オープン環境における入出力の部品化と標準化

User interface Components as Standardized Parts of Solutions in Open Development Environment

お お 津 昌 三

要　約　Windows NT 環境での金融ソリューション開発，特に利用者インタフェース開発の標準化を目指して 1997 年度に SWEETS プロジェクト（旧称金融入出力標準化プロジェクト）が試みられた．SWEETS プロジェクトでは，金融ソリューション開発における生産性の向上を目的とした利用者インタフェース部品の開発，使用性の標準化を目的とした規約の整備，およびこれらを格納するリポジトリとその運用ツールの開発が行われた．現在その成果物は実際のシステム開発への適用を評価中である．

本稿では，SWEETS プロジェクトの成果，その実適用にあたっての課題について，紹介する．

Abstract　The SWEETS project was formed in FY 1998, given a mission to develop financial solutions in Windows NT environment, in particular, to standardize the development of user interface. The outcome of SWEETS project includes user interface components, guidelines for user interface design, repository, and tools for management. These outcome are in the evaluation phase in terms of its applicability to real development projects. Presented herein are what SWEETS project achieved and issues to be tackled when this is applied to a real development environment.

1. はじめに

1980 年代以降，汎用機をベースに様々な金融ソリューション[*1]が開発されてきた．しかし，今日のオープン化の波，特に Microsoft Windows NT ベースのクライアントサーバ型ソリューション提供への要求は，新技術に対しては保守的であった金融機関においてさえ着実に高まっている．これは，単に技術的な評価としてオープン環境が使用に耐えられるようになったというだけではない．いわゆる金融ビッグバンを迎えるにあたり，商品開発に不可欠なシステム開発・保守のスピードという面で，プロプライエタリな環境に比して有利だと判断されることも背景に存在するはずである．

このような環境変化の中で，本来はソフトウェア開発において非常に重要な要素であるはずの使用性（Usability）が再認識されてきている．それまでのソリューションの利用者インタフェースは，一般的に汎用機に接続された各社独自の端末システムあるいはそのエミュレータ上に実装され，その独自端末ハードウェアの使用性に依存していた．また，利用者そのものは，ソリューションへのインタフェースセッション（ソリューションとの会話の開始から終了まで）の中で，処理の逐次性に依存して，ソリューションによって制御されることが当然のごとく扱われていた．

この状況は，Windows がクライアント（利用者が直接的に対話する端末）市場で事実上の標準オペレーティングシステムを勝ち取ったことで大きく変わった．利用者は，Windows 上で動作するソフトウェアの使用性について提供ベンダの如何にかか

わらず共通であることを当然として求め，一般的に「ソフトウェアによって制御される」のではなく「ソフトウェアを制御している」という感覚を重要視するようになったと考えられている[1]．

このことを開発者の立場から見た場合のインパクトは相当なものであり，従来一般的であった機能重視でウォータフォール型開発を前提とした要求定義では，以降の工程で利用者インタフェースを実装する上での指針としては不充分となったこと，利用者インタフェースを実装する上で考慮しなければならないイベントが従来に比べ増大し，それにともなってコード量が飛躍的に増大したこと等があげられる[*2]．これはソリューションを構築する上で，ソフトウェア品質や開発コストに少なからず影響を与えることとなり，利用者インタフェースの部品化/標準化に対する要求が強まった．

このような背景から，1997 年度に当時の金融システム開発本部において金融入出力標準化プロジェクトが立ち上げられた．このプロジェクトは後に SWEETS（コード名：Sophisticated Windows developer's Environment for Enterprise Tactical System）プロジェクトと呼ばれ，その成果物も総称して SWEETS と呼ばれている．本稿では，SWEETS の成果物の概要を紹介するとともに当社標準の情報システム設計方法論の TEAMdesign[*3] との関係や，Microsoft　Windows システム上で利用者インタフェースを構築する際の一般的な注意点について考察する．

2.　利用者インタフェース開発環境―SWEETS 概要

2.1　SWEETS の目標

1 章で述べた背景の中で，SWEETS プロジェクトを立ち上げるにあたり，利用者インタフェース開発の課題と認識されていたものは以下の通りであった．

1）　ソリューションを構築する上で，利用者インタフェースに関する設計ガイドラインは示されていないか,示されていたとしてもソリューション毎に固有であり，必ずしもソリューション横断の標準化は考慮されていない．

