.NET が実現するこれからの情報システムモデル

成 本 正 史

1. はじめに

「Forum 2000」で.NET を発表して以来 2005 年で 5 年目を迎えようとしている． Java の対抗馬や MS の製品ブランドと間違われるなどいろいろな歴史があったが，ここでもう一度 .NET の掲げるビジョンについて整理する．

そもそも.NET はインターネットバブル期の「.COM」時代からの進化系として世に登場してきたが，世界共通基盤であるインターネットを最大限に有効活用し，人々の生活をより快適で豊かに，ビジネスにおいては IT システムをより迅速にまた効率的なバリューチェーンを実現するというビジョンを掲げ，そのビジョンを推進するものとしてさまざまなテクノロジーを開発しマーケットへ提供し続けている．つまりマイクロソフトが考えるこれからのソフトウェアの進むべき方向性が .NET なのである．今後も開発ツール，フレームワークおよびその他の製品群はこの方向性に従って継続的に進化しつづけるであろう．本稿ではその方向性および .NET によって実現される新しい IT システムモデルについて紹介する．

2. これからの IT ライフサイクル

今後の IT の進むべき方向を考える上で，現在抱えている課題を整理してみたい．

エンタープライズにおいて大規模の情報システムを構築する場合に何が課題となっているのであろうか？まずはビジネス要件を分析するところから課題を抱えている．現在のビジネスは多くの部分を IT に依存しており，その複雑な要件を把握して望みどおりの情報システムを構築することが一番大きな課題となっている．エンタープライズアーキテクチャが脚光を浴びているのは，その課題を解決する可能性があるからである．次の課題としてテクノロジー選択が挙げられる．テクノロジーは年々新しくなっている．情報システムを新規に構築する，もしくはリプレースするといった場合にどのベンダーのどのテクノロジーを選定すればいいのか悩まれているケースが多いのが実情である．次に大規模なシステムを一貫性を保って高品質なものに仕上げるためには優れたアーキテクチャが必須となる．本来はエンタープライズ規模で全体を統括するアーキテクトの存在が欠かせないが，ごく一部の方を除きそういうミッションを達成できる人材が不足している．情報システムへのビジネスの依存度が強まるに従って信頼性の要件もますます厳しくなっている，情報システムが止まればビジネスが止まってしまうという時代をすでに迎えている．過去からの積み上げで IT の資産も膨大に膨れ上がりこれらを運用・保守していくことが困難になりつつある．また一世代前と比較するとビジネスそのものの変化のスピードが加速してい

る．企業の統廃合はもちろんのこと新規ビジネスへの進出・撤退，ビジネスパートナーとの関係など変化要因は枚挙にいとまがない．そういった変化への対応も情報システムに求められるわけである．つまり要件分析から保守にいたるまで IT のライフサイクル全体にわたる課題を抱えている．

ではこの課題をどのように解決していけばよいのであろうか．それぞれの課題についてはこう解決したいという考え・意図を漠然と持っていることが多いと思う．問題はその意図を明確化して関係者間で共有できていないことにある．本来は課題を明確に認識し，漠然としていてもよいのでお互いが考えている解決策を共有し，それらの整合性を保ちながらシステムの構築を進めることが望まれる．それに対する答えが『モデル駆動』である．

ばらばらな意図を，一旦，モデルとして可視化し，そのモデルを中心に個々の課題を解決していくというアプローチが『モデル駆動』である．マイクロソフトはこの『モデル駆動』を強力に推進している．開発のみならず，要件分析から運用・保守，システムのバージョンアップにいたるまで情報システムのライフサイクル（システムが誕生してから消滅するまで）の間ずっとモデルを中心に据えたアプローチを採用するわけである．マイクロソフトではこれを“ LIVE　MODEL ”と呼んでいる．ツールで作成し，開発完了時点で役に立たなくなる従来の“ MODEL ”とは別次元だということ意味している．

では『モデル駆動』の具体的な取り組みとしてマイクロソフトが推進しているダイナミックシステムイニシアチブ（DSI）をご紹介する．DSI は以下の三つの要素により構成され，『情報システムのライフサイクル全般にわたる自動化・簡素化を実現する』ことをゴールとしている．

・自律型コンピューティング
・モデル駆動型開発
・サービス指向に基づくソリューションスイート

以降，それぞれについて紹介する．

2.1　自律型コンピューティング

自律型コンピューティングは，一般的には「ユーティリティコンピューティング」という名称で知られているが，仮想的なインフラストラクチャを構築し，システムの負荷状況に応じて動的にハードウェアリソースを割り当てるという試みである．典型的なケースとしては朝一番と帰宅直前などにシステムの負荷が最大になるため，個々のシステムはその最大負荷に耐えられるようインフラを設計するが，全社規模で見ると必ずしもすべてのシステムが同時刻に最大負荷がかかるわけではなく，遊んでいるサーバーも多数ある．こういった冗長なインフラ構成を防ぐために，インフラ自体を仮想化し，負荷に応じてリアルタイムにハードウェアのリソースを現時点で負荷の高いシステムに割り当てることで，全体最適を図ることが可能となる．ただそういった形で負荷に動的に対応できるデータセンターを達成するためにはインフラの仮想化だけでは不十分である．アプリケーション配布や，データセンターの運用ポリシー，またシステム稼動時の運用管理までを視野にいれた包括的な取り組みが必須である．そのために開発の初期段階から運用管理のことを考慮し，

