Studio .NETをベースとしたシステム開発環境＆実行環境である．

そして，日本ユニシスが考える.NETとは，ミッションクリティカルシステムのプラットフォームである．その実現のために，MIDMOST for .NET, ACAB, LUCINA for .NETを提供する．

Abstract Four and a half years have passed since the announcement of .NET concept, which has become comparable to J2EE now. However, this is not a .NET of what it should be. .NET is the Microsoft 's future of the software and aims to facilitate the software to be used as service.

Current status of .NET is the system development environment and execution environment based on .NET Framework and Visual Studio.NET.

And Nihon Unisys thinks .NET is a platform on which of a mission critical system is running. For the realization of our philosophy, we offer MIDMOST for .NET, ACAB（Advanced Cluster Application Builder), and LUCINA for .NET.

1. はじめに

マイクロソフト社が2000年7月に.NETのコンセプトを発表してから4年半が経過した．

2002年4月には，.NETのコンセプトを具現化するための製品(.NET FrameworkとVisual Studio .NET）がリリースされ，システムの開発基盤，実行基盤としてJ2EE（Java 2 Enterprise Edition）に匹敵するものとなった．ガートナー社も，「次世代戦略基盤を構築するための最も現実的な選択肢はメインフレーム，Javaプラットフォーム，.NETプラットフォームの3種類に絞られ，2005年にはメインフレーム：30%，Java : 35%，.NET : 35%になる」と予測している．

しかし，こうした「Java勢力に対抗するためマイクロソフト社が提供したシステムの開発基盤および実行基盤」という姿は，.NETが真に目指しているところではない．

.NETが真に目指しているところは何なのか．

それは，「ソフトウェアをサービスとして利用できるようにすること」である．

マイクロソフト社は.NETを次のように定義している．

“.NET is the Microsoft solution for Web services, the next generation of software that connects our world of information, devices, and people in a unified, personalized way.”

(.NETとは，あらゆる情報やデバイス，そして人を，統一され個別化された方法でつなぐ

次世代ソフトウェアである Web サービスに関するマイクロソフトのソリューションである.）

.NET は，マイクロソフト社が考えるソフトウェアの未来像である.

本稿は，.NET の本来の姿と現状を再認識し，システム開発の現場にもたらした意義を考察し，日本ユニシスが考える.NET を紹介するものである.

2.　.NET 概論

2.1　.NET の本来の姿

マイクロソフト社の設立時のビジョンは，

A computer on every desk and in every home
（全ての机に，家庭に，コンピュータを）

であった.

このビジョンを実現するべく，マイクロソフト社は Windows や Office 製品を提供し，パーソナルコンピュータ（以下 PC と記す）を誰でもが容易に使えるように変革した．そして，ビジョン通り，一家に一台，一人一台コンピュータを保有する時代になった.

ビジョンを実現してしまったので，新しくビジョンを書き換え，2000 年 7 月に次の新ビジョンを発表した.

Empower people through great software, any time, any place, and on any device.
（時や場所，デバイスを問わず，優れたソフトウェアで人々の可能性を広げる.）

この時，このビジョンを実現するために同時に発表されたのが，.NET というコンセプト「あらゆる情報やデバイス，人をつなぐ次世代ソフトウェア」である.

.NET は，ソフトウェアの未来像を示したものであり，ソフトウェアの在り方を「コンピュータ上で稼働するプログラム」から「インターネットをプラットフォームとするサービス」へと変革するものである.

.NET は，当初「NGWS（Next Generation Windows Service）」という名前で発表される予定であった．しかし，マイクロソフト社内で「自分たちがやりたいことは，『Windows』のさらなる拡販ではない．Windows だけに閉じた世界ではなく，インターネット上に開かれた新しいプログラミング・モデルを再構築することだ．これまでにはなかったようなデバイス連携を実現することだ．はたしてこのようなビジョンに『Windows』という名前をつけるのはふさわしいだろうか」という声が沸き起こり，.NET という名前が採用された経緯がある.

このことからも，.NET は，決して Windows 上でのシステムの開発基盤，実行基盤という位置付けのものではないし，ある特定のマイクロソフト製品で提供される機能ではない，ということがわかる．極端なことを言えば，J 2 EE で.NET のコンセプトを実現することも可能なのである.

.NET が特定の製品で提供される機能ではないことを明確にするため，マイクロソフト社はいくつかの製品名に付けていた.NET という単語を外す方針に変更した．当初「Windows .NET Server」という名前で発表される予定であった「Windows Server 2003」を始め，「.NET Enterprise Servers」と呼んでいたサーバ製品群も「Windows Server System」に変更した．「Visual Studio」も次バージョンから.NET がはずれ「Visual Studio 2005」となり，「.NET Framework」も次期 OS である Longhorn がリリースされるタイミングで，「WinFX」という名称に変更される.

