1996.3 JIPDEC. レポート。

1．技術動向

1．1　エンドユーザコンピューティングの動向　(中村)

マルチメディアやインターネットというキーワードに代表される情報化社会の進展にともない、新しい時代に対応した新しいアプリケーションの作り方が模索されており、エンドユーザ主導のアプリケーション開発の時代が迫っているという印象が強い。本節では、幾つかの視点からエンドユーザコンピューティングの位置付けを試みる。

1．1．1　背景

現在、広く利用されているコンピュータのアーキテクチャの基本となっているプログラム内蔵方式が確立して、ほぼ５０年になろうとしているが、この間、アプリケーションソフトウェアの開発は情報処理の専門家によって行われ、利用者はそれをエンドユーザとして使用することに専念してきた。

このような時代において、ソフトウェア開発環境は、８０年代半ばまではプログラマなどの情報処理技術者の作業効率の向上が主目的であった。その後のCASE（Computer-Aided Software Engineering）ツールにおいても、大規模高信頼ソフトウェアの開発に携わる種々の立場の情報処理の専門家をソフトウェア工学的な観点で支援するのが主目的であった。

しかし、高度情報化社会を迎えた現在では、情報システムは、すでに業務の効率化といった従来の目的を越えて、広く社会生活の中に入り込んできている。そして、そのような情報システムの利用に関して、従来の情報システム部門に対比したエンドユーザ部門という範囲を越えて、あらゆる人達がエンドユーザになりつつある。このような状況は以下のような理由によるところが大きく、今後は、エンドユーザコンピューティングを支援する環境が重要になろう。

(1)情報のパーソナル化

ハードウェアの低廉化と共に、パソコン、ワークステーションなどの普及がめざましく、マルチメディアデータのコンピュータによる処理が容易化するにつれて、OA分野を中心にエンドユーザが急増している。「物のパーソナル化」が「情報のパーソナル化」を促進している。

(2)情報のグローバル化

コンピュータの利用形態が大型機によるホスト集中型からパソコン、ワークステーション主体のネットワーク型へと移行すると共に、分散コンピーティングやインターネットの普及がめざましく、エンドユーザがコンピュータを直接操作して、離れた場所に点在するデータにアクセスする機会が急増している。「システムのグローバル化」が「情報のグローバル化」を促進している。

(3) 情報の資源化

コンピュータの利用目的が、従来の業務の合理化から業務革新や経営戦略の実現へと広がるにつれて、さまざまな業務担当者がエンドユーザとしてコンピュータを利用しはじめており、「情報の資源化」を促進している。

オフィスにおけるエンドユーザの業務を大ざっぱに分類すると、従来は定形業務と非定形業務に分けられた。定形業務はビジネス分野の帳票処理に代表されるような企業の基幹業務にかかわるものである。そのためのアプリケーションプログラムは、通常は、エンドユーザの要求をききながら、企業内の情報システム部門が開発あるいは導入して、エンドユーザに提供してきた。保守、拡張も情報システム部門が行い、エンドユーザは定められたユーザインタフェースに従ってコンピュータを使用するだけであった。一方、非定形業務については、表計算に代表されるような市販のOAソフトウェアを購入して利用してきた。

しかしながら、コンピュータの利用目的が、従来の業務の合理化から業務革新や経営戦略の具現化へと広がるにつれて、基幹業務と個人の非定形業務の連携が必要になってきている。この傾向は、コンピュータシステムが分散コンピューティングの方向に進展するにつれていっそう加速されている。その結果、例えば従来の表計算ソフトウェアと基幹業務データベースをつないだり、データベース検索ソフトウェアにデータの加工、編集機能を追加したりして、非定形業務の支援システムを作り上げることが重要になってきている。さらに、CSCW（Computer-Supported Cooperative Work）のように、別々の場所にいるグループが協力しあって業務を遂行していくためのグループウェアが求められている。

このような状況下でタイムリーに必要なソフトウェアを手にいれていくためには、もはや情報システム部門が提供するシステムや市販のソフトウェアパッケージを単に利用するだけでなく、業務担当者は、積極的に自ら使うソフトウェアの開発に関与していく必要がある。このようなエンドユーザコンピューティングのための開発環境としては、情報処理の専門家を対象にしたような従来の手続き型パラダイムに基づくプログラミングの支援ではなく、業務の専門家が自らの業務の知識を基本にしたプログラミングを可能とする新しいパラダイムに基づいた支援機能が必要がある。

1．1．2　多視点からの考察

ここで、エンドユーザコンピューティングの位置付けを明確にするために、次のような５つの視点からエンドユーザコンピューティングについて考察する。

・エンドユーザコンピューティングの定義

・ソフトウェア産業論

・ソフトウェア生産技術

・プログラミング言語

・開発環境

(1) エンドユーザコンピューティングの定義

はじめに、ここで議論するエンドユーザコンピューティングは、「誰が、何のために、どのように利用する技術」であるかを明確にするために、エンドユーザ、対象ソフトウェアおよび開発・保守形態に関して一応の定義をしておく。