アプリケーションの設計を実施するというアプローチを採用している．Visual Studio 2005 という 2005 年にリリースされる統合開発環境では，アーキテクチャを構築する際に論理的なデータセンター（サーバー，ストレージおよびネットワークなど）をビジュアルに表現し，サーバーなどそれぞれの運用ポリシーを定義する．そしてアプリケーションの動作条件（性能，セキュリティ，プロトコルなど）も初期段階から定義をしておくことで，実際に展開する前段階からデータセンターの運用にマッチングさせることが可能となる．詳しくは 4 章で述べる．

これらの情報はすべてモデル（System Definition Model-SDM）として格納され，アプリケーションの配布，運用はこの SDM を参照する形でほぼ自動的に実現することが可能となる．またシステム改変時にはこのモデルを変更することで変更部分がこれもほぼ自動的に適用される．さきほどの“ LIVE MODEL ”はこういった意味を持っている．

このようなデータセンターはマイクロソフト単独では実現できない．主要なハードウェアベンダーとの協業をすでに開始している．

2.2　モデル駆動型開発

2.1 節ではモデルを中心としたライフサイクルを紹介したが，ここでは特に注目を集めているモデル駆動型開発について紹介する．情報システムを構築していく上でビジネスモデリングから物理的なインフラストラクチャまで，また構築フェーズすべてにわたって複数のモデリング対象がある．マイクロソフトが考えるモデル駆動型開発とは，これらモデル対象に最適なモデリング環境を提供するというものである．モデル対象によってそのモデラーの得意分野はまちまちである．重要なのはそのモデラーの持つ知識・意図でありツールや表記法（モデリング言語）を使いこなすスキルが重要なわけではない．モデラーごとに使いやすい表記法・ツールが存在していて，知識・意図を正確にインプットして，各モデル間の連携・整合性が確保されていればよいわけである．IT 業界では表記法を統一しようという動きもあるが，単一の表記法でこれら多種多様なモデリングに対応するというのは少々無理があり，現時点でその限界は露呈されている．エンドユーザーの方は豊富な業務知識を基に，そこからシステムに対する要件をモデルを通じて明確化することが重要であるが，必ずしもすべてを UML で記述しなければいけないわけではないのである．運用管理についても同じことが言える．もっと簡単にモデルが作成できる環境があるべきである．マイクロソフトはこのような考え方に基づいてモデル駆動を実現している．

ビジネスモデルからインフラストラクチャに至るまで各モデリング対象に最適化された環境を提供し，それぞれの成果物として生成されたモデル間を自動的にマッピングすることで，整合性を保つという思想を表している．モデル間のマッピングをどのように自動化するかというと，例えばデータに関しては，ビジネスモデルとしてその企業で利用されているデータ（顧客，製品，在庫など）を抽出し，論理的なデータパターンを適用することで，アプリケーションで処理しやすいモデルへと変換する．そして最終的にはデータベースサーバーでのデータスキーマ（構造）に変換されるわけである．その際にはデータベースサーバーが最大の性能を発揮できるようにスキーマが生成される．データ以外の分野に

ついても同様にパターンを適用することで，モデル間のマッピングを達成している．ただ現時点では約 7 割がこのような自動的な仕組みで実現可能と予測されており，残りの 3 割程度は手作業が必要になる．技術の進歩とともにこの割合も改善していくことになる．

2.3　ソリューションスイート（Windows Server System）

今後のシステム開発の手法としては，「作らない」アプローチが望まれる．つまりすでに存在している製品を利用することで開発期間の短縮，コスト削減，品質の向上が達成できるのであれば，それを採用しない理由はない．また，これらの製品は世界中で利用されているので，一から作るのに比べればはるかに高品質なものが手に入るわけである．

ここからはマイクロソフトが各ソリューションごとに提供している製品群である Windows Server System について紹介する．

Windows Server はサーバーの基盤となる技術がすべて搭載されている．また Windows Server はアプリケーションサーバーの機能を包含している．アプリケーションサーバーとは Web サーバーに始まり，トランザクションモニター，ディレクトリサービス，セキュリティサービス，メッセージキューイング，コンポーネント実行環境などを総称したものを意味している．Windows　Server はこれらをすべて提供し，かつ Web サービスを始めメディアサーバーやコミュニケーションサービスなど数々の先進的機能も搭載し，運用管理・ユーザーインタフェース・ユーザー管理はすべて統合されている．このサーバープラットフォーム上で動作する各サーバー製品は予め統合された形で提供されているので，構築・運用管理が大幅に簡素化される．いまや世界中のサーバーの 70% は Windows Server という時代になっているので，ソリューションごとのサーバー製品もこの基盤の上に最適化され，かつ互いに統合された製品群を使うことを推奨している．マイクロソフトの考え方として，OS やサーバー製品などのプラットフォームを進化させ続けることで，アプリケーション横断的な機能はすべて提供し，残りのわずかな部分だけを開発するという方針に基づいて日々各種製品の開発を行っている．

