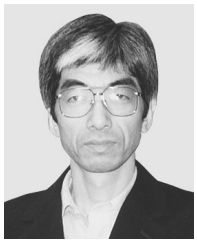
# トラディショナルITからひも解くe技術
## ーダイジェスト版ー



インターネットビジネス推進本部　真野　正

## １．はじめに

リックテレコム社の月刊誌『ソリューションIT』(2001.1より「ネットワークコンピューティング」から誌名変更)に、１年間にわたり「トラディショナルITからひも解くe技術」と題して連載する機会を得た。『ソリューションIT』は、一部のアンテナショップでしか店頭販売されていないが、「経営・業務課題に踏み込む問題解決型情報技術の専門誌」を標榜し、ITプロフェショナルの愛読者は多い。本稿では、その連載から主だったトピックを取上げ、紹介したい。

そもそもの発端は、筆者が出席したある新年会で『ソリューションIT』編集者と交わした一言であった。

・記者氏「Webシステムとかeシステムでは、新しい技術がどんどん出てきているが、....」
・私「Webシステムでの大量トランザクション処理なんか､昔ホストでIMS[*1](Information Managemet System）とかCICS[*2]（Customer Information Control System）でやっていた、トランザクション処理と本質は変わらないと思いますよ。昔、培ったアプリケーション・ノウハウが、eシステムでも充分活かされるんじゃないでしょうか。」
・記者氏「それ、面白そうですね。eシステムを構成する複雑な技術をレガシー（いやトラディショナルにしましょう）な技術と対比してみながら、解説するような連載をお願いできませんか。」

かくして、１年間の連載がスタートした。

## ２．トランザクション処理に見るTとe技術の変遷

B2C、B2Bのeビジネス・アプリケーションの構築では、あるWeb端末から発行した処理（商品発注など）を他の端末からの処理と識別し、整合性を保持しながら更新処理を完結させなければならない。このため、トランザクション処理機能を持ったAP（Application）サーバーが必要とされている。サーバー側がn層になり、インターネットを介しての処理となっているが、トランザクション処理機能そのものは、30年以上前から構築されている銀行オンライン処理と変わらないものである（図１参照)。

製品技術としては、メインフレーム全盛期には、IMS、CICS､COBOLでこと足りたが､Web時代となって､HTML(Hypertext Markup Language)、IIS（Microsoft Internet Information Server)、Apache、CGI（Common Gateway Interface)、さらにAPサーバーとしてのWebLogic、Websphere、そして言語はJavaというように多岐にわたってきている。

このように、製品は時代ごとに変わってきているが、インターネット時代を支える主要ITを構成要素に分解してみると、歴史的に培われた技術が多数応用されていることに気づく。新しい製品技術を追いかける際には、なぜ登場してきたのかを考え、過去に経験したものと対比してみることにより、その技術の真髄を理解できるというものだ。

トランザクション処理の例でも「なぜ、APサーバーが必要なのか」を考えてみる必要があるだろう。クライアン

＊１）IMS：IBMのトランザクション管理システム「IMS／DC」とデータベース管理システム「IMS／DB」の総称。
＊２）CICS：IBMのトランザクション管理システム。IMSではシステムとユーザー･プログラムの実行領域が異なるが､CICSでは同一。

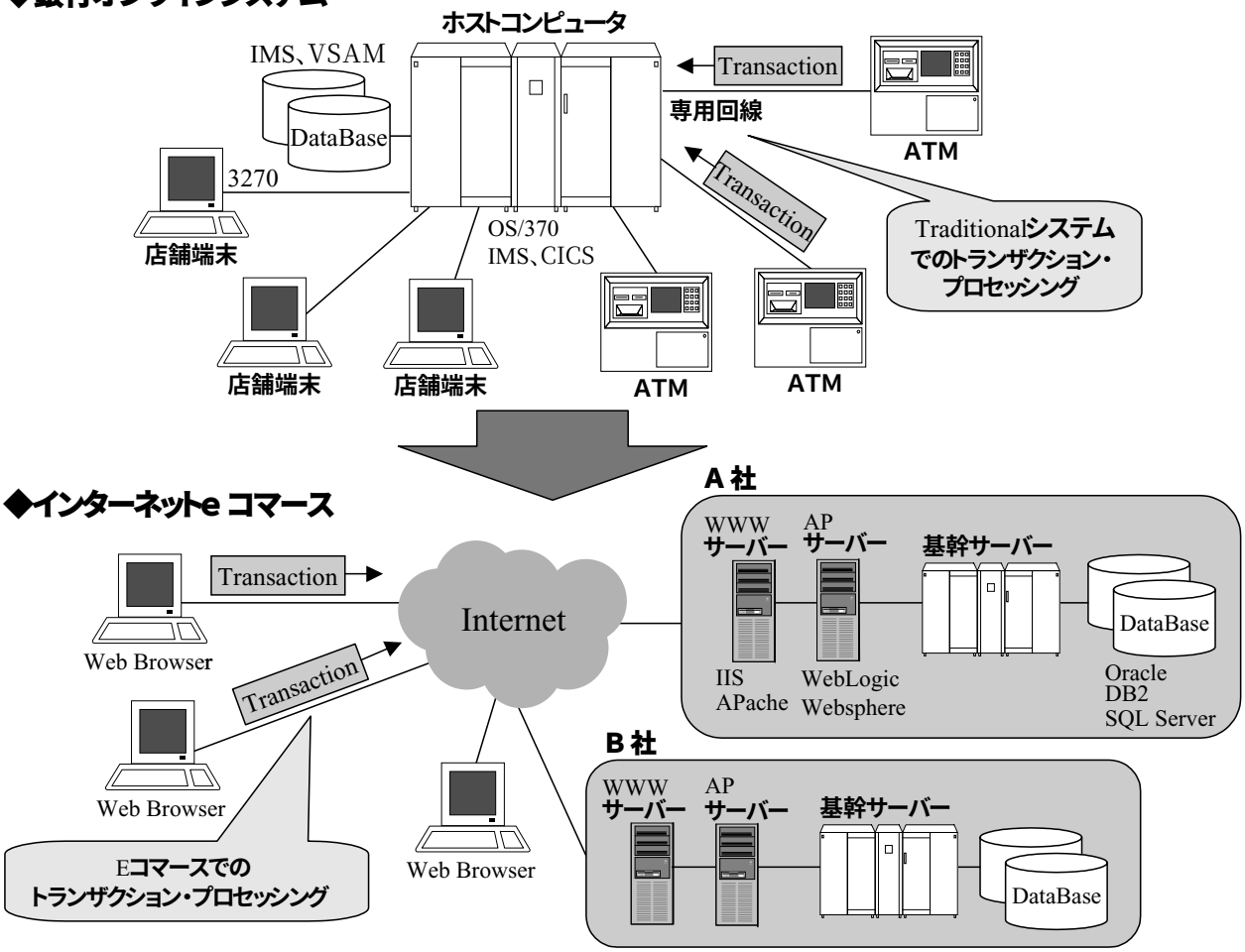



**図1　新旧トランザクション処理の比較**

トごとのセッション管理のためなのか、それとも開発生産性をあげるためのツールとして必要なのかなどだ。

## 3．IT 変遷の 4 時代

まず、トラディショナルと最新の e システムを支える技術を対比してみるために、IT の歴史をひも解いておこう。IT の歴史を、ホストベースのバッチ処理からインターネット Web システムまで 4 世代に分けてみる（図 2 参照）。

処理形態別に時代の変遷を見てみると、ホストベースのオンライン処理では集中システムであったのが、C／S（クライアント／サーバー）では、各部門ごとにサーバーを置き、ユーザー・フレンドリーな分散コンピューティングへと推移した。そして、今日の Web システムは再度、統合／集中化の傾向にある。企業内でもダウンサイジングによって、拠点に配置されたサーバー群が統合されつつある。分散配置によって運用負荷が増大する一方、回線のコストパフォーマンスもブロードバンド時代を迎え、向上してきたからだ。遠隔サイトの LAN（Local Aria Network）化である常時接続も、現実のものになってきた。

また、インターネット経由で大量のトランザクション処理をするため、AP サーバーや DB サーバーをメインフレームに集約する企業も出てきた。Web システムは、ホスト・オンライン・システムにおけるシン・クライアントと、3 層 C／S 処理でのサーバーサイドの機能分担の考え方を合わせ持ったものといえる。

各時代の特徴と、そこで用いられてきた代表的な IT をまとめてみた（表 1 参照）。

まず、「バッチ処理時代」では、ビジネス・アプリケーションの原型ともいえる、バッチ処理の基本 4 パターンである媒体変換、入力チェック、マスター更新＆計算、帳票作成が存在した。ファイル更新のためのマスターとトランザクション・ファイルのマッチング処理はマスター側が DB 化されたため、あまり見られなくなったが、帳票作成のためのキーブレイクやヘッダー、フッター処理は今日もビジネス・アプリケーションの基本ロジックとなっている。