(a) エンドユーザ

一般に、エンドユーザとして少なくとも以下の3種類が考えられる。

(i) 基幹業務担当者

例えば、銀行のようなユーザ企業において、システム部門に対するエンドユーザ部門に所属する人達で、利用するソフトウェアはシステム部門が開発し、提供してくれる。

(ii) 業務の専門家

一般にオフィスワーカーといわれるような人達で、DB検索や表計算などに市販のアプリケーションパッケージを利用する。

(iii) 一般ユーザ

例えば、日常生活の中で銀行のATMを利用するような一般の人達で、将来、マルチメディア時代の主要ユーザとなる。

ここでは、(i)と(ii)のエンドユーザに対象を絞るが、特に非定型業務のコンピュータ化という視点では今後急速に増大すると思われる(ii)の方により重点を置く。

(b) 対象ソフトウェア

ここでは、上記の(ii)のエンドユーザが利用するような、オフィス業務用アプリケーションが中心となる。従って、分散協調型ソフトウェアが主な対象になり、ワークフローシステム、グループウェア、エージェントシステムなども含まれる。規模的には、中、小規模のアプリケーションソフトウェアを想定することになるが、ネットワーク接続するによりシステムとしては大規模化することもある。

(c) 開発・保守形態

エンドユーザ主体の開発を念頭に置くならば、問題領域の分析によるドメインモデルの構築が重要である。即ち、問題領域のモデルを作成し、そのモデル上でのシミュレーションによりモデルの妥当性を検証した後、実用に際しては、そのモデルをインタプリタにより実行するか、あるいは必要に応じてプログラムを自動生成するという方法である。比較的小規模のものについては、ドメインモデルの構築とプログラミングが一体化して、核の部分からだんだん機能を拡張していくようなプロトタイピング方式で開発する

ことになろう。

このように最終的にはエンドユーザが自らの業務のアプリケーションソフトウェアを自ら開発し、自ら利用することを理想とするが、その実現に向けての技術課題は多い。そこで、マイルストーンとして、エンドユーザが主体でシステムエンジニアの助けを借りて開発するが、保守はエンドユーザだけで行うレベルが当面の目標となるであろう。

(2) ソフトウェア産業論

次に、エンドユーザコンピューティングをソフトウェア産業の発展過程という視点でとらえるならば、表1.1-1にも示すように、労働集約型産業、知識集約型産業に続くものとして位置付けることができる。

(a) 労働集約型産業

これまでのソフトウェア産業は労働集約型産業であり、生産コストあたりの生産量という生産性の効率向上が重要であった。ステップ数／人月あるいはステップ単価を改善するために、CASEツール等を用いて自動化率を向上させ、人海戦術からの脱皮の努力をしてきた。しかし、この生産性の尺度の致命的欠陥は、ソフトウェアの価値（質）を規模（量）ではかることである。そのため、ソフトウェアの受託開発において、生産コストを発注者がすべて支払う場合は、ステップ数が多いほど価格（価値）が上がることになってしまい、受注者側の技術力向上の努力がおろそかになってしまう。このような矛盾は、バブル経済崩壊とともにソフトウェア産業が不況業種の代表になってしまったことによくあらわれている。

(b) 知識集約型産業

ソフトウェアの生産性は本来以下のように定義されるべきである。

ソフトウェアの生産性＝生産物の価値／生産コスト

この「生産物の価値」はユーザの視点で決まるべきものであり、高品質のソフトウェアやベストセラーのアプリケーション・パッケージの価値は高い。このようなソフトウェアの開発には、業務の知識と情報処理技術の双方が必要である。

表1.1-1　ソフトウェア産業論

ソフトウェア産業形態	主要な技術職	主要技術
労働集約型産業	プログラマ	自動化（CASE）
知識集約型産業	設計者	標準化（パッケージ）
ポスト知識集約型産業 （知恵集約型産業）	業務専門家	エンドユーザ コンピューティング

今後、ソフトウェアのビジネスは、アプリケーション・パッケージやその基となるコンポーネントウェアの分野とシステム・インテグレーションの分野に2分化して発展していくと思われるが、この場合、種々の分野対応の業務の知識と情報処理の知識をノウハウとして蓄積することが必要である。

(c) ポスト知識集約型産業

情報処理システムが、業務の効率化ではなく、業務革新や経営戦略の具現化に用いられるようになると、ソフトウェアの開発においても効率よりも効果が重要視される。何を作るかが最も大事であり、それを決めるのは業務専門家である。業務専門家が知恵をしぼって効果的なアプリケーションをタイムリーに作っていくためには、エンドユーザ自身が開発でき、かつ保守拡張ができる必要がある。対象は異なるが、例えば、最近注目されているRAD（Rapid Application Development）においてもエンドユーザが全工程に参加することが重要な成功要因になっていることと同じである。