アプリケーションサーバーの機能もその中の一部であり，WindowsNT の時代から提供しつづけている．図 1 に示すとおり OS や DBMS などの基本的なプラットフォームはその機能を発展しつづけており，さらに Out of the Box（箱から出してすぐに使える）なソリューションスイートを Windows Server System として提供しているので，開発しなければならないアプリケーションの割合は劇的に減っている．

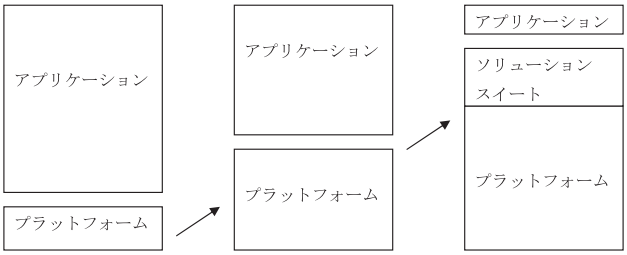

図 1　プラットフォームの進化

また Windows Server System はサービス指向アーキテクチャ（SOA）に基づくラインナップとして開発している．例えば SQLServer ではデータの受け渡しを Web サービスで実現することが可能であり，XML というデータフォーマットを格納する処理も最適化されている．2005 年リリース予定の SQL Server 2005 ではこれらの機能がさらに向上することになる．また SOA では要件分析から実行環境にいたるまでビジネスプロセスが中心となるが，そのビジネスプロセスを定義し，かつ実行エンジン，モニタリングまで提供するサーバーとして BizTalk Server を提供している．この BizTalk Server は開発環境と統合されているのでアプリケーション開発時にそのままの開発環境で利用することが可能である．その他の製品についてもサービス指向アーキテクチャを適用するために最適化されたものとして設計されているので，これからの情報システムを考慮する上で優れた選択肢であると考える．

3. これからのアプリケーションスタイル

2 章で IT ライフサイクルがどのように変わっていくかについて述べたが，本章では今後のアプリケーションスタイルについて説明する．図 2 にあるとおり，アプリケーションを構築するには左側の垂直型の要素が必要となる．一番下からいくとシステムが動作する基盤であるサーバーやストレージなどのプラットフォームがあり，次に各アプリケーション間が相互接続するためのものとしてリモート呼び出し機能やセキュリティ，トランザクション，信頼性などがある．そしてこれらの上に各ビジネス要件に応じたアプリケーションが存在する．

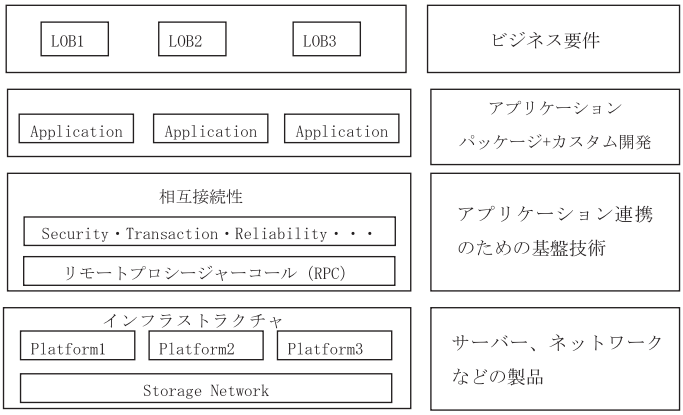

図 2　アプリケーション構造

3.1 アプリケーションスタイルの比較

現状ではこれらをどのように扱っているかというと，図 3 にあるように特定プラットフォームに依存する形でアプリケーションが構築されている．プラットフォームに依存しないということを謳っている技術もあるが，実際には特定ベンダーの製品に依存する形になってしまっているのが実情である．また各アプリケーションは特定要件に最適化されたも

のとして単体で作成されているので個々の要素を再利用することが困難になっている．オブジェクト指向やコンポーネント指向などの試みで解決しようという努力がなされてきたが，現状のマーケットでは再利用性は実現されていない．その結果としていわゆる“アプリケーションサイロ”（ばらばらなアプリケーションの寄せ集め）になってしまっていて相互の連携がうまくとれていない．この問題をEAIで解決しているが本来の姿ではない．さらにはビジネス要件をITシステムとして構築する過程でうまく整合性がとれないため，発注者から見ると満足のいくシステムができあがらないという結果になる．

では今後いかにしてこういった問題を解決していけばよいのであろうか？その答えを図3の右側に示している．インフラストラクチャは仮想化されるため物理的な配置などは意識する必要がなくなる．そして相互接続性に関してもWebサービスによって標準化が進んでいる．このような環境の上でアプリケーションは内部機能をそのまま外部に提供するのではなく抽象化されたサービスとして構築されることになる．各サービスはビジネス概念を表したものなので，ビジネスモデリングの結果として抽出された概念モデル，プロセスモデルがそのままサービスに変換されることになる．このようなシステムを実現するためのアーキテクチャがサービス指向アーキテクチャ（SOA）である．