現在，Visual Studio.NET を使用して，.NET Framework 上で稼働する Web アプリケーションを開発した場合，「.NET で開発した」と見なされるが，本来の.NET の定義からすると，実はこれは誤りなのである．

2.2 .NET 実現に向けて

では，.NET を実現した次世代ソフトウェアとは，どんなものなのか．

まず，「インターネットをプラットフォームとするサービス」 の例として 「.NET Passport」を紹介する．

インターネットのショッピングサイトで買い物をする場合，サイト毎に氏名，住所，電話番号，メールアドレスなどの情報を入力する必要がある．そして，各サイトのユーザ ID とパスワードを覚えておかなければならない．引越で住所や電話番号が変わった場合は，登録した全サイトの変更処理をしなければならない．

「.NET Passport」はマイクロソフトが提供している個人情報を一元的に管理するサービスで，サイト開発者は，このサービスを利用することで，利用者に他のサイトで入力したのと同じ情報を入力させたり，いくつものユーザ ID とパスワードを憶えさせる労苦から解放させることができる．

別の例を紹介する．あるデバイス上でスプレッドシートのデータを計算して，グラフを描画する場合，各デバイス上で計算とグラフ描画を行わなければならない．これが将来の.NET では，デバイス同士，ソフトウェア同士が能動的に連携し，すべての処理を自分で行わなくても，周囲にある計算機資源を必要に応じて活用できるようになる．たとえばグラフを携帯電話機に表示させる場合，計算やグラフ描画は携帯電話で実行しなくても，周りにある処理性能の高い PC がこれに代わって処理を行う．そしてグラフが作成できたら，その結果のイメージだけを携帯電話に転送する．この時に重要なことは，こうしたソフトウェア連携を，人間が介することなく，ソフトウェアの判断で行えるようになることである．

.NET のコンセプトを実現するためには，次の 3 点が重要となる．

① サービス指向アーキテクチャ
② 異機種プラットフォームとの相互運用性
③ ユビキタスとエクスペリエンス

以下，この 3 点について解説していく．

2.2.1 サービス指向アーキテクチャ

.NET が目指している「サービスとして利用できるソフトウェア」は，サービス指向アーキテクチャ（Service Oriented Architecture．以下 SOA と記す．）が目指しているところと重なる．

SOA[*1] の厳密な定義はないが，本稿では牧野[3]による以下の定義を採用する．

SOA では，従来，機能単位で切り出すことが不可能であった一枚岩的なソフトウェアを，「サービスを提供する部品」として捉え，再利用できるように意識して設計・開発していく．ソフトウェアの再利用については，「ライブラリ」や「コンポーネント」といった粒度の小さい単位では現在でも実践されているが，SOA での「サービス」の単位は，従来のサブシステムの単位に相当する粒度の大きい再利用性を目指している．

.NET では，SOA を実現するためには，以下の基盤技術が必要となると考えている．

① サービス作成

Web サービスの設計，開発，組み立て，管理，配置，およびテストに必要なツール，プロセス，方法論，およびパターン

② サービス配信

あらゆるタイプのデバイスのサポートを含め，プレゼンテーションの問題とテクノロジに焦点を当てたポータルおよびクライアントサービス

③ サービス統合

サービスと現在の運用システム，すなわちレガシーアプリケーション，商用アプリケーション，データベース，他の Web サービスとの間の統合と相互運用の提供

④ サービス　フレームワーク

Web サービスベースのアプリケーションが必要とするプロセス，ロジック，機能，および状態管理を含み，Web サービス用の具体的なサポートを提供する完全なエンタープライズアプリケーションサーバ

⑤ サービスプラットフォーム

OS，ハードウェア，ストレージ，ネットワーキング，およびシステム全体のための信頼および管理サービス

.NET 実現のためには，こうした SOA の基盤技術を整備する必要がある．

2.2.2 異機種プラットフォームとの相互運用性

サービスは Windows 上だけで提供されるものではない．マイクロソフト社は従来，プロプライエタリな技術を重視し，業界標準技術の採用には消極的であった．しかし，.NET ではこうした姿勢を改め，異機種プラットフォームとの相互運用性を重要視しており，TCP/IP,XML, SOAP といった業界標準の技術を積極的に採用している．また，こうした通信プロトコルだけでなく，Word や Excel といった Office 製品のドキュメントも XML 形式で保存できるようにしている．これにより，他のアプリケーションプログラムとの相互運用性を実現している．

異機種プラットフォームとの相互運用性を実現するための最有力候補が XML　Web サービスである．XML Web サービスは，XML を用いて RPC（Remote Procedure Call）形式でのアプリケーションプログラム間連係を実現する技術である．ベンダや業界，業種の壁を越えた標準化の取り組みが行われており，標準化団体である WS-I（Web Services Interoperability Organization）には，IT 業界の著名な企業や団体はほぼすべて参加している．

しかし，まだ発展途上の段階であり，仕様の詳細部分の解釈の違いなどから，現時点に於いては，プラットフォーム間での接続を確実に保証できるという状況に至ってない．トランザクションの扱いについても仕様が確定していない．

.NET 実現のためには，こうした異機種プラットフォームとの相互運用性の基盤技術を整備する必要がある．

2.2.3 ユビキタスとエクスペリエンス

「時や場所，デバイスを問わず」「あらゆる情報やデバイス，人をつなぐ」というビジョンが示すように，ユーザの使用する機器は PC だけとは限らない．携帯電話，PDA（Personal Digi-

tal Assistance），情報家電，ゲーム機器，カーナビ，ウェアラブルデバイスなど，あらゆる機器に対応し，「いつでも，どこでも」のユビキタス・コンピューティングの実現を目指している．ユーザインタフェースもディスプレイやキーボードだけでなく，音声や温度などの新しいインタフェースを提供していくことを目指す．このことをエクスペリエンス（体験や驚き）と称している．

.NET 実現のためには，こうしたユビキタスとエクスペリエンスの基盤技術を整備する必要がある．

2.3 .NET の現状

前節で.NET の本来の姿を説明したが，本節では.NET の現状について説明する．

.NET のコンセプトを具現化する製品群とテクノロジを総称して「.NET プラットフォーム」と呼ぶ．「.NET プラットフォーム」は以下の四つの要素から構成される．