このようなエンドユーザコンピューティングを実現するためには、そのためのツールや環境を提供しなければならない。自動化ツールや標準パッケージなどの情報処理技術を統合した上に、きめ細かく分類された応用分野対応（domain-specific, application-oriented）の業務の知識に基づいたアプリケーション・フレームワークの構築が必須である。潜在ユーザとしての業務の専門家の数を考えると、市場規模は現在の情報サービス産業のそれ（数兆円規模）を大きく上回ることになる。ポスト知識集約型産業は、知恵集約型産業ともいえる。

(3) ソフトウェア生産技術

第3の視点として、エンドユーザコンピューティングをソフトウェア生産技術の観点から考察する。情報化社会に対応して、一般にソフトウェアの規模と量と質に関するソフトウェア生産技術への要求は急速に増大しているが、ソフトウェアの問題は多種多様であり、何を開発するか、誰が使うか、誰が開発するか、等々の条件によって問題が異なることが多い。

一般的にメーカの視点（作る立場）から、ソフトウェア生産性に関する解決手段をあげると、以下の5項目が考えられる。

・開発対象（ソフトウェア）の標準化
・開発工程（プロセス）の定式化、自動化
・開発手段（ツール）の高機能化
・開発者（プログラマ）の技術力向上
・業務専門家（ユーザ）による開発可能化

これらは、新規開発量の抑制、工数削減、作業効率の向上、労働の質の向上などの形で直接、間接に「規模」、「量」、「質」の問題の解決に寄与する。従来は最初の4項目に努力が払われてきたが、これから

は５番目のエンドユーザコンピューティングも視野に入れる必要がある。

次に、これらの解決手段を、ユーザのソフトウェア入手方法というユーザの視点（使う立場）で見直すと、表1.1-2に示すような「標準化」、「自動化」、「情報処理技術者の自由業化」、「エンドユーザコンピューティング」の４種類のシナリオ案を描くことができる。「標準化」と「自動化」に関しては従来から相当の努力が行われてきた。「情報処理技術者の自由業化」は、日本の社会構造の変化を伴うため少し先の話になる。当面は「エンドユーザコンピューティング」が大きな鍵になると思われる。以下にこれらの解決方法のシナリオについて簡単に述べておく。

(a) 標準化シナリオ

標準化の基本は共通化による共有化であり、古くて新しい問題である。これまでソフトウェアの標準化には次のような幾つかの段階があった。

・同一組織内での部品化、再利用

・同一組織内、複数機種間での共通化

・複数組織間、複数機種間での共通化

・コンポーネントウェアおよびアプリケーションパッケージの業界標準化

第１段階は、従来からプログラムの部品化、再利用技術として研究がなされてきたものである。第２段階は、アプリケーション・ソフトウェアの異機種間の移行性を高めるために、アプリケーション・ソフトウェアの標準的なアーキテクチャを設定し、プログラミング言語、ユーザインタフェース、データベース管理、通信管理などの外部仕様を統一するもので、８０年代後半にメーカ側から提案されたものである。現在は、第３段階であり、次のような特徴に代表されるオープンシステムの時代である。

・異機種／異種プラットフォーム間でのアプリケーションの移行性

・異機種／異種プラットフォーム間の接続性／相互運用性

表1.1-2　ユーザ視点でのソフトウェア入手の問題の解決

解決手段	ユーザのソフトウェア入手方法
標準化	適正価格の市販パッケージ購入
自動化	要求使用を提示して自動生成
情報処理技術者の自由業化	優秀な技術者に高額で依頼
エンドユーザコンピューティング	自分で作成

９０年代のシステム構成は、大まかには、図1.1-1に示すように、応用ソフトウェア、ミドルウェア、基本ソフトウェア、ハードウェアの４階層で表現される。ミドルウェアや基本ソフトウェアとアプリケーションとの間のAPI（Application Programming Interface）をできるだけ業界標準や国際標準にして、移行性や相互接続性を可能とする。特にミドルウェアの役割は大きい。

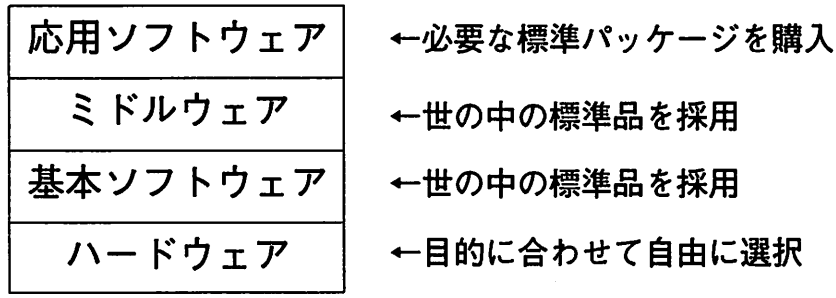

図1.1-1　オープンシステムのシステム構成

最後の段階は、特定用途のアプリケーション・パッケージの中からベストセラーになったものが業界標準になっていく。OAソフト、RDBソフトなどでその傾向が強い。銀行システムではメガステップオーダのパッケージもある。ただし、変化の激しい分野で、１つのアプリケーションパッケージがいつまでも業界標準ではありえず、また変化に対応した機能拡張によるマンモス化は、エンドユーザにとって好ましいことではない。現実には、むしろ、コンポーネントウェアの流通が促進され、コンポーネントの差し替えによってエンドユーザに必要な機能のみを取り込んでいくスタイルになるものと思われる。