次のオンライン処理では、業務の特性に応じて、非会話型、会話型、そしてバッチ処理を起動して終える非同期型

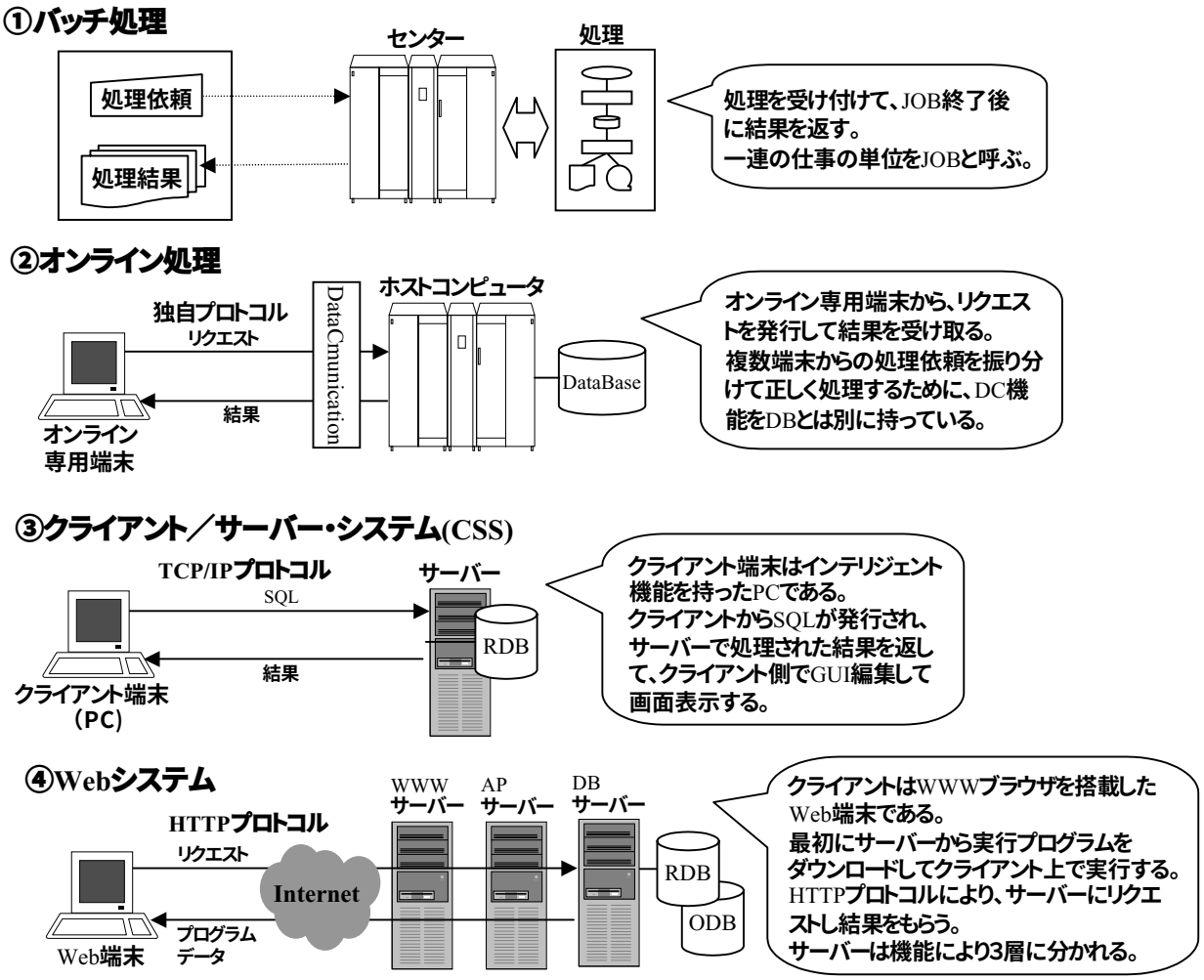



**図２　システム形態から見た IT 変遷**

のパターンを使い分けていた。ホスト・オンライン処理では、画面データをメッセージ・キューとして保持する方式を採っており、アプリケーション・プログラムは、そのメッセージ・キューを読み込んだり、出力したりすることにより画面入出力を行った。この設計パターンは、採用するトランザクション・モニターによって実装方法は異なるが、アプリケーション設計の考え方としては Web ベースのシステムにも充分適用できるだろう。

C／S では、画面１ページと DB のレコードの関係が１対１のケースと１対 n のケース、また受注－受注明細のようなマスター・ディテールの関係でパターン化しておき、DB の構造から画面を半自動で生成するという GUI（Graphical User Interface）ビルダーが多く登場した。これらの技術の背景には、バッチ処理時代の帳票作成のノウハウが活きている。このように、トラディショナル技術が形を変え継承されてきている。

DBMS（DataBase Management System）の登場は、コミット／ロールバック処理、同時アクセス排他制御などの処理を個々のアプリケーション・プログラムから解放した。しかし、初期の階層型 DB では、いったん決めた構造を変えるとアプリケーションへの影響が大きいため、個別アプリケーション処理からの最適化を考慮した設計が必要であった。そのため、論理的な統合よりも物理構造設計に多くの時間が費やされた。その付けが回ってきて、今日どの企業もデータ統合に苦労している。

Web システムでの AP サーバーの果たす大きな役割の１つは、セッション管理であろう。ホスト時代には、端末ごとに論理端末 ID という管理番号を振って IMS などの DC（Data Communication）システム上に定義しておき、トランザクションをどの端末に返すかを制御していた。Web における HTTP のセッション管理も毎回セッションが切れてしまうため、このセッション管理を AP サーバーにゆだねようというものである。

## ４．eシステムの特徴と構成要素技術の層別

e ビジネス・システムが従来の企業システムと大きく異なるのは、システム要件が明確になっていない点ではない

表１　４時代の IT 変遷

| | バッチ処理時代 | ホスト集中オンライン時代 | C／S 時代 | Web 時代 |
|---|---|---|---|---|
| 概要 | 発生データを１日、１週間単位で蓄積しておき、まとめて処理するバッチ処理が中心。 | ファイルアクセスを専用に処理する DBMS、トランザクション処理のための TP モニタがミドルウェアとして登場。 | サーバー側で、データベース処理を中心に行い、クライアント PC 側でメッセージの組み立てと GUI 処理を受け持つ方式。 | 企業内 LAN、WAN またはインターネットを経由してアプリケーションサーバー、DB サーバーで集中して処理が行われる。 |
| 処理形態 | バッチ | 集中、１層 | 分散、２層機能分散 | 集中、３層機能分散 |
| 設計／開発手法 | バッチ処理中心<br>プログラム・ロジックパターン | 構造化分析／設計<br>AP設計とDB設計の一体化<br>企業内での独自の部品化 | RDB 主導設計により ERD 定着<br>DOA の普及 | コンポーネント化、オブジェクト指向設計<br>デザインパターン、ルールベース開発 |
| 主要構成 IT | ジョブ管理<br>シーケンシャルアクセス<br>ランダムアクセス<br>インデックスド・シーケンシャル | 個別ネットワークアーキテクチャ（ex.SNA）<br>データベース管理システム<br>トランザクション処理システム<br>DAM 端末 | オープンネットワーク・アーキテクチャ（TCP／IP）<br>オープン OS（Unix）<br>インテリジェントパソコン<br>リレーショナル DB | インターネット・アーキテクチャ（HTTP）<br>WWW ブラウザ<br>Web サーバー<br>AP サーバー<br>DB サーバー<br>セキュリティ |
| 実装インフラ IT 例 | OS／370、SAM、ISAM、VSAM | SNA、IMS／DB･DC、CICS、ADM、AIM | TCP／IP、Unix、Win／NT･95、Oracle、SQLserver、 | HTTP、Linux、InternetExploerer、NetscapeNabigator、CGI、IIS、WebLogic |
| 言語 | ASSEMBLER、COBOL | COBOL、PLI、４GL | C、C++、VB、PB、Delphi、Access | Perl、Java、HTML、XML |
| 適用領域 | 給与計算、税額賦課計算 | 銀行勘定系業務、受発注業務 | 部門内システム、企業内システム | イントラネット、e ビジネス（対企業、対消費者） |
| 標準化規定 | | CODASYL | SQL、ODBC | JDBC、分散オブジェクト（CORBA、EJB）、XML |
| 利用者 | システム部門、経理部門など一部 | 社内事務センター、専任オペレータ | 社内基幹業務部門 | 社内全部門、社外取引先、他企業、一般消費者 |
| | トラディショナル技術 | | | |

だろうか。ユーザー自身でも何がやりたいのかを明確に定義することはできず、EC（Electronic Commerce）であれば、いったんサイトを立ち上げてからスパイラルに機能を増殖していく、というアプローチが要求される。また、先行者のメリットもあり、アイデアを早く実行した者勝ちというネットビジネスでは、３～６カ月といった短期開発が要求される（図３参照）。

最近、ある流通業のシステム化計画に携る機会があった。ネットビジネスも包括した店舗システムの再構築であったが、半年経ってもなかなか要件が固まらなかった。店頭での物販に関して、立地、季節、催事などを反映しがら、限られた店舗面積の中でいかに売れる商品の品揃えを行なうかがシステム構築の要件となった。さらに、店頭を商品引渡しと代金決済の場とするネットショップ構想や、常連客の囲い込みを図るために、新たな顧客関係管理のための DB 構築構想などが持ち上がってきた。要件定義が難航したのは、企画段階で全てを確定しようとしたところに無理があったからかもしれない。特に、新ビジネスに取り組む場合、最初から完璧なビジネスモデルを作ることは至難の業に近い。実証しながらビジネスモデルを進化させていく必要がある。

e ビジネス・システムを構成する要素をビジネス定義、モデリング、構築／実装、インフラで分類層別してみた（図４参照）。このうち、いくつかのトピックを取上げて、トラディショナル技術と対比しながら見ていくことにする。

## ５．部品化技術

「モジュール化」と呼ばれた部品化は、IBM OS／360 のような大規模 OS 開発で採用され、ビジネス・アプリケーションにも適用されるようになった。from-to の日数計算や年齢計算、チェックディジットのチェックなど、いわゆる共通モジュールと呼ばれるものは、どの開発プロジェクトにも存在していた。これらのモジュールは、ビジネス・アプリケーションでは共通で使用されるため、OS が変わらない限りそのまま使われてきた。そして、プラットフォームや言語を変えながら、ほぼ同機能のものが作成されていった。