① .NET Framework
② Visual Studio.NET
③ ビルディングブロックサービス
④ Windows Server System

.NET プラットフォームの概念を図 1 に示す．

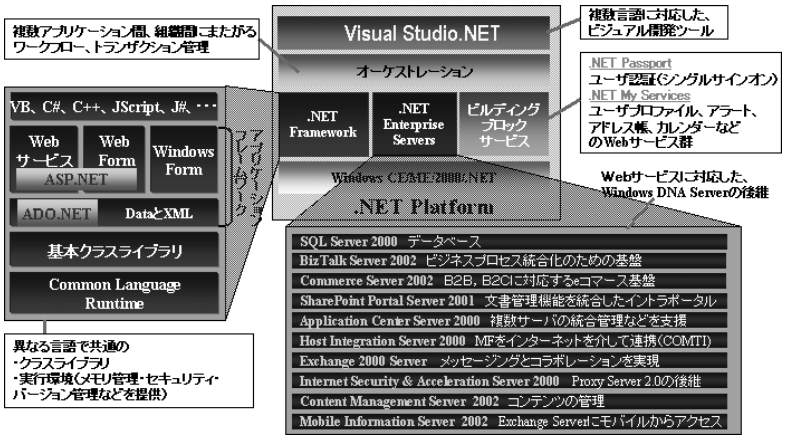

図 1 .NET プラットフォーム

2.3.1 .NET Framework

.NET Framework は，.NET プラットフォームで実行される全てのプログラムの実行環境および共通ライブラリを提供する．

図 1 に示すように，.NET Framework は，共通言語ランタイム（Common Language Runtime．以下 CLR と記す．），基本クラスライブラリ，アプリケーションフレームワークから構成される．

.NET Framework は，現在，PC の Windows OS 上で稼働するものと，携帯電話や PDA，組み込み機器など，小型の情報機器のための縮小版の.NET Compact Framework がある．

1） 共通言語ランタイム（CLR）

.NET Framework 以前は，Windows 上でのアプリケーションプログラムの開発は，VB,C,C++，COBOL,Java などの開発言語を使用した．これらの言語で開発したアプリケーションプログラムは，コンパイルされ機械語または独自の中間言語に変換される．各言語は独自のランタイムを持ち，その上で実行される．ランタイムが異なるので，同じ処理をするのに実装方法が異なったり，言語によっては実現できない処理もあった．

.NET Framework では，すべての言語が同じ中間言語（Microsoft Intermediate Language, MSIL）にコンパイルされ，CLR 上で実行される．すべての言語が同等に扱われるので，開発プロジェクトの要員が得意な言語で開発することができる．異なる言語間での部品の再利用（クラスの継承・メソッド呼び出し）も可能である．現在，.NET Framework に対応している言語は，C#,VB.NET,C++，Java,COBOL など 20 種を超える．

C# は C++ をベースとしたオブジェクト指向言語で，Java の影響を強く受けているとともに，Java の弱点を補う言語仕様となっている．Java がアカデミックな理想の言語を目指しているのに対し，C# は実行効率を重視した現実的な仕様となっている．

また，CLR には以下のようなアプリケーションプログラムを安全に実行させるための機能が備わっている．

・ガベージコレクション

不要となったインスタンスを自動的に破棄するメモリ管理機能．開発者が不要となったメモリを開放する必要がなく，従来不具合の原因となっていたメモリリークがなくなる．

・コードベースセキュリティ

アプリケーションプログラムのコードそのものに対して何を実行して良いのかを設定することができる．従来，プロセスの実行アカウントや操作者のユーザアカウントなど，アカウントベースでアプリケーションプログラムの実行権限を判断していたため，高い権限をもったアカウント上であれば，どんな不審なプログラムでも実行できてしまった．コードベースセキュリティでは，たとえ Administrator などの高い権限をもったアカウントであっても，許可されていないアプリケーションプログラムは実行を制限することができる．

・グローバルアセンブリキャッシュ

名前が同じライブラリファイルであっても，バージョンが異なれば違うものとして管理することができる．従来，他のアプリケーションプログラムをインストールしたところ，共通で使用していたライブラリが上書きされたため，動作がおかしくなってしまうという現象が多々発生していた．CLR は，ファイル名が同じであっても，バージョンが異なれば違うものとして扱うので，複数バージョンの共存が可能となっている．.NET Framework 自身のバージョンも複数共存させることが可能である．

2)　基本クラスライブラリ

アプリケーションプログラムでは，メモリの確保，ファイル入出力，データベースのアクセス，他のプロセスとの通信などが必要となるが，通常これらの機能は OS やミドルウェアが提供しているものを使う．Windows 上では，提供される機能によって Windows API と呼ばれている手続き型言語の関数の形態で提供されている場合と，COM と呼ばれているマイクロソフト社独自のコンポーネントの形態で提供されている場合がある．また，Windows API は，16 ビットの時代から OS のバージョンアップの度に API が無秩序に追

加され，系統立った形で整理されていない．こうしたことから，開発者は自分が必要とする機能を見つけるのに大変手間取った．

また，開発言語によっては，OS の機能をラップしたり，独自の機能を追加するなどした専用のランタイムを持ち，同じ処理をするのに，言語によって実装方法が異なる場合もある．言語によっては実現できない処理もあった．