このように標準化が進めば、ユーザは、自分の業務に必要な市販のアプリケーションパッケージやコンポーネントウェアを適正な価格で買ってくることができる。

(b) 自動化シナリオ

「自動化」、即ち「自動プログラミング」は、プログラム内蔵方式のコンピュータの発明以来のソフトウェア技術者の夢である。かっては、アセンブラやコンパイラが自動プログラミング技術と呼ばれた時代もあった。プログラムの記述形式は、機械語からアセンブラ、高級言語へと発展し、機械語への展開率の向上という意味での「自動化」を実現したが、プログラムを手続き的に記述するというスタイルは本質的に変わっていない。現在は、仕様記述言語と仕様からのプログラム自動生成技術の研究が活発に行なわれている。

最終的には、ユーザは、自分の業務の用語で要求定義を記述するだけで、プログラムを自動生成することができる。

(c) 情報処理技術者の自由業化シナリオ

現在、情報処理技術者の社会的待遇が他に比べて明らかに優位にあるということはないが、今後、急激な変化がありえる。勿論、この業務は個々の技術者の能力差が非常に大きいので、建築士などの専門家と同じように個人の能力に応じた収入を得られるように自由業化するべき分野である。銀行オンラインシステムや電話交換システムなど、社会的責任の大きいシステムの数が増加するので、優秀な情報処理技術者の高収入は保証されるであろう。

その結果として、ユーザは、お金さえ払えば、優秀な技術者が作る良質のソフトウェアをタイムリーに入手できる。

(d) エンドユーザコンピューティングシナリオ

ユーザ業務を大まかに定形業務と非定形業務に分けて考えると、定形業務については、「標準化」シナリオあるいは「自動プログラミング」シナリオによって解決する。一方、非定形業務については、自分の業務に固有の部分のソフトウェアを作成する必要がある。今後急激な増大が予想されるこのようなソフトウェアはユーザ自身が自分で開発できることが理想である。

エンドユーザコンピューティングが発展すれば、ユーザは、"プログラミング"を意識することなく、自分の業務をメタファベースで、あるいは業務の言葉を用いて簡単にコンピュータ化できる。

(4) プログラミング言語

第4の視点として、エンドユーザコンピューティングをプログラミング言語の観点から考察する。プログラミング言語は、当初からソフトウェア生産性向上に大きな役割を果たしてきた技術である。プログラミング言語の研究開発に関するこれまでのマクロなトレンドを表1.1-3のように要約することができる。

表1.1-3　プログラミングパラダイムの変遷

時期	目的	内容
１９６０年代	量的高級化	記述水準の向上
１９７０年代	質的高級化	プログラミング方法論
１９８０年代	脱手続型パラダイム	宣言的記述
１９９０年代	エンドユーザ コンピューティング	脱プログラミング

(a) 高級言語化

１９６０年代は、高価なコンピュータを効率良く利用することが重要であった。このようなコンピュータ利用の初期の段階に、プログラムの実行効率をあまり低下させないで生産効率をあげるために、アセンブリ言語からFORTRAN、COBOL、PL/Iなどの高級言語への移行が行われた。これは、機械語への展開率の向上という意味で「量的高級化」といえる。

(b) 構造化

次に１９７０年代は、ソフトウェアの大規模化という第１次のソフトウェア危機への対応が重要であった。そのため、高信頼性と保守性を重視したプログラミングスタイルの研究が行なわれ、構造化技法などによりプログラムの理解容易性を実現する「質的高級化」が追求された。

(c) 脱手続型パラダイム

そこで１９８０年代には、ソフトウェアの生産性、信頼性、保守性に関する諸問題は、本質的に手続き型のプログラミングパラダイムに起因するという考えから、論理型、関数型などの非手続き型のプログラミングパラダイムとそれをベースにした言語の研究が盛んに行なわれるようになった。

(d) エンドユーザコンピューティング

最近では、情報処理の専門家でなくてもソフトウェアを開発できるように、非手続き型プログラミングの概念をより進めて「脱プログラミング」を実現するエンドユーザコンピューティングの研究が盛んである。エンドユーザがプログラミング言語を用いないで業務のコンピュータ化を行なえるための現実的な技術としては、業務向け簡易言語、日本語プログラミング、ビジュアルプログラミング、プロダクションシステム等がある。いずれも脱プログラミングのコンセプトに基づいているが、現状ではまだ「脱プログラミング言語」のレベルであり、プログラミングの概念やセンスが不要というわけではない。