2）　利用者インタフェースを構築する上での設計手法および開発環境が統一されておらず，再利用性が考慮されていない．

3）　Windows 上のソフトウェアを対象とする開発環境としては，VB（Microsoft Visual Basic）や VC＋＋（Microsoft Visual C＋＋）があり，それぞれ汎用的なフレームワークを用意しているものの，開発者がソフトウェア要求を直接写像するには抽象度が高く多大な労力を要する．また，その写像経験を得るための努力が共有されていない．

これらの課題を解決するために，SWEETS プロジェクトでは「利用者インタフェース構築に関する各種規約・ガイドの整備，再利用可能な金融ソリューション用共通部品の整備，およびそれを管理する仕組みを構築することで，生産性の向上・品質の向上・保守性の向上を図り，ユーザの満足度向上に寄与する」ことを目標とした．

2.2　SWEETS の対象範囲

2.2.1　コンピューティングモデル

SWEETS が対象とするのは，クライアントサーバ 3 層モデルのプレゼンテーション層，すなわち，図 1 のプレゼンテーションロジック部およびその関連領域である．

なお，3層の物理的な配置については特に限定していないし，アプリケーションロジックとデータベースアクセスロジックの分離の必要性も要求はしていない．

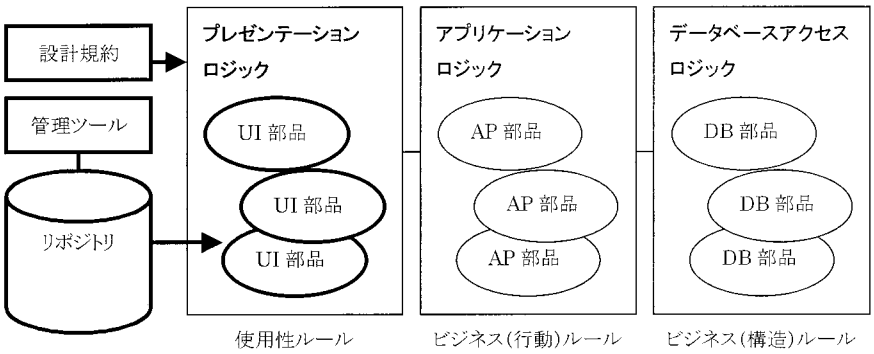

図 1　コンピューティングモデルと SWEETS の範囲

2.2.2 開 発 言 語

Windows ベースのプレゼンテーションロジックを開発するための言語は VB，VC＋＋をはじめとして数多く存在し検討対象となったが，SWEETS では VB を対象とした．これは，SWEETS プロジェクト開始時点で利用できる部品数が圧倒的に VB に有利であったこと，リポジトリモデルを設計するに際し言語非依存と管理ツールの使用性がトレードオフであったことによる．

2.2.3 実 行 環 境

SWEETS では，それを使って開発されたソフトウェアを実行するための環境として，Intel アーキテクチャ PC 上の Windows NT 4.0 を対象とした．ハードウェアについては PC 97（PC 97 Hardware Design Guide)[2]，PC 98（PC 98 System Design Guide)[3]，Net PC(Network PC System Design Guidelines)[4]を参照の上，Workstation PC 97 と Office PC 98 をベースに定義した．これは性能目標を設定する上で重要である．Windows NT のバージョンを 4.0 に限定したのは，それまでの 3.51 から 4.0 になった時点で扱える標準コントロールが増加したこと，使用性について Microsoft 社がそれまでのガイドラインを大幅に改訂したことによる．それまでのガイドラインは基本的に 1987 年に IBM 社が SAA（Systems Application Architecture）の中で Presentation Manager を前提として提唱した CUA (Common User Access) Basic Interface Design Guide[5]準拠であった．

2.3 SWEETS の成果物

2.3.1 SWEETS/SUGAR（Standardized User interface Guideline And Rule）

SWEETS/SUGAR は，利用者インタフェースを構築する上で必要となる原則および規約を，開発担当者に対し提示する．一般的に Windows 上で利用者インタフェースを構築するためのバイブルとしては，マイクロソフト社の「Windows ユーザインターフェイスデザインガイド」[6]が存在するが，非常に多くのことがらが盛り込まれており，これの読破と理解を開発者に強いるのは，2.1 節で定義した SWEETS の目標にそぐわないことから，TEAMdesign の技法の内容も加味して，使用性の統一に