.NET Framework の基本クラスライブラリは，アプリケーションプログラムの開発に必要となる機能を，系統立てて整理したものであり，こうした問題点を解決している．

3） アプリケーションフレームワーク

アプリケーションフレームワークとは，業務に非依存なアプリケーションプログラムの枠組みであり,ユーザコードを呼び出すオブジェクト指向的なクラス・ライブラリである．.NET Framework では，Web アプリケーション開発用の業務非依存アプリケーションフレームワーク ASP.NET を提供している．

ASP.NET には，以下の特長がある．

・Web アプリケーションの開発を VB ライクなイベントベースのプログラミングモデルで開発することを可能とした．
・アプリケーションロジックとプレゼンテーションの分離を可能とした．
・クライアント側に合わせて HTML を自動的に生成するサーバコントロールを提供している．携帯電話などのモバイル端末にも対応している．
・複数台の Web サーバによるファーム構成でのロードバランスに対応したセッション管理機能を提供している．
・入力データチェックの設計，実装負荷を大幅に軽減する検証コントロールを提供している．
・データベースアプリケーション開発を大幅に効率化するデータバインド機能を提供している．

データベースのアクセスは，高スケーラビリティを実現するための非接続型データベースアクセスを特長とした ADO.NET を提供しており，ASP.NET のデータバインド機能と連係して，Web アプリケーション開発の高生産性を実現している．

2.3.2 Visual Studio.NET

Visual Studio .NET は，.NET framework 環境において XML Web サービスおよび Web または Windows アプリケーションを迅速に構築し統合するための包括的な開発環境である．

従来，開発言語毎に別々であった開発環境を統合し，複数言語によって開発したアプリケーションプログラムのデバッグも，ひとつの開発環境のもとで行える．

グラフィカルなユーザインタフェースでソースコードを自動生成し，Web アプリケーションの短期開発を可能にしている．

また，ソースコードの生成，リバースエンジニアリングが可能な UML 表記法によるモデリングツール Visio や，Web アプリケーションの負荷テストツール Application Center Test，ソースコード管理ツール Visual SourceSafe など，開発に必要十分なツールが含まれている．

2.3.3 ビルディングブロックサービス

ビルディングブロックサービスはマイクロソフト社が提供する Web サービスであり，現在，.NET Passport と.NET Alerts の二つのサービスが提供されている．

.NET Passport は，ユーザ情報を一元的に管理するサービスで，ユーザはいろいろなサイトで何度も同じ情報を入力する手間を省くことが実現でき，サイト毎にユーザアカウントとパスワードを憶えておく必要もなくなる．

.NET Alerts は，ユーザがあらかじめ登録しておいた設定に従って，企業が配信する通知を，いつでも，どこでも，どのようなデバイスでも受け取ることを可能にする．通知の配信先は，ユーザのインターネット利用状況に合わせて自動的に変更されるように設定でき，オンラインの時にはデスクトップ PC 画面へのポップアップで，オフラインの時には電子メールで通知を受け取るなど，ユーザは知りたい情報を，知りたい時に知ることができる．

これらは，「インターネットをプラットフォームとするサービス」の実例である．

2.3.4 Windows Server System

各種業務要件の実現のためにマイクロソフト社から提供されているサーバ製品群が，Windows Server System である．

例えば，消費者向けの Web ベースのシステムの場合，Commerce Server, SQL Server によって中核が構築される．そのシステムのスケーラビリティと可用性は複数のサーバに分散し Application Center によって管理される．このシステムにエンドユーザがアクセスする際には Internet Security & Acceleration Server を経由し，高いセキュリティとパフォーマンスを維持する．

エンドユーザからの注文をサプライヤに発注するためには BizTalk Server を利用し XML ベースでシステム連携が行われる．

また在庫管理がバックエンドシステムであるメインフレームで行われていれば，Host Inte-

表 1　Windows Server System の製品群

名称	機能概要
SQL Server	構造化された XML データの格納，検索，および分析 （データベース）
BizTalk Server	アプリケーションと組織間で XML ベースのビジネス プロセスを構築
Share Point Portal Server	ビジネス情報の検索/共有/発行が可能 ビジュアルツールによるポータルサイトの作成
Commerce Server	Ｂ２Ｂ，Ｂ２Ｃに対応するｅコマース基盤 スケーラブルな電子商取引ｿﾘｭｰｼｮﾝの短期構築を実現
Application Center Server	高可用性でスケーラビリティの高い Web アプリケーションをバージョニング・導入管理
Content Management Server	動的な e ビジネス Web サイトのコンテンツを管理
Host Integration Server	レガシーシステム上のデータとアプリケーションへのブリッジ を提供 （COMTI）
Exchange Server	メッセージングとコラボレーションを実現
Internet Security & Acceleration Server	安全で高速なインターネット接続を提供 Proxy Server 2.0 の後継
Mobile Information Server	アプリケーションが携帯電話などのモバイル デバイスをサポートできるようにする （Exchange との連携も有り）

gration Server あるいは BizTalk Server によって在庫管理処理をメインフレームに受け渡す．

ユーザへの通知や，社内でのワークフローの処理には Exchange Server が使われる．

3. .NET の意義

現時点での.NET プラットフォームは，本来の.NET 構想を実現するまでには至ってないが，Windows 上でのシステム開発に大きな変化をもたらした．本章では，この変化について説明し，現時点での.NET の意義について考察する．

3.1 Web アプリケーションの短期開発

アプリケーションフレームワーク ASP.NET を内包した.NET Framework と Visual Studio .NET の組み合わせにより Web アプリケーションを短期間で開発できるようになった．