(５) 開発環境

最後に、エンドユーザコンピューティングを開発環境の視点から考察する。開発環境とは、狭い意味では、ソフトウェアの開発のためにコンピュータ上で利用するツールの集合であるが、実際にはそれらのツールが提供する開発技法に加えて、それらのツールの操作対象となるデータの管理、それらのツールを有効活用するための開発方法論やプロセス管理、さらには開発担当者まで含めたプロジェクト管理も関連している。むしろ開発方法論の実現手段として、あるいはプロセス管理やプロジェクト管理の一環として開発環境ないしツールがあると考えるべきである。したがって、開発環境の目的は、一般には「良いものを早く安く作ること」すなわち高品質ソフトウェアの高効率生産を促進することである。

開発環境をツール史の視点からみると次のようにまとめられる。その概要を表1.1-4に示す。

表1.1-4　ソフトウェア開発環境の変遷

年　代	特　　徴
５０年代 ｜ ６０年代	●バッチ型プログラミングツール ・高級言語コンパイラ、ソースファイル操作ユーティリティ
７０年代 前半	●対話型プログラミングツール *～バッチ型処理から対話型（ＴＳＳ）処理へパラダイムシフト～* ・上流工程：ドキュメント作成支援 ・下流工程：ツールボックス（エディタ，デバッガなど）
７０年代 後半 ｜ ８０年代 前半	●統合プログラミング環境 *～開発手段（ツール）の高機能化から* *開発工程（プロセス）の定式化へパラダイムシフト～* ・ユーザインタフェース、プログラミング用データベース共通化 ・特定プログラミング言語向き環境（構造エディタなど）
８０年代 後半	●統合開発環境 *～製造工程(how-to-make)中心から* *計画・分析・設計工程(what-to-make)中心へパラダイムシフト～* ・ＣＡＳＥツール ・ユーザインタフェース統合、リポジトリによるデータ統合
９０年代 前半	●オープンシステム *～開発工程（プロセス）の定式化から* *開発対象（プロダクト）の標準化へパラダイムシフト～* ・開発環境のプラットフォーム、ミドルウェアの共通化 ・分散環境化
９０年代 後半	●エンドユーザコンピューティング *～生産者中心の視点から利用者中心の視点へパラダイムシフト～* ・コンポーネントウェア、ビジュアルプログラミング

(a) ５０－６０年代

ソフトウェアツールの歴史は、１９４９年のプログラム内蔵方式のコンピュータの実用化に始まる。プログラムの記述形式は、機械語からアセンブラ、さらに高級言語へと発展した。この時期には、コンパイラやプログラムのソースファイル操作用のユーティリティなどのツールがバッチ処理形態で使われていた。

(b) ７０年代前半

ソフトウェア工学の研究初期の１９７０年代は、上流工程に注目した要求定義技法や設計技法の研究が盛んに行なわれたが、コンピュータによる支援という観点では、ドキュメント作成支援程度の自動化しか

実現できなかった。そのため、下流工程で利用するプログラミングツールの個別開発が中心に行なわれた。UNIXに見られるような、いわゆるツールボックス型といわれるものである。この時期には、ツールの利用形態が、バッチ処理形態から対話処理（TSS処理）形態へと移っていった。

(c) ７０年代後半－８０年代前半

その後、プログラミングツールの統合化が進んだ。この時期に従来の開発手段（ツール）の高機能化から開発工程（プロセス）の定式化へのパラダイムシフトが生じ、環境という表現が一般化した。当時の統合プログラミング環境は、ユーザインタフェースとプログラミング用データベースの共通化によるツール統合が基本であった。ツール間の連携は、特定のプログラミング言語を対象にした言語指向プログラミング環境が先行する形で発展した。上流工程に関しては、構造化技法が定着し始めた時期であり、データフロー図やプログラム構造図などの作図やドキュメンテーションが支援されるようになった。同時に構造化図式を対象としたエディタも利用され、ビジュアル化が重視された。

(d) ８０年代後半

ソフトウェア開発の真の難しさは上流工程にあるという認識から、再び上流工程の方法論や技法のツール化が積極的に試みられ始めた。how-to-makeが主体の製造工程からwhat-to-makeを重視する計画、分析、設計工程へと関心が移っていった。このような上流工程支援を含む統合開発環境を統合CASEと呼ぶ。ユーザインタフェースに関しては、ワークステーションの進歩により使い勝手の良いGUI（Graphical User Interface）が一般化した。データの管理に関しては、リポジトリによるライフサイクル支援がある。

(e) ９０年代前半

最近では、情報処理技術者の不足（正確に言えば、技術不足）とアプリケーション・ソフトウェアの複雑化の挟み撃ちにあって、CASEへの期待が高まっている。オープンシステムの時代とともに、開発工程（プロセス）の定式化・自動化から開発対象（ソフトウェア）の標準化へのパラダイムシフトが生じている。プロセス・イノベーションよりもプロダクト・イノベーションが求められている。

(f) ９０年代後半

このようなトレンドの中で、ソフトウェアに関してもアプリケーションがより重要となり、生産者中心の視点から利用者中心の視点へパラダイムシフトが起きている。エンドユーザコンピューティングが注目され、新しいアプリケーションの作り方が模索されている。コンポーネントウェアやビジュアルプログラミングの技術が発展し、分野別にアプリケーション・ソフトウェアの標準アーキテクチャと部品ライブラリを用意し、業務モデルから部品合成によりアプリケーションを生成するようになる。