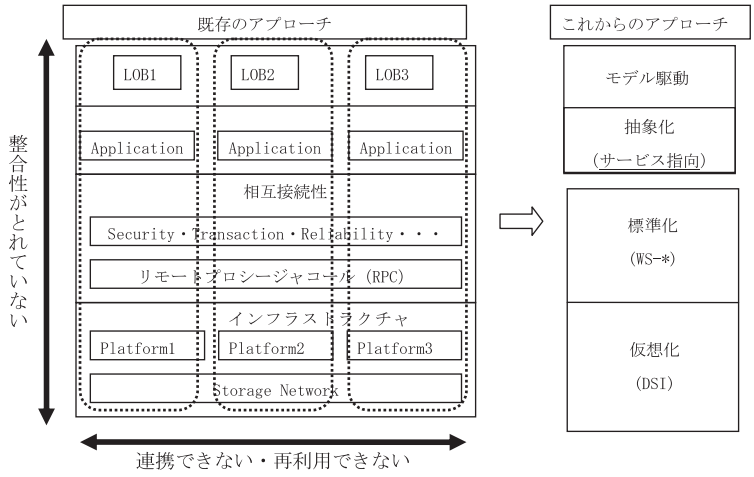

図3　アプリケーションスタイルの進化

3.2　サービス指向アーキテクチャ（SOA）

企業活動をモデル化すると，それらはITシステムへの多岐にわたる要件へと変換される．これらの要件をIT化するときに発生するギャップをいかに解決するかが，今，重要な課題となっている．サービス指向はこのギャップを埋めることに大きく貢献するものである．つまりビジネスを構成する諸概念（ビジネスプロセス，エンティティなど）をサービスとして構成し，それらを利用するためのビジネスルール，制約条件をそのサービスの契約という概念で明確化することでビジネスの概念がそのままITのサービスへとマッピングされる．まず，サービス指向を前提とした場合には企業活動をモデル化する際にビジ

ネスプロセスに着目する．大企業の場合にはおおよそ 10 個のコアとなるビジネスプロセスが存在すると言われている．そのビジネスプロセスは複数のアクティビティにより構成され，エンティティ（データ）にアクセスする．

このようにモデル化された各ビジネス概念をサービスとして抽出する手法を以下に示す．

まず業務プロセスそのものがサービスになる．そしてプロセスを構成する各アクティビティもサービスとして構築される．またプロセスがアクセスしているエンティティを複数業務横断的に最適化した結果，これもサービスとして構築される．さらには業務プロセスを実行するための非機能要件（性能・コスト・信頼性など）をもとにインフラストラクチャを選定する．これらサービスを利用するユーザーにはその環境に最適なデバイスを提供することで利便性を高めることが可能となる（図 4）．

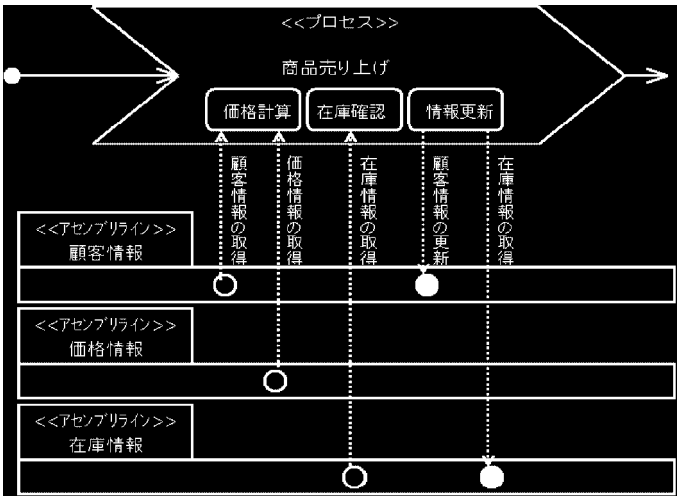

図 4　業務プロセスモデルからサービスの抽出

サービス指向アーキテクチャという考え方は従来から存在したが，Web サービスや XML の普及などテクノロジーの進歩により，実現環境が整ったことが普及の大きな要因として考えられる．またサービス指向アーキテクチャが普及しつつある背景として，情報システムがカバーする範囲の広がりを無視することはできない．過去 20 年間を振り返っても対象としている範囲に応じた開発方法論が存在している．これを図 5 に表す．開発対象がスタンドアローン型のアプリケーションであった時代からネットワーク経由でのクライアント・サーバー型に移行し，Web アプリケーションを経て現在では複数システムが統合され，さらには企業間でのビジネス取引をもその範囲に含む時代に突入している．このように対象システムの規模が大きくなるにつれて，開発方法論も，オブジェクト，分散オブジェクト，コンポーネントそしてサービスへとその概念的粒度が大きくなってきていることがわかる．