実際に，同じ仕様の Web アプリケーションを J 2 EE ベースと.NET ベースで開発してみたところ，.NET ベースの方がコーディング量は 1/2，実装工程以降の工数は 2/5 という結果であった．

実案件での実績値においても，.NET ベースの開発での物理設計以降の 1 人月あたりのファンクションポイント値が Java ベースのものの 1.4〜1.9 倍，.NET 以前のマイクロソフトテクノロジである VB＋ASP ベースのものの 2〜3 倍となっている．

.NET ベースの開発生産性がいかに高いものであるかを証明している．

案件実績も，開発期間 2〜3 か月の超短期開発のシステム構築が増えている．

3.2 XML Web サービスの普及

.NET の出現により，XML Web サービスが身近なものになった．

Visual Studio.NET を使用することによって，WSDL や SOAP の文法を知らなくても，必要なコードが自動生成されるので，XML Web サービスを使うことの敷居が低くなった．

また，Visual Studio.NET に刺激されて，Java の開発ツールも XML Web サービスを容易に利用できるようになってきている．

現時点での国内事例では，企業間より企業内で利用しているケースが多いが，米国では企業間での利用が多く，日本でも今後は企業間での利用が増えてくるものと予想される．

3.3 クライアントサーバの復活

コンピュータのシステム形態の歴史は，集中と分散の間の往復運動である．

1970 年代までのメインフレーム全盛時代は，ホストコンピュータとダム端末による一極集中型であった．

80 年代に入り，Unix サーバと PC の高性能化，RDBMS の実用化，PowerBuilder,SQL Windows などの 2 層クライアントサーバシステム（以下 CSS と記す．）開発ツールの流行とメインフレームのソフトウェア変更に対する情報システム部門のバックログ増大などの理由から，Unix サーバを DB サーバとし，PC をクライアントとする 2 層 CSS が主流となった．分散の時代である．

90 年代に入り，インターネットの普及とともに Web のテクノロジが出現した．当初は情報の閲覧だけであった Web ブラウザも，アプリケーションのクライアントとして使用できるよ

うになった．ちょうどその頃，クライアントサーバ型では，クライアント側 PC にアプリケーションプログラムを導入する負荷が高く，総所有コスト（Total Cost of Ownership．以下 TCO と記す）が嵩むことが問題となっていた．

クライアント側にはアプリケーションを導入する必要がなく，Web ブラウザだけで良いという Web アプリケーションが主流になった．また，集中型に戻った．

しかし，Web ブラウザはユーザインタフェースが限られており，専用のクライアントアプリケーションに比べて操作性が劣る．すべての処理がサーバだけで行われるので，年々高性能になっているクライアント側 PC の性能を生かしていない，などの問題がある．

そこで，クライアントサーバの高 TCO 問題と Web アプリケーションの低操作性の問題を解決し，両者の良いところだけを取ったソリューションがスマートクライアントである．

スマートクライアントは .NET ベースの Windows アプリケーションであり，XML　Web サービスを使ってサーバ側のサービスを呼び出すものである．クライアント側 PC からサーバ側に保存されているクライアントアプリケーションの URL を呼び出すことで，アプリケーションをクライアント側 PC にダウンロードしてから起動する．既にアプリケーションがクライアント側 PC にダウンロードされている場合は，新たにダウンロードすることなく起動する．サーバ側に保存されているクライアントアプリケーションをバージョンアップした場合は，新たにダウンロードしてから起動する．

Web アプリケーションでは運用管理者の管理コストは軽減したが，エンドユーザの操作コストは増大している．Web アプリケーションでは実現できない高い操作性を実現するため，スマートクライアントによる新しい CSS が復活し始めている．1980 年代に流行した CSS は，クライアントアプリケーションから直接 RDBMS にアクセスする 2 層 CSS であったが，スマートクライアントを使用した CSS は，クライアントアプリケーション＋サーバアプリケーション＋RDBMS の 3 層 CSS となる（図 2）．

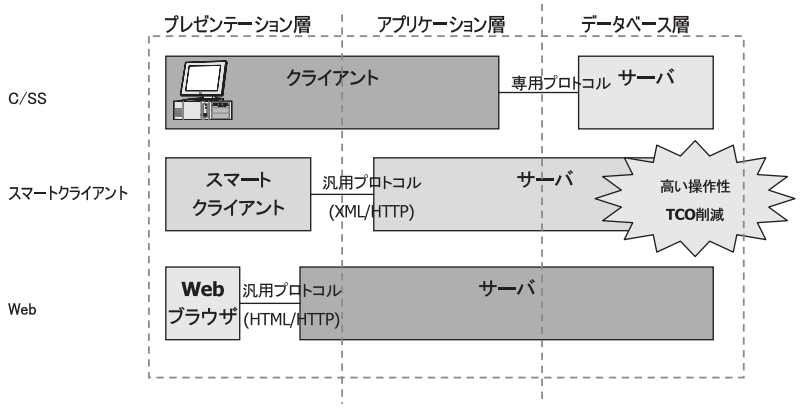

図 2　コンピュータのシステム形態

スマートクライアントを実現するためには，クライアント側 PC に.NET　Framework が導入されている必要がある． 現時点では，.NET Framework が導入されている PC は少ないが，Windows　OS の将来のバージョンでは.NET　Framework が標準で導入されるようになる．そうなれば一層，CSS が見直されるであろう．