必要最小限なことがらを記述している．内容については3章で一般的な注意点も含め，解説する．

また，2.3.3項で紹介するリポジトリの運用規約（SWEETS動作環境，開発者運用ルール，管理者運用ルール，ネーミングルール等）について記述した文書もSWEETS/SUGARに含まれる．

2.3.2 SWEETS/DROP（Dynamic Reusable Object Parts）

SWEETS/DROPは，主に金融ソリューションを構築する上で必要と思われる利用者インタフェース部品群である．これらの部品は，当社で開発されたWindows NTをベースとする金融ソリューションの利用者インタフェースを分析し，使用頻度が高いと思われる部品を抽出，汎化したものである．これらの部品を使用することで，2.1節で述べたSWEETSの目標に沿った開発が可能となる．

SWEETSとしては，表1の通りActive-X部品や関数を用意しているが，開発者は必要に応じて部品を開発あるいはサードパーティから調達してくることでSWEETS管理対象とすることが可能である．新たに管理対象となったそれらの部品は，SWEETSユーザにとっては標準で用意されたものと同等に取り扱うことができる．

表1 SWEETS/DROP

カテゴリー	概要	数
1 標準入出力コントロール	Windows 標準コントロールを組み合わせた入出力用のActive-X部品	8
2 金融入出力コントロール	金融業務に特化した入出力用Active-X部品	4
3 コントロール関数	上記Active-X部品群のサポート関数	10
4 メッセージ表示関数	エラーメッセージの表示とトレースログ/イベントログの書き込みを支援する関数	10
5 共有メモリ関数	アプリケーションロジックとのメッセージ送受信を支援する関数 *4	16
6 サービス関数	(1) 日付操作 (2) 文字列操作 (3) コード変換 (4) その他	14 3 8 1

2.3.3 SWEETS/CORN（Common Object Repository for Nt）

2.3.2項で述べたSWEETS/DROPを格納するリポジトリ，およびリポジトリインタフェースツール（SWEETSエクスプローラ）を総称してSWEETS/CORNと呼ぶ．

リポジトリエンジンとしては，当社のオブジェクト指向型リポジトリUREP（Universal REPository）を採用しており，図2のような論理モデルになっている．

リポジトリに対する管理ツールとしてはUREP自体がリポジトリエディタと呼ばれるものを持っているが，ことSWEETSの運用という限定された目的のためには，運用の自由度が高すぎて使いにくい．そのためSWEETSとしてはその限定された運用に合致するように，SWEETSエクスプローラを用意している．SWEETSエクスプローラは図3に示すような，いわゆるエクスプローラタイプのビューを持っており．Windows NT操作が可能であれば直感的に理解できる構造になっている．

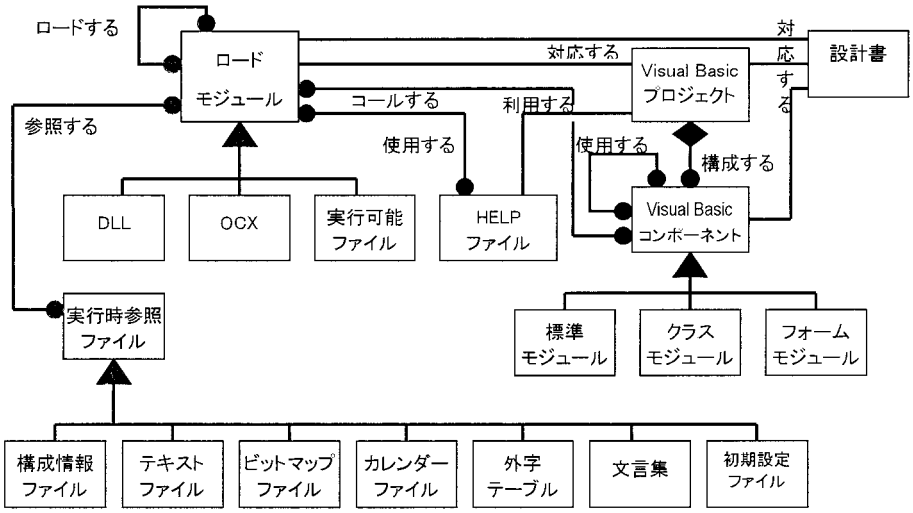

図 2　SWEETS/CORN リポジトリ論理モデル

SWEETS エクスプローラは以下の機能を持つ.