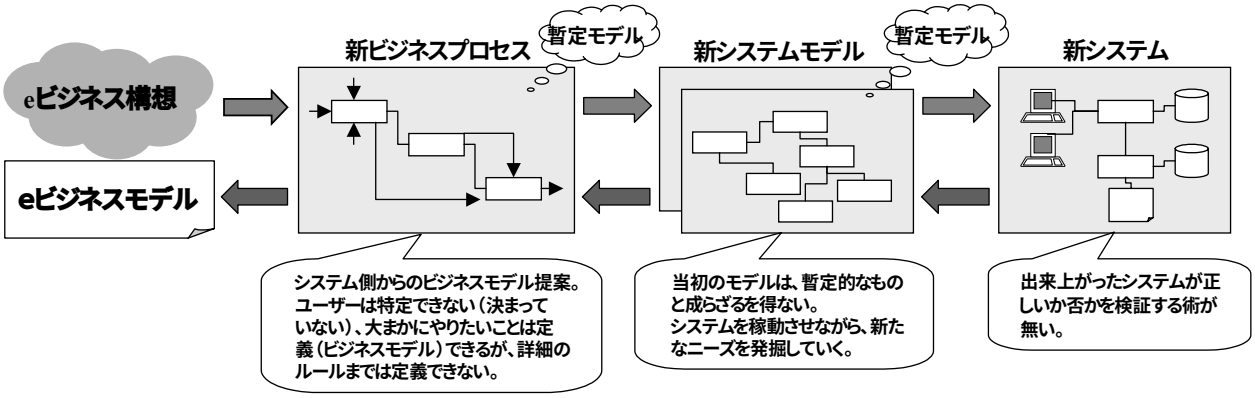



**図３　e システムの特徴**

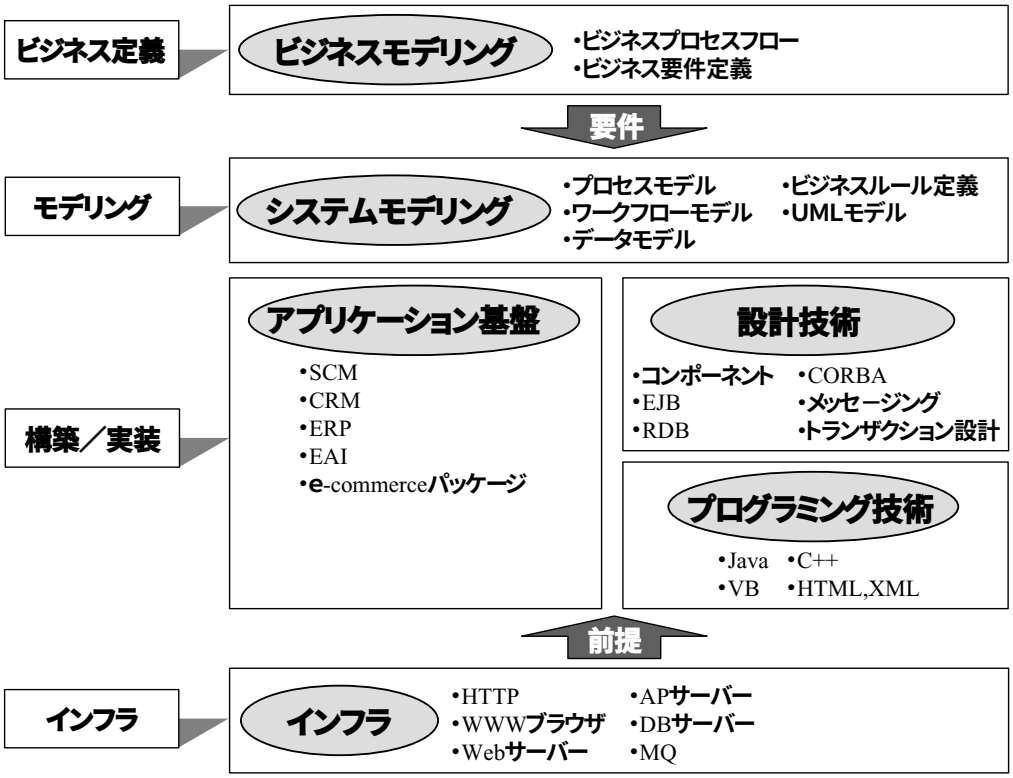



**図４　システムの構成要素技術の層別**

そして、例えば利率計算や納入予定日算出などの販売系、会計系、人事／給与系など、業務固有で現れる機能へと拡充していった。これらは、特定の業務システム内で閉じており、業務共通モジュールとも呼ばれる部品だ。さらに、モジュール化がシステム間で流通されるようになったものが、アプリケーション・パッケージだ。

Ariba、CommerceOne などの B２B パッケージは、EJB[*3]（Enterprise　JavaBeans）ベースのオブジェクト指向技術で構築されている。これらのパッケージをカスタマイズすることは、新たなクラスを作成し、プラグインしていくことである。

実行可能形式のものを呼び出して利用する部品とは別に、ソースコードの一部に組み込んで利用する部品も作られた。バッチ処理におけるファイルアクセスの処理パターンやオンラインでの画面ハンドリングパターン、DB アクセスパターンなどである。COBOL のコピーライブラリとして登

＊３）EJB : Java をビジネス・アプリケーションで活用する際にサーバー側に必要な機能をまとめた Enterprise　Java 仕様の一部で、Web サーバーなどに実装されている。

録しておき、ユーザープログラム内にコンパイル時に呼び出したり、PL／I[*4]（Programming Language／Ｉ）ではマクロ展開して使うなどして利用した。いささか乱暴ではあるが、オブジェクト指向でのコンポーネント（クラス）は、このパターンマクロの考え方に近い。クラス定義自身は実行できないが、定義（オブジェクトを生成）することで利用可能になる。

粒度を小さくすると部品化が容易になるが、数が増えることで管理や検索利用が煩雑となり、徐々に使われなくなる傾向にある。逆に粒度が大きいと汎用的ではなくなってくる。ボトムアップによる部品化技術と業務パッケージで提供されているようなトップダウンでのモデル化アプローチを、うまく融合させていくことが厳しい条件下にあるｅビジネスシステム構築には有効な方法となる（図５参照）。

## ６．DOA vs UML

EC、Ｂ２Ｂパッケージでは、オブジェクト指向設計で構築されているものが多く、これらをカスタマイズしていく上でも、オブジェクト指向はこれからの必須技術といわれている。では、DOA[*5]（Data Oriented Approach）の考え方に基づくデータ・モデリング技術は、オブジェクト指向のもとでは不要となり単なる過去の技術となってしまうのだろうか。

オブジェクト指向では、切り出した業務のドメイン単位で、オブジェクトがデータと振る舞いを合わせ持って、正常な処理が行なわれることを第一義とする。つまり、対象とするシステムが最適化された部品を組み合わせて動くことを目標としている。

ところが、DOA では対象システム以外からデータが利用されることも想定して設計を進める。ゆえに、システム

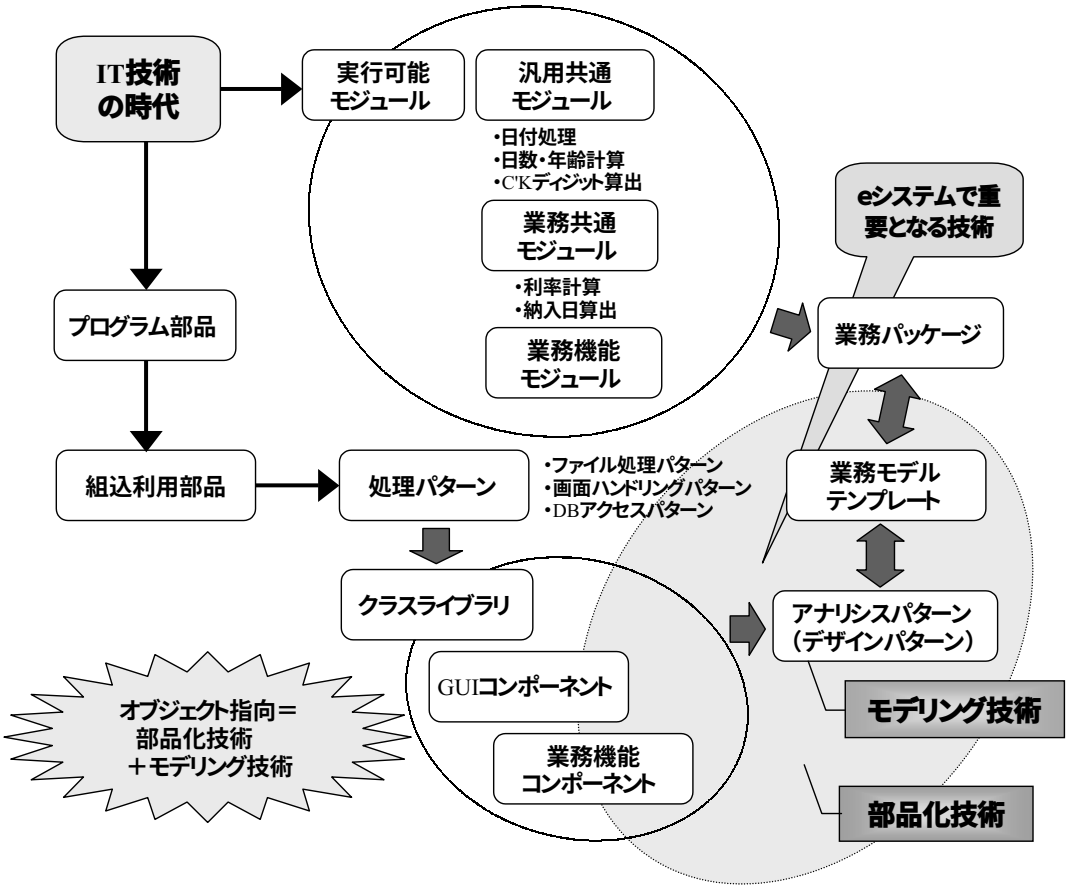



**図５　部品化技術の変遷**

＊４）PL／I：1960年代に IBM 社によって開発されたプログラミング言語。あらゆる用途に耐える汎用の言語として、すべての言語を置き換えるべく開発された。

＊５）DOA：データ中心設計のこと。情報システムにおいては、機能よりもデータの方が安定的であるとの考え方に基づいて、データの分析／設計をできるだけ先行して行なおうとする設計／開発の考え方。