3.4 DOA の復活

現時点では，ほとんどすべてのシステム開発で，データの保存に RDBMS（Relational Database Management System）を使用している．

しかし，J 2 EE ベースでのシステム開発では，オブジェクト指向分析，設計を行うため，RDBMS との相性が悪い．データベースのアクセスも EJB の Entity Beans や O/R（Object/Relational）マッパーを介して，あくまでオブジェクトとして処理しようとしている．

それとは対照的に.NET では，オブジェクト指向分析，設計にはこだわらない．.NET でデータベースに関する処理を受け持つ ADO.NET では， RDBMS の表をそのまま表として扱う．したがって，DOA（Data Oriented Approach）との相性が良い．

3.5 COBOL の復活

.NET Framework に対応した COBOL は，富士通製の Net COBOL for .NET とマイクロフォーカス製の Net Express with .NET，日立と NEC が共同開発した COBOL 2002 for .NET Framework がある．

従来，基幹系システム向けの主要開発言語として用いられてきた COBOL でも，XML Web サービスを提供したり利用したりすることが可能となっている．

現在，メインフレーム上で稼働しているアプリケーションをサービスとして提供する場合，.NET Framework に対応した COBOL にマイグレーションして，XML Web サービスとして公開することが可能であり，COBOL プログラムと COBOL 開発担当要員が活用できる．全世界で COBOL プログラマは 2 百万人いると言われており，これは Java プログラマの 2 倍以上である[5]．

3.6 VB 要員の活用

VB（Visual Basic）は，従来，クライアント側のアプリケーションを開発する言語として使用されてきた．VB.NET になり，サーバ側プログラムの開発言語としても使用に耐えうるものになった．

VB を使用する開発者は世界で最も多く存在すると言われている．したがって，.NET の開発環境で VB を使用する開発者を活用できることは有効である．

3.7 IT 技術の進歩への貢献

.NET が登場した事による変化をいくつか挙げたが，これらの事象から考察される現時点での.NET の意義は

「オープンシステムでのサーバアプリケーション構築環境として，J 2 EE 以外の選択肢も残った」

ということである．

もし，マイクロソフト社が.NET を発表せず，従来の製品のバージョンアップしか実施していなかったとしたら，サーバアプリケーションのプラットフォームは恐らく J 2 EE の独擅場となっていたことであろう．そして，XML Web サービスの普及も現在より遅れていたことであろう．

技術の進歩は善き競争相手が存在することでスピードアップする．マイクロソフト社と Java

陣営は，善き競争相手として，熾烈な戦いを繰り広げながらも，相手の技術の良い点は「良い」と認め，その弱点を補う新しい技術を創出し合っている．

.NET が存在することで J 2 EE の進歩があり，J 2 EE が存在することで.NET の進歩がある．

両者が存在していることで，IT の技術革新が急速に進んでいく．

4.　日本ユニシスが考える.NET

マイクロソフトの.NET は「サービスとして利用できるソフトウェア」を目指したものであるが，日本ユニシスにおける.NET とはどんなものであるか．

それは，高度な信頼性，可用性，保守性が要求され，従来メインフレームコンピュータ上でのみ稼働が可能であったミッションクリティカルシステムのプラットフォームである．

日本ユニシスでは，長年メインフレームベンダーとして数多くのミッションクリティカルシステム構築を手掛けてきた．そして，そこで培われたノウハウを結集し，Windows 上でミッションクリティカルシステムを構築することにチャレンジした．その成果が，2001 年秋に本番稼働を開始した三井住友銀行の対外接続システムである．稼働開始から現在までの 3 年半の間，大きなトラブルもなく順調に稼働し続けている．

さらに，現在，Windows 2003 上で百五銀行の次期勘定系基幹システムを構築中であり，2007 年度より本番稼働する予定である．

こうした，Windows 上でのミッションクリティカルシステム構築のノウハウを.NET に適用し，.NET をミッションクリティカルシステムのプラットフォームとして位置付けたものが日本ユニシスの.NET ミッションクリティカルプラットフォームである．

日本ユニシスでは，ミッションクリティカルシステムのプラットフォームが備えるべき要件として以下の 3 点が必須と考えている．

① 業務アプリケーション開発者が，ミッションクリティカルを意識せず，業務部分の開発に専念できること

② 障害の発生を迅速に検知し，適確にリカバリでき，障害原因が特定できること

③ アーキテクチャと開発プロセスが定義されていること

これらの要件を，具現化したものが

① MIDMOST for .NET

② ACAB（Advanced Cluster Application Builder）

③ LUCINA for .NET

である（図 3）．

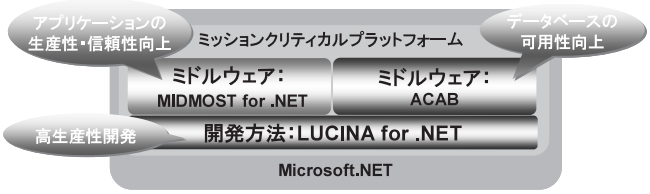

図 3　.NET ミッションクリティカルプラットフォーム

それぞれについて，その機能と特徴を紹介する．

4.1 MIDMOST for .NET

.NET では，Windows 上で業務システムを構築する際の基本的な開発基盤，運用基盤が提供されている．.NET が提供する開発基盤，運用基盤は，.NET 以前の Windows 上のものと比較すると格段に整備されてはいるが，メインフレームコンピュータが提供しているものと比較すると不足しているものがある．現在メインフレームコンピュータ上で運用されている基幹業務システムと同水準のサービスレベルを提供するシステムを.NET 上で実現するためには，こうした不足している部分を補完する必要がある．