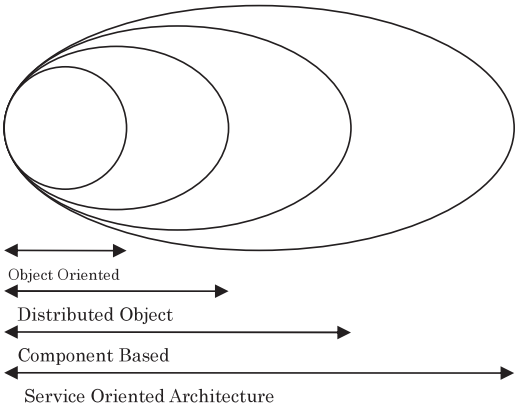

図 5　開発方法論の変遷

ただ，サービス指向と Web サービスとを同じものとして考えると無理が生じる．図 6 に相違点をまとめる．

Web servicesで可能なこと	***テクノロジーへの中立性***	プラットフォームには依存しない
	標準化	標準プロトコルを提供
	広範囲な利用性	自動的なディスカバリと利用が可能
SOAで可能なこと	***再利用性***	コードや**EXE**のコピーではなくサービスを再利用
	実装非依存	サービスは物理実装には依存しない
	インタフェースを公開可能	標準フォーマットでのインタフェース公開が可能
	利用者とプロバイダーとの契約を明確化	利用者とプロバイダー間で標準に基づく契約を締結
	利便性の向上	利用者から見て適度な機能セットが提供されている

図 6　Web サービスと SOA の違い

ここからはサービス指向アーキテクチャに従った実装方法について紹介する．

まず“サービス”の簡潔な定義を以下に記述する．

「サービスとはネットワーク呼び出し可能なソフトウェアで，業務ロジックを実装し，状態を管理し，メッセージ交換により対話を行い，ポリシーにより運用される」

サービスの粒度に関する指針としては以下のものが考えられる．

・管理するデータの自律性：外部に依存しない
・提供する機能の自律性：外部に依存しない
・バージョンアップの頻度：変化の度合いが一定である
・再利用の単位：再利用性の高いものを独立したサービスとして提供

特にサービスの自律性を確保することが重要である．サービス同士が連携すること自体は問題ないのであるが，サービスの内部実装や管理しているデータが他のサービスに依存

しているような状態は避けるべきで，必然的にサービスの粒度はコンポーネントなどと比較すると大きくなる傾向にある．サービスは粒度の大きいコンポーネントと考えることもできるが，コンポーネントとは以下の相違点がある．実際の開発プロジェクトではこれらの間のトレードオフを考慮して適用方針を決定するべきである．

1)　サービスの利点
- ・独立した運用管理，配布が可能となるため自律性・保守性が向上する
- ・異なる実装技術同士での対話が可能
- ・物理配置，外部システムへの依存度が弱まるため拡張性が向上する
- ・抽象化のレベルが上がるので，再利用性が向上する
- ・サービスとはビジネス概念からトップダウンで抽出されたものなので要件を忠実に反映することに寄与する

2)　サービスの欠点
- ・非同期メッセージ処理，例外処理，適切な粒度の選択など設計の難易度が上がる
- ・抽象化レイヤーが増えるため処理のオーバーヘッドが増える

サービス指向アーキテクチャを構成する要素の一覧を表 1 に示す．それぞれの構成要素がアーキテクチャを構築していく際でのポイントになる．

表 1　SOA の設計要素

構成要素	意　　味
サービス	ネットワーク経由で機能、データを提供する実体
メッセージ	サービス間で転送される情報の単位
インタフェース	サービス機能を公開する手段
コントラクト	対話するもの同士が相互に取り交わす契約
ポリシー	サービスが従うべきルール
カンバセーション	インスタンス化されたコントラクト
状態	サービスが管理するデータ、業務データ、プロセス状態、メッセージなど
トランザクション	ACID, 補正トランザクション
プロセス	複数サービス、ビジネスロジックのフロー制御

4.　これからの開発スタイル

現時点におけるソフトウェア開発と製造業におけるそれとを比較すると残念ながらソフトウェアは手工業といわざるを得ない．つまり産業革命以前もしくはその途中の状態である．

では今後どのように発展していくのであろうか（図 7）．2 章でも述べたとおりビジネスアナリストから開発者，運用担当者にいたるステークホルダーはそれぞれの役割に応じた意図を持っている．つまりこういう IT システムを作りたいという思いから，一度作ったものを再利用したい，効率的な運用管理を実現したい，といった意図が存在するのである．