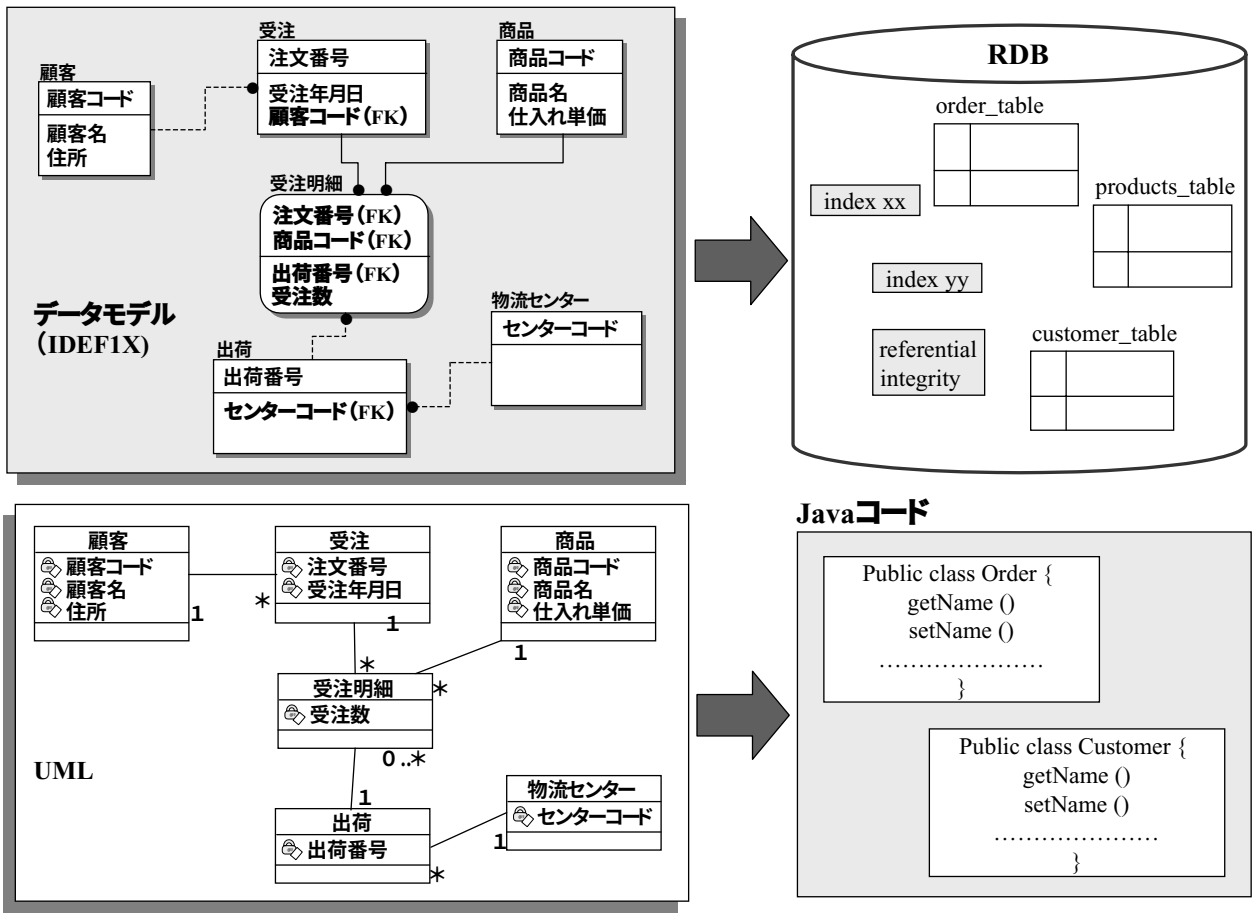



**図6　各モデラーの描く物理実装の相違**

内で共有されるリソース系データ（いわゆるマスター）については、他システムからの利用、あるいは共用も考慮して分析することになる。例えば、電子市場で取り扱う商品、それを売り買いする企業、取引形態など、システム全体としての管理体系を決めてからイベント系の分析に入るのが理想だ。個別の業務単位で考えるよりも、全体としてどのような体系にするのか、属性をどこまで管理するのかを決めてから、個別のケースで検証していくやり方が採用される。

オブジェクト指向で設計しても、永続オブジェクトはRDB（Rerational Database）として実装されることがほとんどである。そのため、クラス設計と平行して、RDB実装を意識したデータ・モデリングが必要となるのが実態だ。現時点でのUML[*6]（Unified Modeling language）モデリング・ツールでは、RDB実装上、validationルールやconstraintルールなどの設定ができないなど完全ではないため、別途ERD[*7]（Entity Relation Diagram）ベースのモデリング・ツールが必要となる。クラス図をベースにERDに展開するか、最初からクラス設計と並行してデータ・モデリングを進めるかのやり方が考えられる。業務分析フェーズで、システム全体を鳥瞰できるような概念データモデルを作成し、その後に設計フェーズで業務ドメイン単位でユースケース[*8]、クラス設計を行なうと同時に、全ての属性を加味したデータ・モデリングを並行して進める、というのが筆者の考えである。

レビューなどで、データモデルとUMLベースのクラス図で、議論がかみ合わないことが、しばしばある。クラス図の最終ターゲットはJavaのプログラムであり、物理に近づけば近づくほど、RDBのテーブルをターゲットとするデータモデルとはかけ離れてくるのかもしれない。ある論理イメージまでは、クラス図だろうが、データモデルであろうが、扱うコンテンツは同じで表記法だけの違いだと思うのである（図6参照）。ゆえにDOAをベースとしたデータ・モデリングは、設計以前の上流で有効であるし、実装設計でもRDBが主流である限り必要な技術と考える。

---

＊6）UML：オブジェクト指向分析／設計に関するノーテーション（記号法）と用語を定義した統一モデリング言語。

＊7）ERD：エンティティと、その構成要素である属性、そしてエンティティ間の関連を図示したもので、データモデル図と同義。

＊8）ユースケース：システムの使用方法に対する特定の形式、パターン、使用例、シナリオのこと。

## 7．XML とデータ整備

XML[*9]（eXtensible Markup Language）スキーマを使えば、型定義や属性ドメイン（定義域）が定義できるようになる。さらに、属性ドメインや一意性制約、キー参照／制約などが定義でき、RDB のスキーマ定義に近づいている。また、XML は基本的に階層構造によってデータを表現するため、階層型 DB と似ているが、XML ではメタデータを一緒に保持している点が異なる。

このように、XML は RDB のスキーマ定義に類似しており、その設計には伝統的な ERD モデリング手法が使える。但し、実装のためには、階層構造を加味して物理モデルとしては若干の変形が必要になってくる。

XML の利用としては、当初のテキストデータ以外の文書管理の目的から、B２B 接続の手段として注目されている。このような、XML スキーマを活用したデータ流通を実現するには、データの標準化と部品化が鍵となる。XML はスキーマを階層構造で表現できるため、部品化が容易と考えられる。例えば、図７で「配送先」と「請求先」は共に氏名、郵便番号、都道府県名、市区町村名、番地からなる。

これらをまとめて「住所」とし、その内訳の要素属性を定義して、配送先と請求先に割当てることができる。また、任意の型を定義できるため、型定義時にバリデーションを設定することもでき、アプリケーション個別のチェックが不要となる。

タグに付す名称については、取引企業間で異音同義語や同音異議語が発生しないように注意しなければならない。そのためには、まず自社内で用語を定義した辞書を整備し、それを企業レベルからユニバーサル・レベルへと繋いでいく努力が必要である。RosettaNet や ebXML、Ariba の cXML、Commerce-one の xCBL など、業界標準や MP（Market Placement）標準の XML スキーマが整備されつつある。スキーマ構造に加えて、データ要素名（データ項目名）を構成するための用語定義、データ要素名の意味内容、データタイプ、ドメイン定義などの整備が必要になる。さらに、エレメントを組み合わせて、再利用可能なエレメント構造をスキーマ部品として登録しておく。さらに、これらの登録集を基に XML スキーマやインデックスが作成される。そして、各参加企業自身の XML と MP-XML と