これらの不足している機能をミドルウェアとして製品化したものが MIDMOST for .NET である．MIDMOST for .NET を利用することで，開発者は業務処理の開発に専念することができ，運用管理者は統合的な管理をすることが可能となる．

なお，MIDMOST for .NET に関する詳細な説明は，本誌掲載の論文「ミッションクリティカルシステム構築・運用支援ミドルウェア『MIDMOST for .NET』（溝上昌宏著）」で記述されているので参照願いたい．

4.2 ACAB

Windows 上で高可用性を実現する技術として MSCS（Microsoft Cluster Service）がある．MSCS では，最高 8 台までのサーバを組み合わせ，Shared-Nothing のクラスタ構成を構築することができる．クラスタ構成に参加している各々のサーバをノードと呼び，ノード障害が発生した際には，障害が発生したノードで実行していたサービスが，他のノードに引き継がれて実行される．

高可用性が求められるシステムを構築するには，以下の 3 点が重要であると考える．

① 障害の発生を迅速かつ的確に検知できること
② 障害内容によって迅速かつ的確にリカバリ動作を行えること
③ 障害原因の追求が可能であること

ACAB は，この 3 点を実現するために，MSCS では十分でない機能を補完したミドルウェアである．

1） 障害の迅速な検知

ACAB は，MSCS だけでは十分でない以下の障害検知機能を補完する

・プロセスの生死は，プロセスが存在するだけでなく，コマンドや電文による監視を行う
・プロセスのメモリ使用量，ハンドル数の監視をする
・プロセスの終了処理は kill ではなく，正常な終了処理を可能とする
・業務で使用しているデータベースにアクセスできない状態になっていることを検知する
・ローカルディスクの障害を検知する

2） 適確なリカバリ

ACAB は，MSCS だけでは十分でない以下のリカバリ機能を補完する

・クラスタリソースが障害となっても，直ちにフェイルオーバせず縮退運用をする
・障害ノードを強制的にリブート，シャットダウンさせる
・クラスタリソースの障害時にコマンドを実行する
・フェイルオーバ時に設定したコマンドを実行する

3） 障害原因の追及

ACAB は，MSCS だけでは十分でない以下の障害原因の追及に関する機能を補完する

・クラスタリソースの挙動に関するメッセージをイベントログに出力する
・障害の発生したプロセスのメモリダンプを採取する
・障害の発生したノードのメモリダンプを採取する
・発生した情報が上書きされないように，クラスタログのバックアップをとる

4.3 LUCINA for .NET

LUCINA for .NET は，企業システム全体の最適化を目指した，.NET によるアプリケーション開発のアーキテクチャと開発プロセスを定義したものである．LUCINA for .NET では，企業システム全体をエンタープライズシステムと定義し，エンタープライズシステムはコンポーネントシステムと呼ばれるサブシステムの組み合わせで構成されると考える（図 4）．

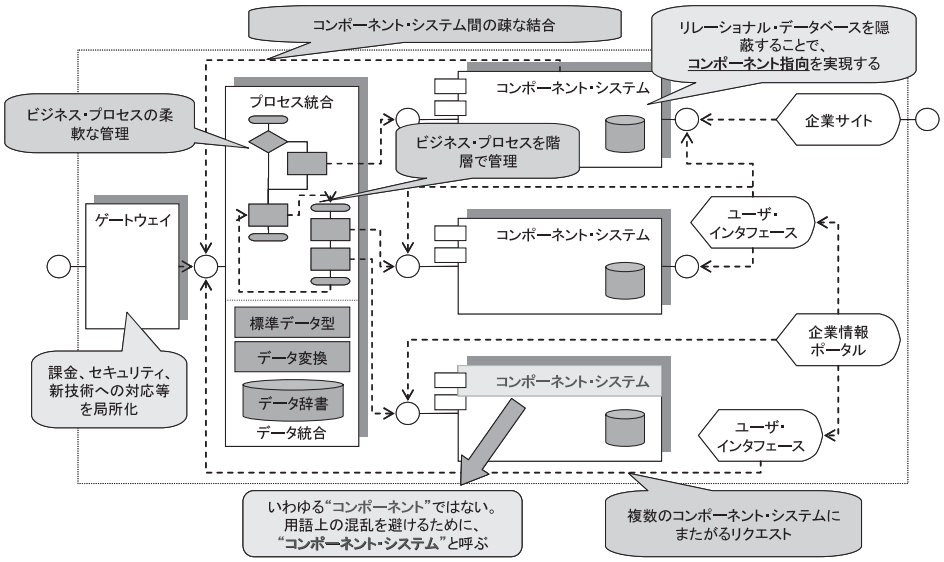

図 4　エンタープライズシステムとコンポーネントシステム

それぞれのコンポーネントシステムは，図 5 に示すように，ビジネスロジック層，データアクセス層，データ層の論理 3 階層構造をとっている．

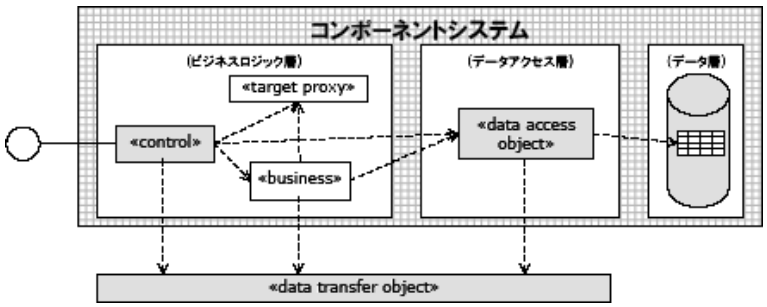

図 5　コンポーネントシステムの論理構造

LUCINA for .NET の開発プロセスは，コンポーネントシステムの外部仕様を導出するまでの「業務分析フェーズ」とその内部仕様を設計して実装する「システム構築フェーズ」の 2 つで構成される．開発プロジェクトの開発体制とフェーズ毎の各チームの主たる作業を図 6 に示す．