1．1．3　主要技術

エンドユーザコンピューティングのための技術として、5年前に業務向け簡易言語、日本語プログラミング、ビジュアルプログラミング、プロダクションシステムを候補[7]に上げたが、最近、発展がめざましいのはビジュアルプログラミングである。

(1) ビジュアルプログラミング

ビジュアルプログラミングの発展の経緯を以下のような流れでとらえることができる。

・オープンシステム時代のソフトウェア・アーキテクチャの要請

→・オブジェクト指向技術による階層化と部品化

→・コンポーネントウェアの出現

→・ビジュアルプログラミングの実用化

(a) オープンシステム時代のソフトウェア・アーキテクチャ

最近の情報システム構築に関する大きな変化であるオープンシステム化においては、異機種コンピュータ間でのアプリケーションソフトウェアの接続性や移行性の要求に応えるために、アプリケーションソフトウェアのアーキテクチャをできるだけ標準的に作る努力がなされており、図1.1-1にも示したように、以下のようなソフトウェア構成をとる。

・階層化：ハード、基本ソフト、ミドルウェア、応用ソフトウェアの階層構造化。

・部品化：各階層を部品の集合で構成。

・統一インタフェース化：階層間のインタフェースの統一。

(b) オブジェクト指向技術による階層化と部品化

このようなアーキテクチャを実現するための技術としてオブジェクト指向技術が注目され、既にプログラミングの分野では実用的効果をあげている。社会的要請から最近発展の著しいオープンシステムと四半世紀の歴史を有するオブジェクト指向技術との歴史的出会いとも言えるものである。

オブジェクト指向概念の基本的な要件として、以下の4項目をとりあげる。

・分散協調型計算モデル

・データ抽象化機能

・クラスからのインスタンス生成機能

・クラスの階層化と継承機能

これらは各々独立した概念であるが、ソフトウェア開発のどの局面に注目するかによって、適用すべき概念とその効果が決まる。上記の階層化、部品化、統一インタフェース化と関連では以下の効果が期待される。

(i) 開発者のための部品化、再利用

・データ抽象化により、独立性の強い、汎用的な機能部品を作りやすい。

・階層化と継承機能により、部品ライブラリを構築しやすい。

・インスタンス生成機能により、部品のカストマイズがしやすい。

・分散協調型モデルに基づくプログラム設計により、部品の抽出、利用が容易である。

(ii) 保守者のための拡張性と移行性

・階層化と継承機能により、機能追加が容易である。

・データ抽象化により、機能変更や他機種への移行が容易である。

(iii) 利用者のための操作性と統一性

・データ抽象化により、画面上のオブジェクトの機能の１つを選ぶという直接操作で簡単に対話できる。

・種々の応用ソフトウェアのユーザインタフェースは共通部品での構築により、統一できる。

(c) コンポーネントウェア

このように従来からオブジェクト指向プログラミングにおいて、オブジェクトの部品化・再利用の利点が期待されていたが、抽象データ型のレベルのクラスライブラリを提供しただけでは、なかなか部品の有効活用が難しい。部品としての粒度が小さすぎて、プログラミング技術レベルの知識が要求されるのが現状である。

そこで、アプリケーションと部品の粒度とのこのようなギャップを埋めるために、最近では以下のような３種類の粒度のコンポーネントが考えられている。

・アプリケーションフレームワーク

・デザインパターン

・クラスまたはオブジェクト

フレームワークは、アプリケーションのソフトウェアアーキテクチャ、クラスは、アプリケーション用部品、デザインパターンは、フレームワークに部品を埋め込むときの工法／定石という位置づけで、上記の記述順に粒度が大、中、小である。各々のレベルでドメイン固有のものと汎用性の高いものがある。

最近、特に複数のクラスを組み合わせてそれらの典型的な協調作業をパターン化した手法が注目されている。Eric Gammaらは、オブジェクト指向プログラミングの経験から２３の典型的なデザインパターンを抽出し、「オブジェクト指向ソフトウェア設計における、特定の問題に対する”simple and elegant”な解」として利用することを薦めている。これらのデザインパターンでは、「特定のコンテキストの中で一般性のある設計問題を解くためにカストマイズされた、オブジェクト／クラスの協調」をパターン化している。

コンポーネントウェア普及のためのインフラも実用レベルのものが出始めている。コンポーネントウェ

ア関連のミドルウェアとして、分散環境下でのオブジェクト管理を共通化して相互運用性を確保したり、アプリケーション間連携を容易にするためのプラットフォームが普及しはじめている。その例として、CORBA（OMG）、OLE（Microsoft）、OpenDoc（CI Labs）などがあげられる。

(d) ビジュアルプログラミングの変遷

このようなコンポーネントウェアが充実しても、プログラミングの本質的な難しさを克服できなければエンドユーザコンピューティングツールにはならない。これまで以下の３つの観点からのプログラムの視覚化の研究が活発に行なわれてきた。