今後の開発スタイルはこれらの意図を汲み取ってモデルを構築しそのモデルを中心として各成果物をほぼ自動的に構築するというスタイルになる．つまりソフトウェアは手工業

からオートメーション化されていくのである．それを実現するのがソフトウェアファクトリである．

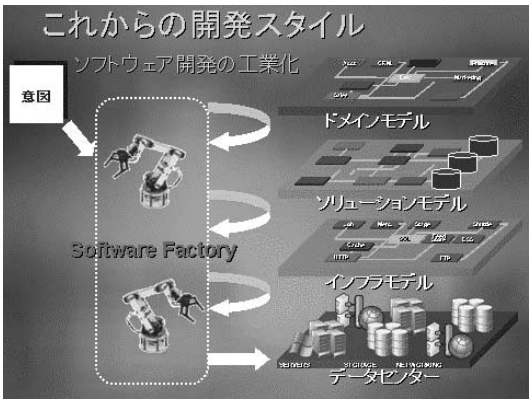

図7　ソフトウェアファクトリ

4.1　ソフトウェアファクトリ

ソフトウェアファクトリではプロダクトラインという考え方をベースとしている．つまり製造業でいうところの生産ラインである．プロダクトとはソフトウェア（システム，アプリケーション）のことである．簡単にいうとソフトウェアファクトリとはソフトウェアの仕様（漠然とした意図）を生産ラインに乗せればあとは自動的にデータセンターができあがる仕組みである．しかもソフトウェアの場合にはハードウェア製品とは異なり仕様がはっきりせず，かつ多品種少量になる．そのため漠然とした意図を明確な仕様に変換するところから始まり，またソフトウェアの種類に応じて生産ラインを柔軟に組み替える必要が出てくる．詳細を図8で示す．

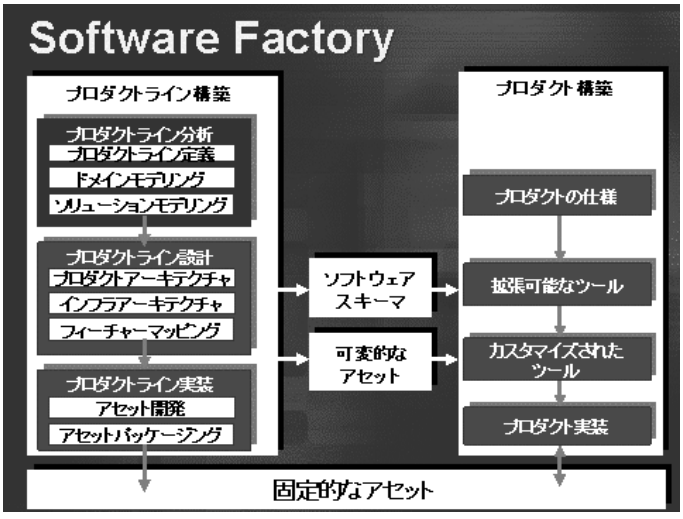

図8　ソフトウェアファクトリの構成

作業手順としてはまず構築しようとしているソフトウェアの中から類似性のある集団を抽出する．この集団をプロダクトファミリーと呼ぶ．そしてその中身を構築するにあたり，ファミリーごとに図の左側にあるプロダクトラインを構築する．プロダクトラインを構築するには，まずそのファミリーが扱う問題領域（ビジネスドメイン）のモデリングを実施する．そしてその問題領域を解決するソリューションモデリングを行う．これらモデリングの過程で必須のフィーチャー(提供される機能)と可変的なフィーチャーとを識別する．これが FODA（Feature Oriented Domain Analysis）と呼ばれるものである．一旦，フィーチャーが洗い出されると，それを実行するためのアーキテクチャを設計する．そしてそのアーキテクチャに基づいて再利用可能なアセット（ソフトウェア資産）を実装する．ここでいうアセットとはコンポーネントや開発プロセス，アーキテクチャ，成果物一覧，ドキュメントテンプレートなどがすべて含まれる．

以上の作業により構築されたプロダクトライン（生産ライン）は似通った各プロダクトを構築する上での最適な環境を提供するので，各プロダクトはそこからのわずかな差分だけを扱えばよいことになる（図 8 中右側）．まず個別の仕様を開発ツールにインプットすることで作業を開始する．開発ツールはプロダクトライン構築作業によって生成されたソフトウェアスキーマ（モデル成果物）をインポートすることで，プロダクトを開発するために最適化（カスタマイズ）された環境に変わる．その最適化された開発環境を利用し，固定的なアセットを利用することでほぼ 70% の作業は自動化される．残り 30% は可変的なアセットを利用しながら開発者が作業を行うことになる．

4.2 開発プロセス　Microsoft Solution Framework（MSF 4.0）

MSF は長年にわたるマイクロソフトの製品開発から得たれたベストプラクティスをベースとして 1994 年に登場したもので，今日にいたるまで継続的に進化し現在 3.0 がリリースされている．2005 年は MSF 4.0 という新しいバージョンをリリースする．開発プロセスに関しては，ここ数年の傾向として XP などに代表されるアジャイルプロセスモデルと，UP などのフォーマルプロセスモデルの 2 大勢力にわかれているが，それぞれに対して一長一短があり，プロジェクトタイプ，規模，開発メンバーのスキルおよびスケジュールなど与えられた条件に適したものを選択するのが相応しいのである．MSF には中小規模プロジェクト向けのライトウェイトなプロセスである MSF　Agile と大規模向けプロセスである MSF　Formal の 2 種類が存在し，チームモデル，ワークフローやワークアイテムが定義されている．これらはいずれも後述する開発ツール Visual Studio 2005 に組み込まれている．