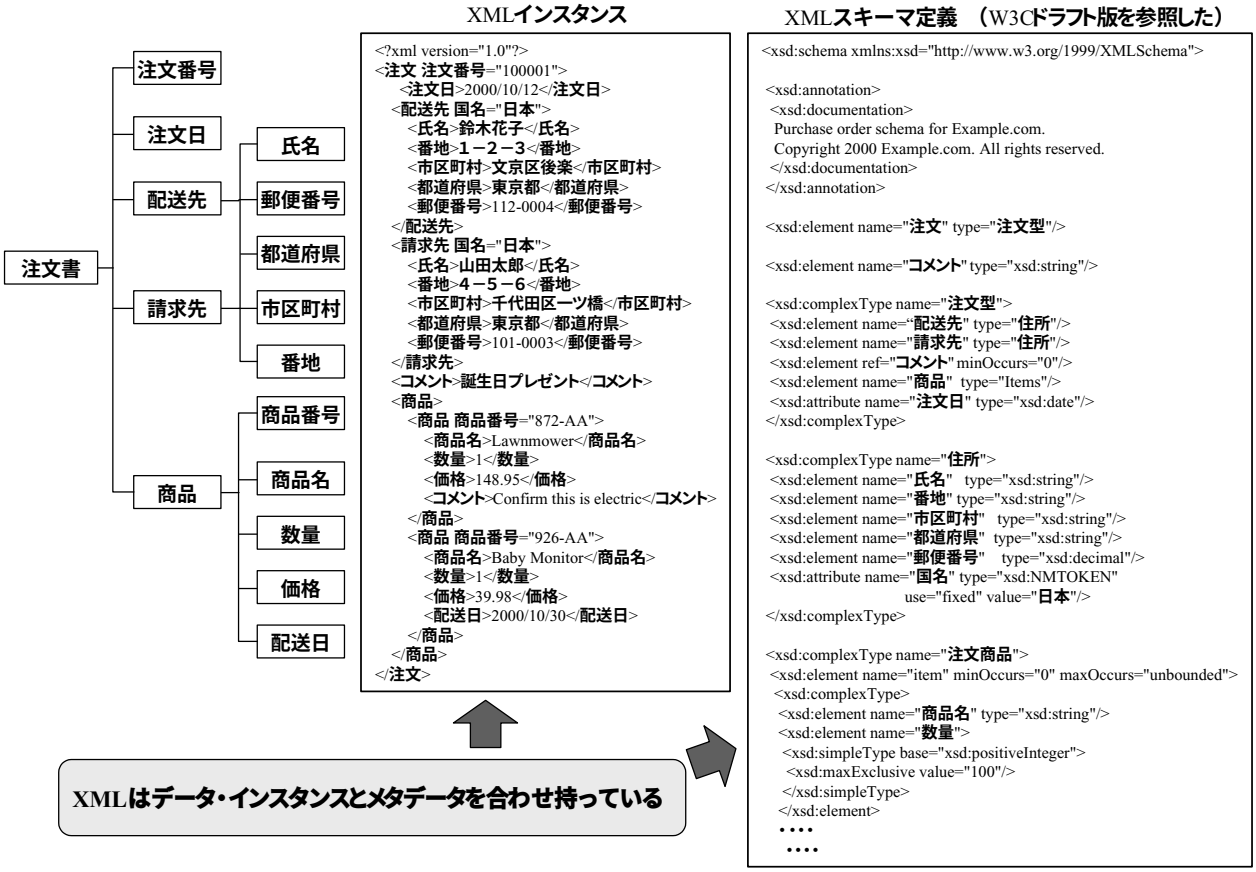



**図７　XML 構造**

＊９）XML：インターネット環境で利用することを前提に開発され、HTML と同等以上の優れた機能を持つマーク付け言語。

の間でデータ交換が行なわれることになる。その際、お互いデータ要素辞書を持っていれば、どのタグとどのタグが対応するか判断できる。そして、お互いの変換ルールはXSLT[*10]（XSL Transformation）で記述しておく（図８参照)。

## ８．基幹システム連携

B２B システムで最も難しいのが、企業のバックボーンの基幹システムとの連携だ。自動車メーカーによるオークション形式の部品調達サイトがあったとしても、発注システムや供給側企業の既存システムと連動できなければ、無駄なプロセスが発生してしまう。例えば、MP で部品をオーダーする際、自社の購買・発注システムと連動していなければ、購買計画を作成した後で、MP のシステムに対して二重入力しなければならない。基幹システムへの影響を読みきれず連携を断念し、ERP（Enterprise Resource Planning）で再構築する、といったケースも発生しているようだ（図９参照)。

基幹システムとの連携を難しくしているのは、各システムが異なったアーキテクチャのもとに、異なったインフラ上で稼動していることが挙げられる。時代と共に主要 IT が変化してきているため、技術者のシフトも進み、トラディショナル技術を習得した技術者が少なくなってきていることも足かせとなっている。このために、トラディショナルと最新の技術をラッピングし、異なったアーキテクチャ間をつなげよう、というミドルウェアが EAI（Enterprise Application Integration）である。

## ９．EAI からビジネスプロセス統合へ

EAI 登場以前のアプリケーション統合では、基本的にはデータ転送により各システム間のインタフェースを実現していた。まず、ターゲットのシステム化領域について、データモデルとシステム機能を定義するのがアプリケーション設計の常套手段であった。続いて隣接するシステムとの境界となるインタフェースを定義し、隣接システムに渡すデータと貰うデータを決める。後は、リアルタイムか日時／月次のバッチかなど、データ授受のタイミングとファイル転送／メッセージ連携などの授受方式を決めれば良い。

EAI とはアプリケーション間を接続するアーキテクチャであり、その基盤となるテクノロジーは多岐にわたる。メインフレームのトラディショナル・アプリケーション、パッケージベースの ERP、C／S システム、そして最新の Web システムを構成しているテクノロジーなどが総動員される。その実現手法を４つのレベルに分けてみる。

①データレベル

DB 間でデータ・レプリケーションしたり、お互いのシステムで所有するデータを、ビューなどを介して仮想的に共有するやり方である。

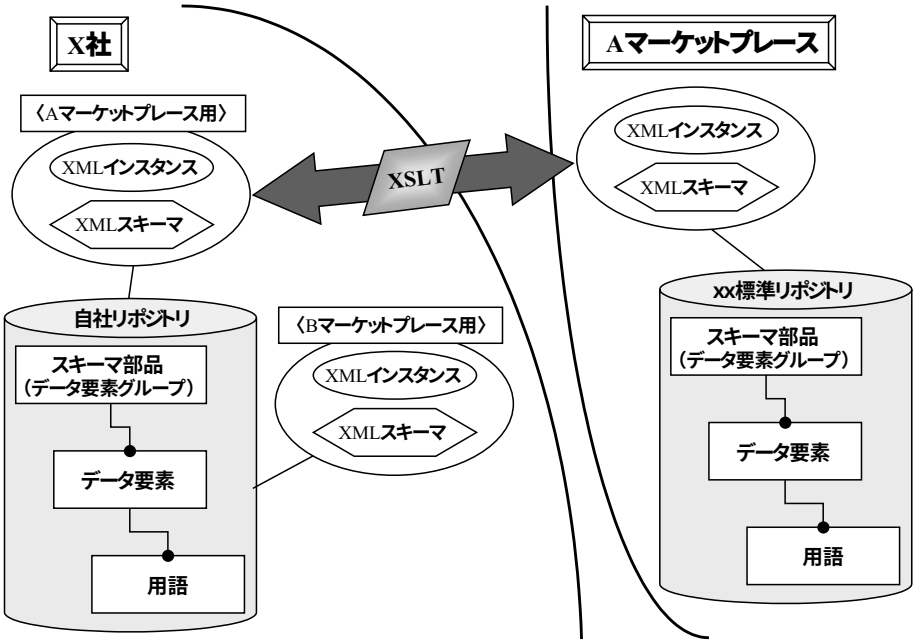



**図８　ユニバーサルデータ連携**

＊10）XSLT : XML（eXtensible Stylesheet Language＝文書の構造を記述する言語）の変換用スタイルシート。

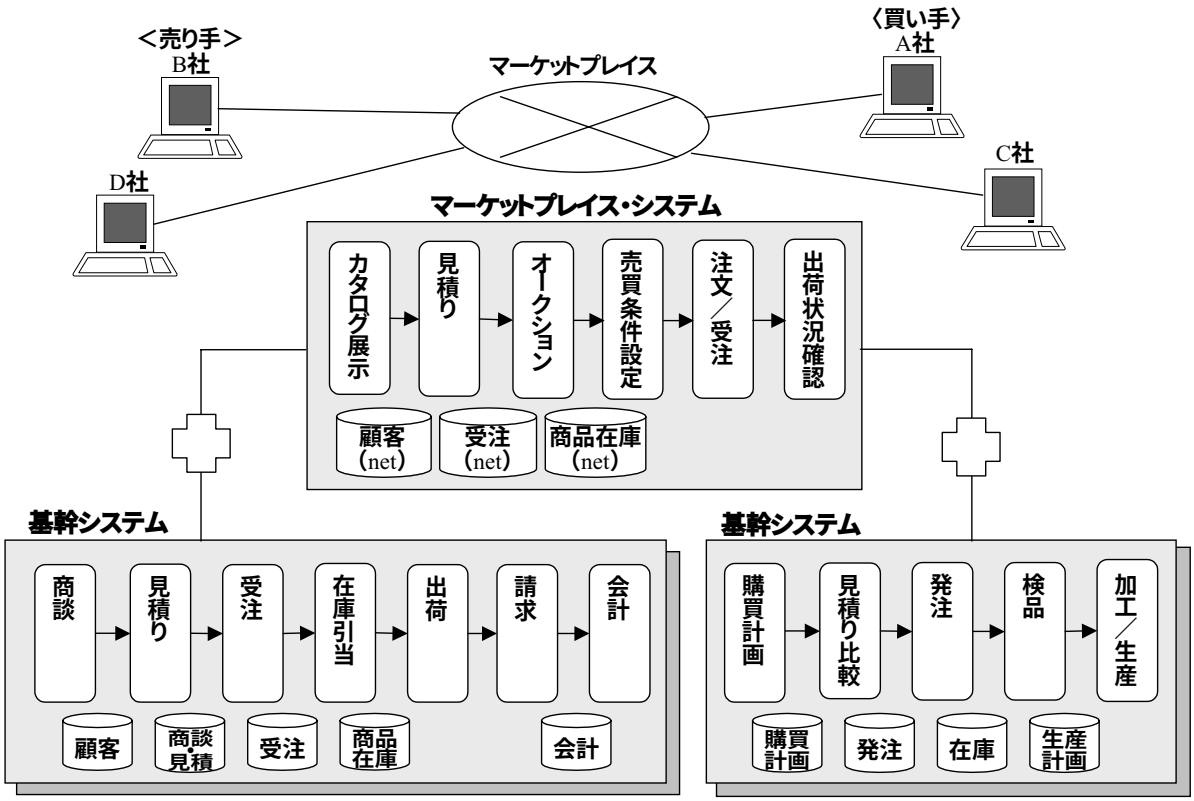



**図9　基幹システム連携モデル**

②メッセージレベル

MQ Series[*11]やCICSなどメッセージ・キューイングやトランザクション・モニターによるメッセージレベルで交換する方法である。

③アプリケーション・インタフェース・レベル

お互いに公開されているAPI[*12]（Application Program Interface）を用いて、データをアクセスするやり方である。CORBA[*13]（Common Object Request Broker Architecture）やEJBの共通コンポーネントを介してのインタフェースもここに含まれる。