・視覚的ユーザインタフェース

・視覚的編集

・視覚的言語

第１の視覚的ユーザインタフェースは、プログラミング環境を用いてプログラムを開発するときに、システム側から提供される種々の情報をグラフィカルに表示し、ユーザ側からの入力もそのグラフィカルな画面上で行なえるようにするものである。

第２の視覚的編集は、従来のテキストエディタに代わり、プログラミング言語の文法規則の基本構造をテンプレートとして表示し、文法的に正しいプログラムを誘導するものである。構造エディタや構造化図式を用いた図式エディタがすでに使われている。

第３の視覚的言語は、本来的に視覚的表現を基本とした言語であり、その視覚化に用いる図式の表現形式の違いにより、フォーム指向言語、グラフ指向言語、アイコン指向言語に分類される。

フォーム指向言語は、表形式を基本とするもので、OA分野のスプレッドシートを用いたフォームベースプログラミングが代表的である。グラフ指向言語は、ノードとアークで構成される有向グラフ表現を基本とするもので、データフロー図、状態遷移図などがある。アイコン指向言語は、ビジュアルオブジェクト（アイコン）とその関係表現を基本とするものである。基本的な処理単位をアイコンとして用意しておき、画面上でアイコンとアイコンを結ぶことによってプログラムを作成していく。

以上のビジュアルプログラミング技術のうち、視覚的ユーザインタフェースと視覚的編集はプログラミング言語の概念から開放されていないので、エンドユーザコンピューティングという観点では、不十分であろう。

最近、この視覚的言語レベルの製品としてコンポーネントウェアを用いたビジュアルプログラミングツールがいろいろと出始めている。例えば、IntelligentPad（富士通、日立ソフト）、HOLON/VP（NEC）、APPGALLERY（日立）、VisualAge（IBM）などがある。いずれもオブジェクト指向技術をベースにビジュアルに部品を組み合わせてアプリケーションを構築していく方法や、部品あるいはそれをアプリケーショ

ンに組み込んだインスタンスがオブジェクト指向プログラミング言語で表現されている点などが共通している。部品の組み合わせ方法も部品の重ね合せによる方法か部品間を線で結ぶ方法が一般的である。

(2) その他

エンドユーザが業務のコンピュータ化を行なえるための現実的な技術の例として、ビジュアルプログラミング以外のものについても簡単に説明しておく。

(a) 第4世代言語

エンドユーザコンピューティングを指向した業務向け簡易言語として第4世代言語（4GL）がある。第4世代言語という名前は、第1世代言語（機械語）、第2世代言語（アセンブラ）、第3世代言語（コンパイラ言語）に続く言語という意味で付けられた。James Martinは、4GLを13ヶ条の特徴で定義したが、その主なものは、非職業的プログラマが使用可能であること、アプリケーションプログラムの記述ステップ数と作成工数がCOBOLより1桁少ないこと、非手続き的記述形式の採用、などである。

4GLユーザの98%がエンドユーザだけでのソフトウェア開発可能性を否定しているという調査結果もあるが、その理由は、現在の4GLが、COBOLと比較した展開率の向上という「量的高級化」アプローチに基づいて設計されており、プログラミング技術を必要としているためと思われる。

今後、4GLのような業務向け簡易言語をエンドユーザコンピューティングのツールとしていくためには、業務の言葉で要求仕様を記述でき、業務知識のデータベースを用いてプログラムを自動生成できるようにする必要がある。

(b) 日本語プログラミング

職業的プログラマでないエンドユーザにとって、プログラムの日本語表現は書き易さ、読み易さの点で魅力的である。特にアプリケーションプログラムの保守性に不可欠であるプログラムの理解容易性の向上には、プログラム構造がきれいであることと共に、プログラムの各文の意味がわかりやすいことが重要である。

プログラムの日本語表現には次のようなレベルがある。

(i) 既存のプログラミング言語でデータ名や手続き名に日本語を使う。

(ii) 既存のプログラミング言語のif, endなどのキーワードも含め、すべて日本語で記述する。

(iii) 既存のプログラミング言語相当の記述レベルで独自の文法規則を持つ日本語プログラミング言語で記述する。

(iv) 仕様記述言語として日本語を導入し、プログラムを自動生成する。

現在の実用レベルは(ii)、(iii)が主で、4GLに組み込まれているものが多い。今後は(iv)が主流になっていくと思われるが、次のような技術課題がある。

・業務用語の使用に対応した業務用語辞書の作成保守機能

・日本語記述レベルですべて処理するための開発保守環境

・日本語表現のあいまいさを回避する文法規則

日本語化が進んでいる分野としてデータベース検索のコマンド・インタフェースがある。例えば、データベース検索用自然語インタフェースを用いて入力された日本語文をSQL文に変換して実行してくれる。