このようにプランニング－開発－テストという作業をインクリメンタルに繰り返し実施していくが，作業を進めるにあたり全体の中のどの部分を担当するかを明確にし，ロール（役割）という概念でチームメンバーをモデル化している．

※詳細は下記サイト（英語）にて公開している．

http://www.microsoft.COM/downloads/details.aspx?familyid=9f3ea426-c2b2-4264-ba0f-35a021d85234&displaylang=en

4.3 開発ツール　Visual Studio 2005 Team System

Visual Studio 2005 Team System は DSI で実現しようとしているダイナミックなシステム構築を支援するための機能を提供している．またそれに加え，システム大規模化，複雑化に対処するために開発ツールに求められる以下の機能をすべて提供している．

・ソフトウェア開発ライフサイクル（SDLC）全般にわたる統合された環境
・要件定義から実装にいたるまで一貫したモデリング機能
・プロジェクトごとに選択可能な開発方法論・プロセスとそれに基づくプロジェクト管理機能
・コード解析機能
・ユニットテストや負荷テストなど充実したテスト機能
・柔軟な拡張性

まず Visual Studio Team Foundation としてソース管理や成果物・スケジュール管理など開発プロセスに基づくプロジェクト管理機能を提供する．さらには進捗状況の共有ポータルなどのコラボレーション機能も合わせて提供する．そして Visual Studio Team Suite がアーキテクト，デベロッパー，テスターといった開発プロジェクトにおけるチームモデルに対応する形でそれぞれの機能を提供している．また本ツールはマイクロソフトが採用している開発プロセスである Microsoft Solutions Framework（MSF）およびサービス指向に基づいた分散システムアーキテクチャのガイダンスをサポートしている．そして Visual Studio Industry Partners（VSIP）により本ツールのあらゆる側面で拡張性を提供している．ここではアーキテクト向け機能である分散システムデザイナー（コードネーム“ Whitehorse ”）とそれに関連する部分を重点的に取り上げる．

分散システムの設計が困難な理由としては，既存資産を含めて散在する複数のサブシステム，アプリケーションの構築，運用管理をシステム全体を通じて可視化し，連携させなければならないことが挙げられる．また設計時点でアプリケーションの配布や運用管理の観点が考慮されないと後になって物理構成との不整合が判明し，期待した性能が出ない，余計な追加投資が発生するといった問題点を抱えることになる．今日の時点ではこういった問題を解決する手段としてドキュメントを通じた関係者間のコミュニケーションにより解決する努力がなされているが，このようなドキュメントはフェーズが進むにつれて実際のコードや物理システム構成とのギャップが発生し，システム規模が大きくなるにつれ管理能力の限界に達する．結果的に保守に大きな課題を残すことになる．これら課題を克服するツールとして登場するのが分散システムデザイナー（コードネーム“ Whitehorse ”）である．分散システムデザイナーは GUI により分散システムの設計・検証を可能とするツールで，内部的には以下の四つのデザイナーで構成される．これらのデザイナーは System Definition Model（SDM）と呼ばれるメタモデルを共有しているため，分散システムの構築から配置，実行時環境，運用管理にいたる複数のメンバーによって分割される一連のフェーズにわたって成果物のスムーズな相互利用が可能となる．またすべてのデザイナーで GUI による設計が可能なため，SDM のメタモデルやスキーマといった複雑さを意識する必要はない．

・Application Connection Designer
・Logical Datacenter Designer
・System Designer
・Deployment Designer

分散システムデザイナーの四つのデザイナー間の関係を図 9 に示す.

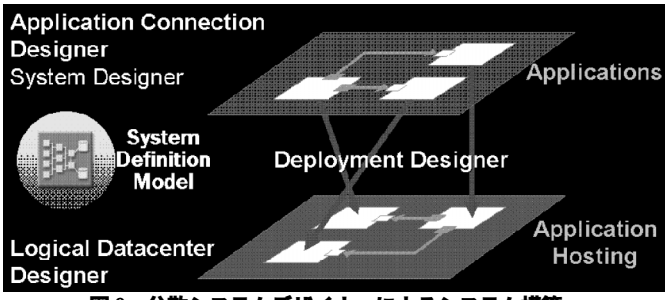

図 9　分散システムデザイナーによるシステム構築

ACD によりアプリケーションの接続関係，構成設定，ホスティング制約を定義し，System Designer によりシステム全体の観点でアプリケーションをまとめあげる．一方で LDD によりアプリケーションのホスティング環境の構成，制約条件を指定し，最終的には Deployment Designer によってそれぞれの要求と制約がマッチしているかの検証を行う．つまり個々のアプリケーション実装とは切り離した形でシステム構成が可能となり，初期段階から運用管理を考慮した設計がなされることになる．ネットワークトポロジー，性能，信頼性・セキュリティなどの非機能要件をもとにホスティング環境との整合性が事前に検証され，実行時に問題が判明して対処に追われるということはなくなる．逆に運用時にはどのサーバーでどういったアプリケーションが動作しているか，バージョンや通信プロトコルなどそれらの構成定義がすべて把握可能となる．これらはすべてが SDM を共有しているからこそ成せる技である. これらデザイナーは開発期間を通じて反復利用され，課題や要件変化のフィードバックループを確立する.