④ユーザー・インタフェース・レベル

CUI（Character User Interface）のホスト・アプリケーションやC／Sアプリケーションを、Webブラウザをフロントエンドにして透過的に見せるやり方である。加工処理したユーザー・インタフェース・レベルでの変換になるので、バックエンドのデータ構造を意識しないでアプリケーション統合ができるため、多くの企業でレガシー・アプリケーションの再生策として用いられている。

次に、アプリケーション統合の進化であるが、データ交換／転送レベル→データ共有レベル→プロセス統合という順に密結合していく必要がある。第二段階のデータ共有レベルでは、全体を見渡した重複のないデータモデルを描いた上に、個別のアプリケーションを構築するために、全体のプロセス（ワークフローといってもいい）としては、重複や寸断が発生したりする。このために、全社レベルでのプロセス統合が必要となってくる。

統合プロセスモデルをベースとして、ワークフロー制御のみを新たに構築することにより、MQやTP（Transaction Processing）モニター、EJBなどのミドルウェアを介して、レガシーのERP、ホストベースのアプリ、C／Sでの部門アプリケーションが利用できる。このような、ビジネスプロセス・マネジメントの考え方が企業内EAI、そしてB２Bへの進出で必要とされる。既存の資産、技術インフラを継承しつつ、統合化を目指す方向である（図10参照）。

---

＊11）MQ Series：単一のAPI、非同期処理により、分散して構築された複数の自動化システムを統合し、一貫した開発を可能にするメッセージング・ミドルウェア（IBM製）。

＊12）API：ユーザー・プログラムから、パッケージやミドルウェアなど、他のシステムを呼び出すためのプログラム間インタフェース。

＊13）CORBA：オブジェクト指向の分散処理環境を実現するための規約。

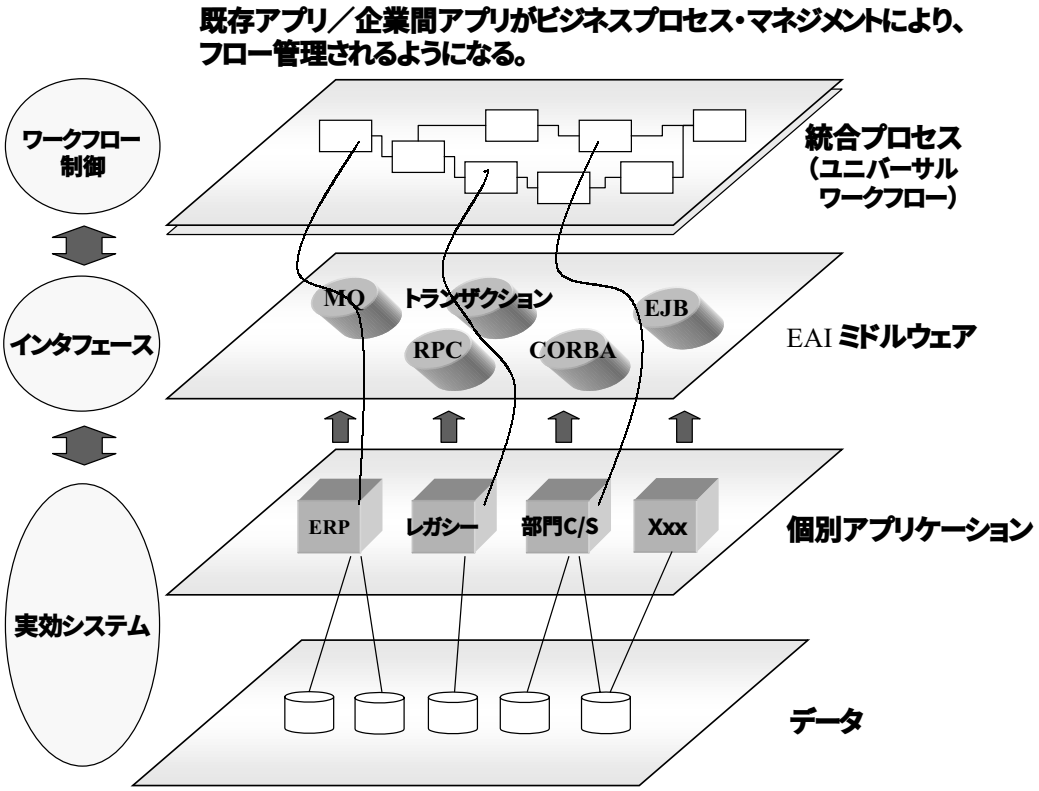



**図10　EAI レイヤ**

## 10. COBOL vs Java

Web システムの構築案件が増えるのに伴い、「Java 技術者が足りない」という声を良く耳にするようになった。ここでいう Java 技術者は、多岐にわたる意味を持っている。Java のプログラミングができればいいのか、J2EE（The Java2 Platform, Enterprise Edition）準拠の AP サーバーの知識を持ち、EJB コンポーネントが作れることなのか、または、UML ベースでの設計ができることなのかというと、どうもそれらを総称しているようだ。

対比する純トラディショナル言語といわれる COBOL は、Y2K 問題の騒動を終えて、すっかり化石化したように思えるが、実際には全世界で300万人のプログラマが現存する最大勢力である。しかし、世の中はマスコミの煽りもあってか、プログラマの総 Java 化へ動いているようにも思える。一方、かつてのメインフレーマを中心に、COBOL コンソーシアムを設立して Web アプリケーションでも積極的に COBOL を使っていこう、という活動が開始されている。ここまで COBOL にこだわるのはなぜか、そして今後 Java でなければならない必然性は何か、どうやら理由には単なる言語を越えたものがあるようである。

もともと COBOL は、事務処理計算用に開発された言語で、ファイルアクセスに強みを持っていた。しかし、DBMS の登場によりファイルアクセスは COBOL から離れ、画面処理などのユーザー・インタフェースも他のミドルウェアが受け持つようになり、結局、COBOL 本体に残ったのはビジネスロジックの部分だけである（図11参照）。

Java は、言わずとしれたオブジェクト指向言語であるため、クラス継承やメソッドの呼び出しが容易にできるようになっている。COBOL でもメソッド呼び出し的なコーディングは可能であるが、分岐が発生したりして、非常にメンテナンスしにくいものになるだろう（図12参照）。

COBOL は、40年にわたってビジネス・アプリケーションを記述してきた。そのノウハウは、COBOL の中に染み付いている。COBOL で培ったビジネス・アプリケーションのコーディングは、企業内で脈々と生きつづけており、それを使ってきた技術者の中にも叩き込まれている。このような、COBOL から継承されたビジネスルール記述のノウハウやモデリング技術は、Java に置き換わっても継承していかなければならない。そして、Java からはオブジェクト指向設計、実装環境を選ばない JavaVM[*14]（Java Virtual Machine）の新しい考え方が、e ビジネスのインフラに反映されて拡充していくことになろう（図13参照）。