日本語プログラミングのニーズが大きいもう１つの分野は、金融オンラインシステムのようなメガステップオーダの大規模高信頼ソフトウェアの開発である。これまでは、ユーザの情報システム部門、ソフトウェアハウス、メーカが協力して、COBOLやPL/Iなどのプログラミング言語を用いて開発してきたが、これまで以上の大規模システムの開発は以下のような理由で無理がある。

・計画段階では数年先の稼働時の機能設計は不可能。

・開発管理が規模的に不可能。

・開発中や稼働後の機能変更への迅速な対応が不可能。

これに対し、日本語プログラミング技術を適用すれば、プログラムの仕様が業務仕様書レベルになり、エンドユーザがプログラムの開発や保守に関与できるので、上記の問題は軽くなる。

(c) 人工知能言語

知識ベースシステムやエキスパートシステムが種々の分野で実用になっているが、その理由は、"推論機能"というよりはむしろ"簡易プログラミング"としての魅力にある。職業的プログラマでない業務専門家が業務の言葉で表現できる利点が大きい。さらに、もっともよく用いられている知識表現であるプロダクションルールの場合は記述形式が「もし～ならば、～せよ」という１つのパターンに限定されており、かつ基本的には記述順序が自由であるので、業務知識の記述、追加、修正が容易である。

従来の手続き型プログラムが、N. Wirth の図式で、

プログラム＝アルゴリズム＋データ構造

と表現されたのに対応させると、知識ベースシステムでは、

知識ベースシステム＝推論エンジン＋知識ベース

となるが、アルゴリズムに対応する推論エンジンは既に用意されている。したがって、ユーザは、データに対応する知識だけを記述すればよい。

分散コンピューティングの世界では、このようなルールベースシステムがグループウェアやエージェントとしてお互いにメッセージをやり取りして分散協調システムを形成することになる。

1．1．4　おわりに

新しいアプリケーションの作り方として、最近のコンポーネントウェアとビジュアルプログラミングの動きは注目に値する。コンポーネントの流通市場が確立するような本格的な実用化に向けての課題は多いと思われるが、夢も多い。高度情報化社会を真に豊かな社会とするためには、エンドユーザコンピューティングの成否が鍵である。その成功の秘訣は、従来の生産者中心の視点から利用者中心の視点への転換ではないだろうか。

従来のソフトウェア工学の分野では、論文になった新しい技術が実用になる道のりは意外とけわしく、早くても１０年はかかるが、それは一握りの成功例にすぎない。技術移転の難しさの原因の１つは研究開発におけるエンドユーザの視点の欠如と思われる。幸いにもエンドユーザコンピューティングの分野においては、さすがにエンドユーザの視点の欠如はないであろうが、他の分野以上にエンドユーザの視点に立って、身近なところから始めて本質に迫るというアプローチが必要である。

【参考文献】

1) 青山幹雄：コンポーネントウェア：部品組み立て型ソフトウェア開発技術, 情報処理, 37, 1, 71-79 (1996).

2) 鯵坂恒夫 編集：特集「ソフトウェア生産環境」, コンピュータソフトウェア, 10, 2, (1993).

3) 中所：開発環境, 情報処理ハンドブック, 情報処理学会編, オ-ム社, 747-758 (1995).

4) 中所：M-base：「ドメインモデル≡計算モデル」を志向したアプリケーションソフトウェア開発環境の基本概念, 情報処理学会ソフトウェア工学研究会資料, 95-SE-104, 104-4, 25-32 (1995).

5) 中所：エンドユーザコンピューティング　ーソフトウェア危機回避のシナリオー, 情報処理, 32, 8, 950-960 (1991).

6) 日本情報処理開発協会(編)：情報化白書1995, コンピュータ・エージ社(1995).

7) Ambler,A.L. and Burnett,M.M. : Influence of visual technology on the evolution of language environments, IEEE Computer, 22, 10, 9-22 (1989).

8) Gamma, E., et al. : Design pattern, Addison Wesley (1995).

9) Grudin, J., Computer-supported cooperative work : history and focus, IEEE Computer, 27, 5, 19-26 (1994).

10) Helm, R. : Patterns in practice, Proc. OOPSLA'95, 337-341 (1995).

11) Jones, C. : End-user programming, IEEE Computer, 28, 9, pp.68-70 (1995).

12) Maes, P. : Agents that reduce work and information overload, Comm. ACM, 37, 7, 30-40 (1994).

13) Malone, T., Lai, K. and Fry, C. : Experiments with Oval : a radically tailorable tool for cooperative work, Proc.

CSCW92, 289-297 (1992).

14) Martin,J. : Fourth generation languages, Prentice-Hall (1985).

15) McLean, E., et al. : Converging end-user and corporate computing, Comm. ACM, 36, 12, 79-92 (1993).

16) Medina-Mora, R., et al. : The action workflow approach to workflow management technology, Proc. CSCW92, 1-10 (1992).

17) Riecken, D. (Ed.) : Special issue " Intelligent agents," Comm. ACM, 37, 7 (1994).

18) Smith, R. B. (Moderator) : Prototype-based languages: object lessons from class-free programming (Panel) : Proc. OOPSLA'94, 102-112 (1994).