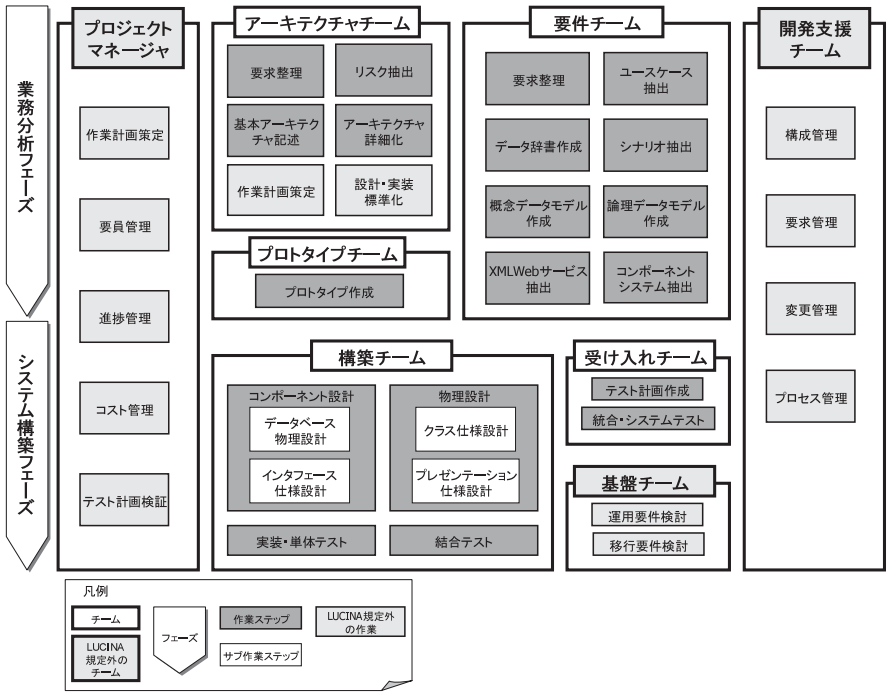

図 6　開発体制と各チームの主たる作業

LUCINA for .NET の開発プロセスの特徴は，ユースケース駆動，アーキテクチャ中心，データ中心アプローチである．

なお，LUCINA for .NET に関する詳細な説明は，本誌掲載の論文「開発方法 LUCINA for .NET――開発プロセスとアーキテクチャの策定（白木和彦，幸野俊介著）」で記述されているので参照願いたい．

5. お わ り に

マイクロソフトが 2005 年中にリリースを予定している Visual Studio 2005 では，SOA を意識したモデリングツールが提供される予定である．.NET の本来の姿である「サービスとしてのソフトウェア」は，SOA の本格的な普及とともに，必要となる基盤技術が整備され，現実化していくと予想される．

IT は今やライフラインのひとつとなっているが，「サービスとしてのソフトウェア」が普及したら，さらにこの傾向が強くなって行くことであろう．ひとつのサービスが停止することで，日常生活に多大な影響を及ぼす可能性が出てくる．従来，ミッションクリティカルと言えば，金融機関の基幹系システムをイメージしたが，今後はあらゆるものにミッションクリティカル性を要求されることになるであろう．

どちらかというと高生産性が強調される.NET であるが，日本ユニシスの.NET ミッションクリティカルプラットフォームでは，信頼性，可用性，管理性，保守性といった面をより充実させていきたいと考えている．

＊1　SOA：ネットワーク上で散在するシステムを，業務視点で部品化された機能（サービス）の集合と捉え，業務要求に対して，必要となる機能（サービス）を組み合わせ，処理や情報を統合するシステム構築方針である．

参考文献
［1］ 赤間信幸，.NET エンタープライズ Web アプリケーション開発技術大全 Vol.1，日経 BP ソフトプレス，2004
［2］ Thuan Thai, Hoang Q. Lam,.NET Framework エッセンシャルズ，O ’ Reilly, 2002
［3］ 牧野友紀，「サービス指向アーキテクチャによるビジネス・プロセス統合」，UNISYS 技報 81 号，2004，日本ユニシス
［4］ 今道正博，猪股健太郎，「LUCINA for .NET によるアーキテクチャ中心のプロジェクト・アプローチ」，UNISYS 技報 81 号，2004，日本ユニシス
［5］ Joseph, McKendrick, January 2000, “ The Post-Y 2 K Wish List ”, Enterprise Systems Journal, p.18.

執筆者紹介　山　岸　重　雄（Shigeo Yamagishi）
1983 年日本ユニシス(株)入社．主にオフィスコンピュータでのシステム開発，UNIX&PC でのクライアントサーバシステム開発，Windows 上でのミッションクリティカルシステム構築，.NET でのシステム構築支援に従事．現在，日本ユニシス・ソリューション(株)テクノロジコンサルティングサービス　ミドルウェアビジネス基盤ビジネスに所属．