---

＊14）JavaVM：プラットフォーム・フリーの Java 言語で作ったソフトを動かすための土台となるプログラム。

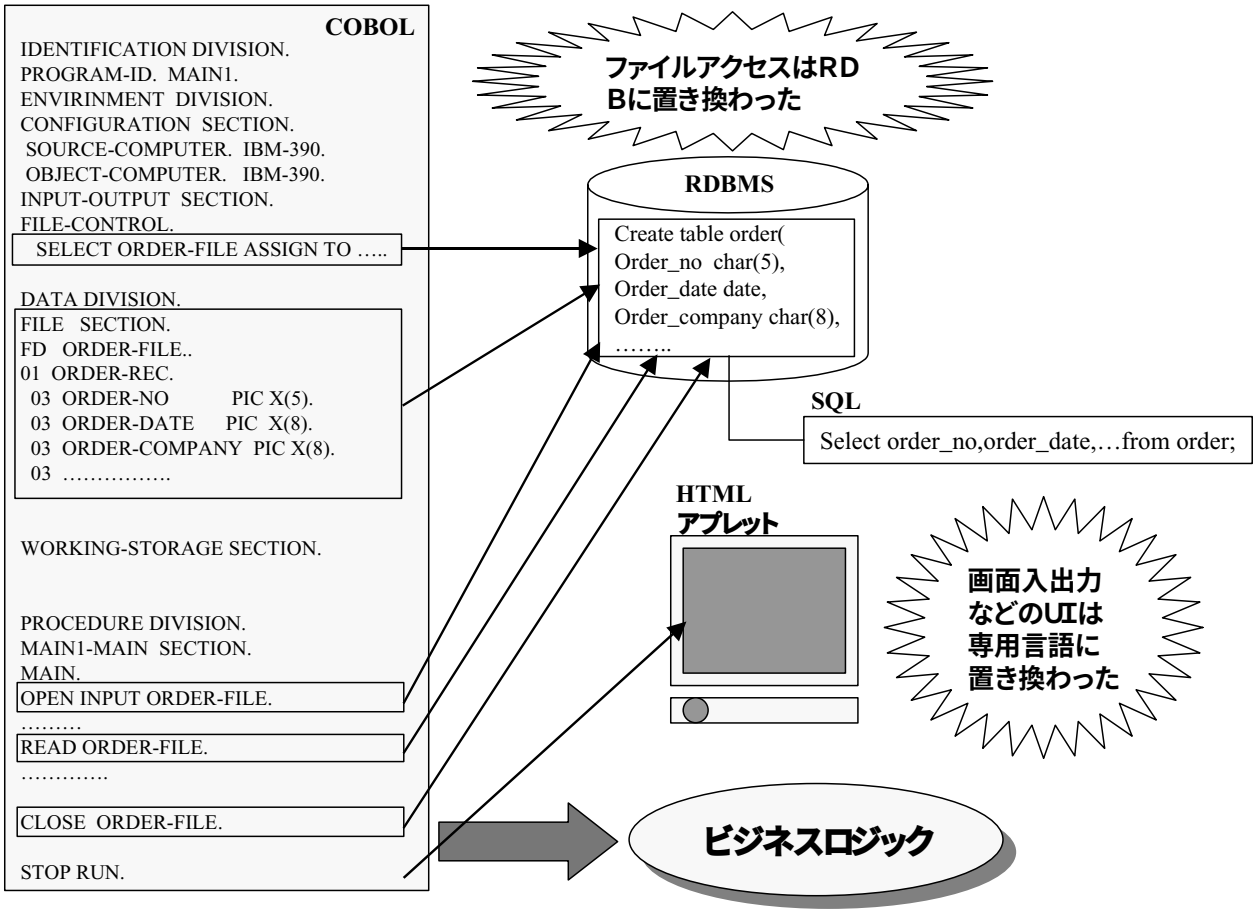



**図11　COBOLの構造と変遷**

物理的な言語としては、Javaの方が現状のテクノロジーに適合したものであり、今後も確実にJava化へシフトしていくと思われる。しかし、多くの企業が2000年以降にも引き継がれた膨大なCOBOLソフトウェア資産を抱えており、さらに、今までの設計ノウハウを有した技術者のシフトが済むまで、当面はCOBOLとJavaとの共存が続くであろう。

## 11．B2Bでのアクセス管理

今までの企業内システムでは比較的ルーズだったセキュリティの仕組みは、eシステムにおいては避けて通れない重要な技術である。ここでは、アプリケーションの設計レベルでのセキュリティともいえるアクセス権限について考えてみる。

アクセス権限は、メニューレベル、データアクセス・レベル、機能レベルの3段階で設定するケースが多い。まず、メニューレベルはあらかじめアクセスが許可されたユーザーのみにメニューの選択を許可するものである。権限設定を行なう場合には、社内組織構造とは別に、ユーザー単位や組織単位で業務を遂行する機能である業務ロールを設定する。この業務ロールとアプリケーションの交叉エンティティとして、許可されるアプリケーション権限を設定できる。次のユーザーデータへのアクセス権限は、DBMSのセキュリティ機能でも読み書きの制約は設定できるが、インスタンス（レコード）レベルでのアクセス管理はできないため、特に制限を設ける必要のあるものについては、業務ロールとユーザーデータのレコード識別情報の交叉エンティティで管理することができる（図14参照）。

B2Bなどの企業間システムになった場合の考え方は、企業内でのモデルを拡張することで対応できそうだ。個別の企業や取引企業の納入業者／販売顧客といった役割である企業ロールを、社内システムのアクセス管理対象に付加することにより実現できる。例えば、ある企業がMPに参加しようとした場合、立場が買い手か売り手かによって、アクセスできるアプリケーションは異なる。また、同じ買い手でも特定の企業間で公開されるデータもありうるからだ。

また企業内のアクセス管理をシングル・サインオンにシフトしていったように、B2BでもMPに参加するようになるとアカウント管理が課題となる。社内システムと同じアカウントで社外システムにもアクセスできることが理想だ。例えば、購入したい商品のカタログが社内シス

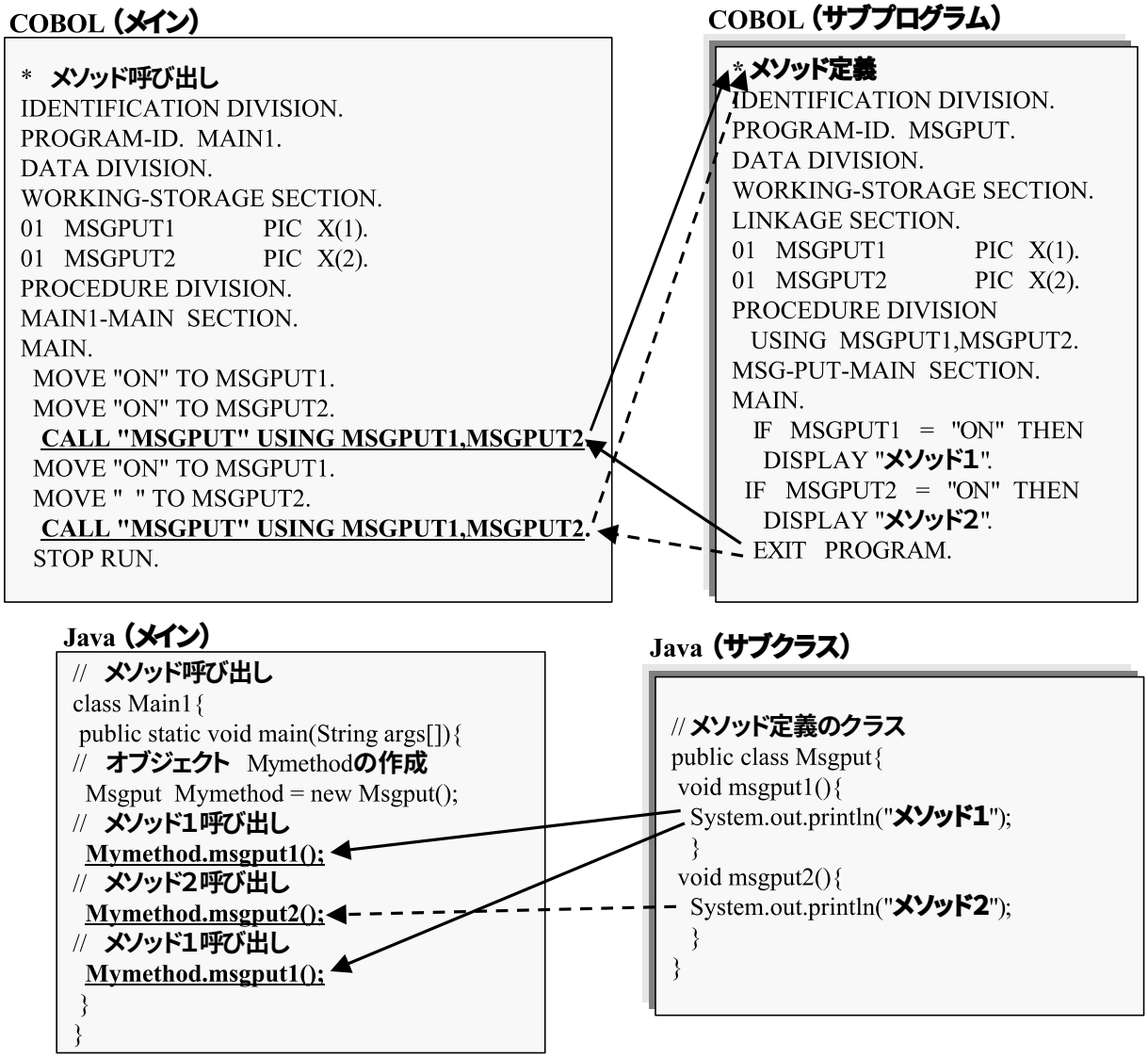



**図12　COBOL と Java のスタイル**

テム上にない場合に、商品供給元サイトに掲載されているカタログを直接参照する際などに、社外へアクセスしていることを意識しない設計が望まれる（図15参照）。

B２B ではセキュリティ問題を考える際に重要なのは、データの漏洩や機密保持よりも、データをどこまで公開するのかという点である。例えば、製造業の場合には生命線でもある生産計画データ、また、販売／卸業にとっては生産計画や販売実績といったデータを公開していかないと、SCM（Supply Chain Management）や e プロキュアメント・システムは成り立たなくなる。かといって、特定の利用者や企業以外からはアクセスできないように、インフラやアプリケーションが守られていなければならない。

このように、データ公開やデータ・セキュリティに関するコンセプトをデータ管理標準で規定し、その考え方に則った実現方式を策定することが重要となる。従来のデータ管理標準は、どちらかというと企業内でのシステム開発や運用に主眼を置いたものが多かったが、今後は企業内でのデータ活用や外部企業とのデータ取引のための企業指針を定義したものに変えていく必要があるだろう。例えば、データ公開に関する指針、公開先、公開データ内容、外部データ利用時のネーミング規約、公開先企業へのアクセス権限などである。