4.4　プロジェクト管理機能

今日，プロジェクト管理者は要件から実装へとシフトしていく作業間でのギャップや立案されたスケジュールと進捗状況のギャップ，チーム内でのコラボレーションの問題といった複数の課題に直面しており，これらを全て解決することは困難である．Visual Studio 2005 Team System のプロジェクト管理機能はこういった課題をクリアするための様々な工夫を施している．ここまで述べてきたようにベースとして SDM というモデルを共有しているため各作業における成果物のギャップはなくなり，作業のスムーズな連携が可能となる．またスケジュール管理に関しても MSF や 3rd パーティ製の開発プロセスに対応したテンプレートが用意されており，管理者の判断でプロセスの選択を行うことができ，選択したプロセスに従ってテンプレートが提供され，プロセスにより規定されているワーク

ストリームと個々のワークアイテム，マイルストーンごとに参加すべきメンバーロール，また次のマイルストーンに移行する条件などがあらかじめ設定されている．プロジェクト管理者は EXCEL を利用してそれらを編集することも可能である．また開発プロセスごとのドキュメントテンプレートが含まれており，仕様作成，リスクアセスメント，プロジェクトのプランニングなどの局面で利用される．その他レポーティングテンプレートも用意されていてプロジェクトの進捗状況が把握できるようになっている．さらにはプロジェクトメンバー間での情報共有のためのポータル機能も提供している．これら充実した機能を実現するために Visual Studio の IDE と Office, Microsoft Project, Windows Share Point Service, SQL Server 2005 Reporting Service などと連携することによりプロジェクト管理者の負荷を大幅に軽減することが可能となる．

5. ま と め

結果として生み出されるのは情報セントリックなシステムである．システムのユーザーが必要としているのはアプリケーションやソフトウェアではなく，情報なのである．ユーザーの観点からすればこういう情報が欲しい，または管理したいということであって，それらがどのように処理されているかは問題ではない．さらにはその情報を通じてお互いが連携し合うことが重要なのである．マイクロソフトでは Connected Systems と呼んでいるが，図 10 にあるとおり互いに協調してなんらかのタスクを実行するには（アプリケーションではなく）情報を通じて Connect されている必要がある．Connect とは単にネットワークがつながっているということではない．どのような環境でも常に情報の交換が実現でき，協調することが可能であるということを意味している．

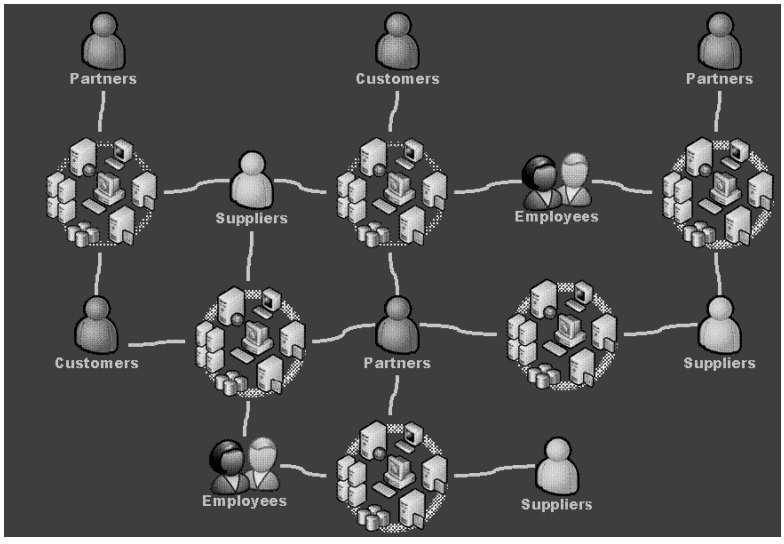

図 10　Connected Systems

このようなシステムを実現するためにマイクロソフトは製品を展開している．次世代プラットフォームである Longhorn では情報（デジタルデータ）を標準スキーマによって一元管理する WinFS，Web サービス標準すべてを実装する Indigo，革新的なユーザーインタフェースを提供する Avalon，そして DSI 構想を着実に具現化することで IT は次の世代へと進化することになる．

参考文献［1］ Software Factories Jack Greenfield（著），Keith Short（著）
［2］ DSI のホワイトペーパー
http://www.microsoft.COM/japan/windowsserversystem/dsi/default.mspx
［3］ Visual Studio 2005 TeamSystem
http://www.microsoft.COM/japan/msdn/vstudio/teamsystem/overview/

（マイクロソフト株式会社デベロッパーマーケティング本部
テクノロジーエバンジェリズム部マネージャ）