## 12．おわりに

本稿では、連載記事の中からポイントをかいつまんで紹介してきた。ここで、今までのポイントを整理して以下に列挙しておこう。

・大規模トランザクション処理の原型は、ホスト・オンライン処理にある。
・IT 変遷の傾向を見ると、設計の考え方は実装方式を変えながら継承されている。

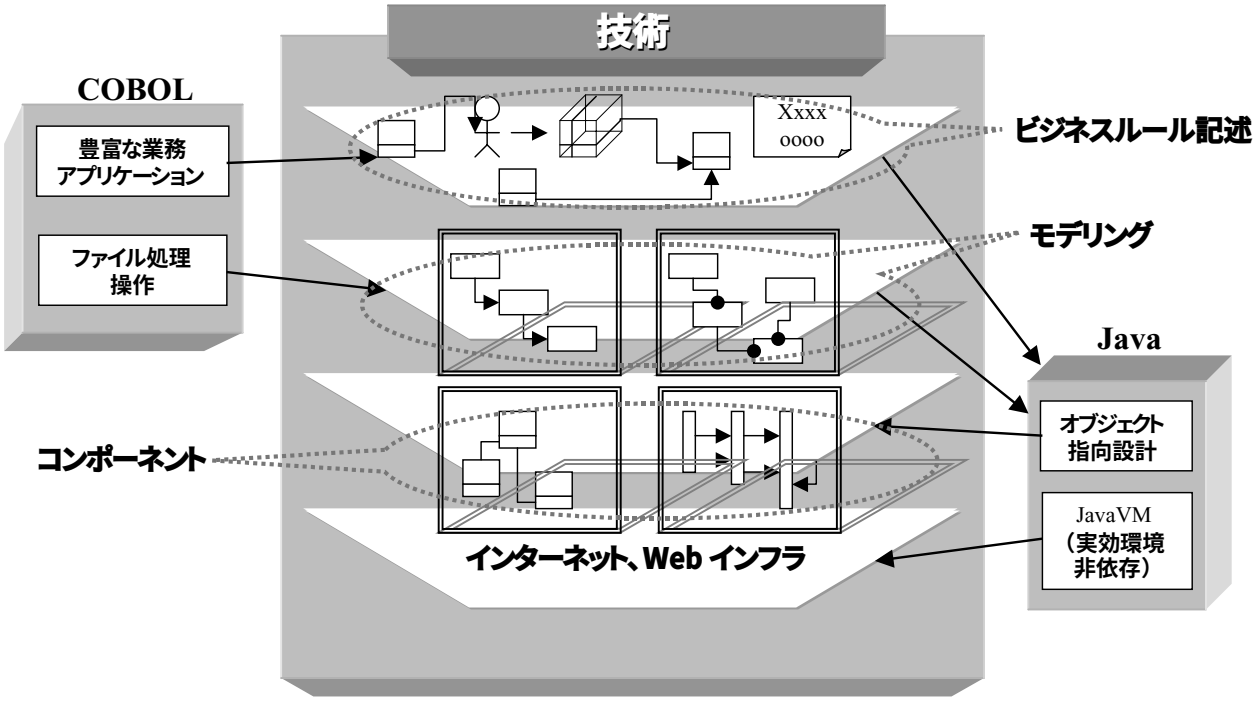



**図13　COBOL からの継承と Java での創出技術**

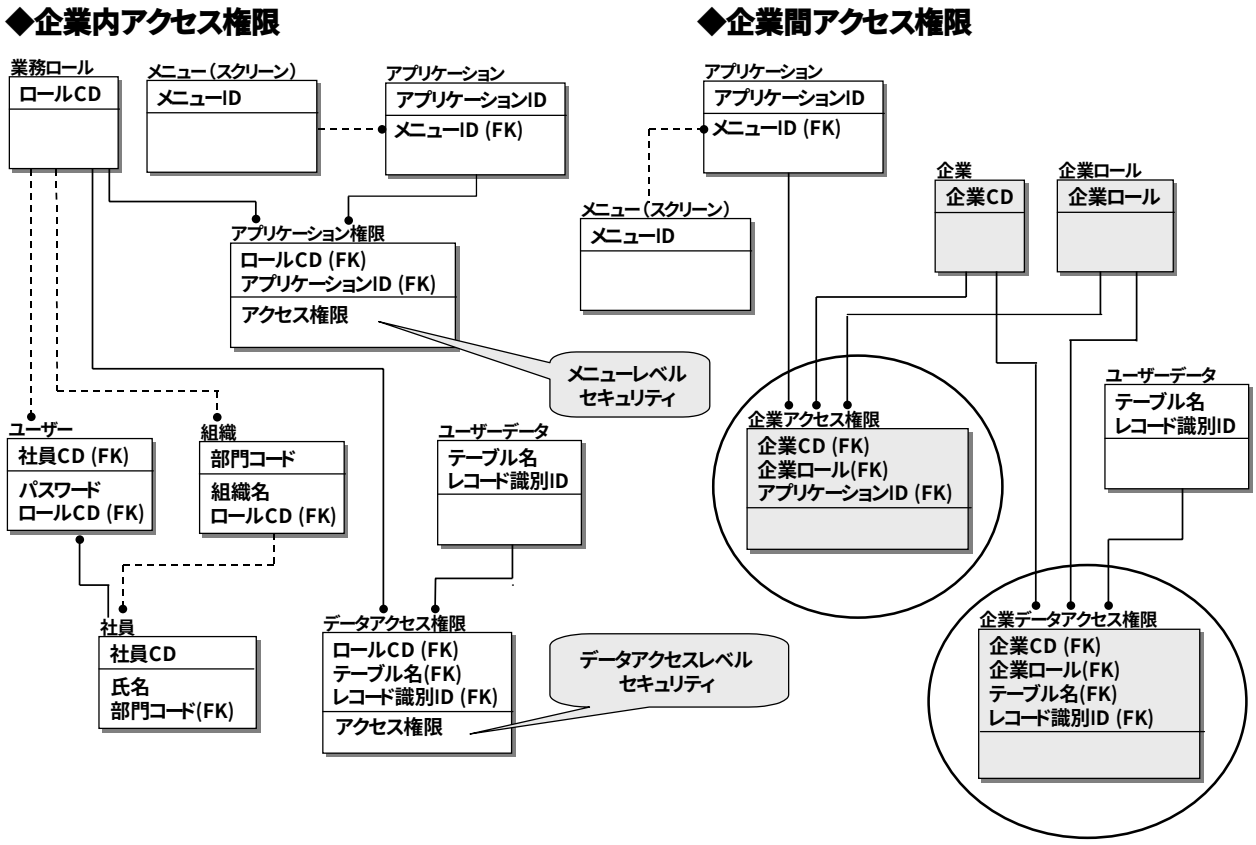



**図14　業務セキュリティ・データモデル**

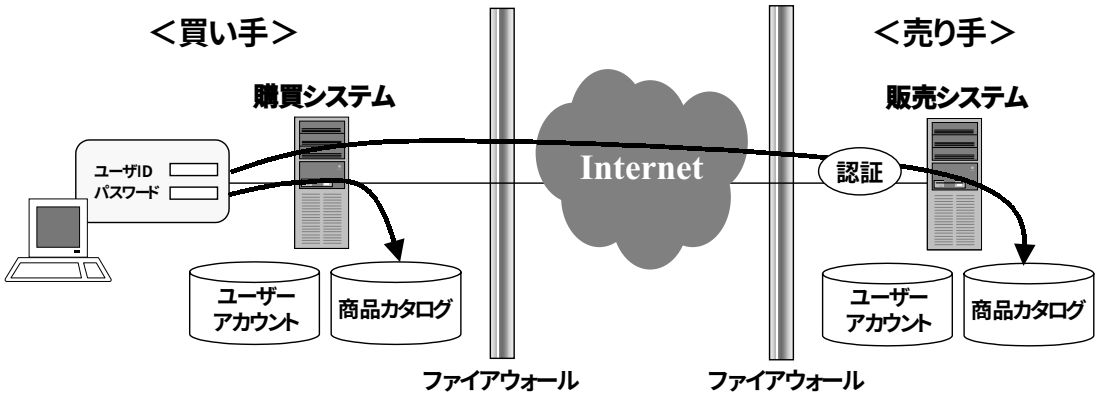



**図15　B2B セキュリティ**

・今後、オブジェクト指向設計は重要な技術となることは確かだ。しかし、そこにいたるビジネスプロセス・モデリング、管理対象を見抜くデータ・モデリングの技術は相変わらず必要である。
・XML による企業間データ流通促進のためには、データの標準化整備を避けて通れない。
・B2B では、基幹システム連携が重要課題だ。連携のためのトラディショナルと e システムのラッピングに EAI 技術を利用する。
・B2B では、データ統合だけではなくビジネスプロセス統合が望まれる。
・Java、XML、AP サーバーなど、最新 e テクノロジーの根源には COBOL アプリケーションで培った技術が継承されている。
・企業間 SCM や eMP によるデータ公開、さらに、アクセスのための権限などを盛り込んだデータ管理標準の整備が必要だ。

e ビジネス・システムというと、Web をベースとした Java、XML、AP サーバーなどの技術が取りざたされている。これらの新しい技術は確かに革新的で必要だが、アプリケーションの分析／設計という中核技術は、昔も今とそれほど変わっていない。e ビジネス・プロジェクトへの要員アサインを見ると、最新技術を習得した技術者を中心に集めてスタートするケースが多いようである。その結果、業務要件からシステムモデルへの変換において、アプリケーション設計の経験不足から失敗を招くケースが発生している。

パッケージ・インテグレーションの発展に伴い、いわゆる泥くさい設計技術が中抜きとなってしまい、柔軟な発想と対処ができる技術者が育っていかない、ということも言えるのではなかろうか。それを教育で補っているか、というと全く逆である。今日の情報処理教育は、ベンダー資格を取得するためのカリキュラムに基づいたコース制が幅を利かせている。そのような教育だけでは、即戦力は期待できるが、応用の利く技術者は育ちにくい。

B2B パッケージでは、オブジェクト指向で設計／実装されているものが多い。IT 技術者は、インフォメーション・エンジニアリング習得時のように、オブジェクト指向設計を身に付けることが必須であろう。しかし、オブジェクト指向設計にいたる以前のビジネスプロセス・モデリングや、全社を見渡したエンティティを切り出すデータ・モデリングの考え方は、ビジネスからシステムへの橋渡しを担った不変の設計技術として、将来に渡って継承していかなければならない、と改めて思うのであった。

最後に、連載を通じて考えをまとめる機会を与えていただきました『月刊ソリューション IT』の編集者の皆様に感謝いたします。

## 〈参考文献〉

1．真野　正：『トラディショナル IT からひも解く e 技術』(「ネットワークコンピューティング」，リックテレコム社)，2000.6～2000.12連載
2．真野　正：『トラディショナル IT からひも解く e 技術』(「月刊ソリューション IT」，リックテレコム社)，2001.1～2001.5連載

〈月刊ソリューション IT URL〉
http : //www.ric.co.jp/sol/index.html