1) チェックイン　：リポジトリにコンテンツとして部品や VB プロジェクト，設計書を格納する.
2) チェックアウト：リポジトリ内のコンテンツを抽出する.
3) 部品情報参照　：個々のコンテンツの説明，作成日，作成者，バージョン，サイズ等の参照を可能にする．また，コンテンツ間の依存関係等の参照も可能にする.
4) リリース　：あるコンテンツを関連するコンテンツとまとめて別環境へコピーする.
5) バージョンシュリンク・削除：
旧バージョンのものを削除したり，新バージョンを削除し旧バージョンに戻す.
6) レポート　：フォルダ内のコンテンツの一覧情報等をレポートする.
7) ユーザ管理　：ログインによるユーザ権限（管理者あるいは開発者）の管理をする.

3. 利用者インタフェース構築上の留意点（SWEETS/SUGAR を中心に）

2 章で見たように，ソリューション構築における利用者インタフェースの重要性は相対的に増している．ここでは SWEETS/SUGAR で記述されている利用者インタフェース構築上の原則と規約の概要を紹介するとともに，注意点を考察する．ここで記述されている内容は，一見あまりに一般的なので要求者あるいは利用者と開発者の間で議論されない場合が多い．その結果として両者の解釈のレベルに差があり後に問題になることがある．要求者あるいは利用者は，これらの解釈が開発コストや使用性に直接インパクトを持つことを理解する必要がある.

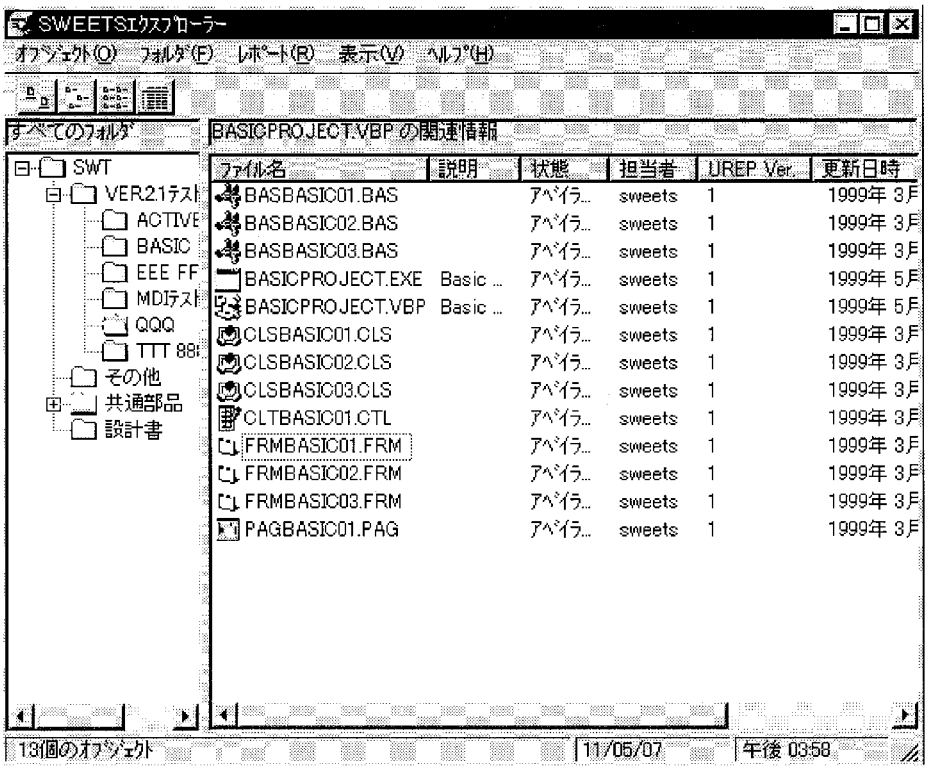

図 3　リポジトリエクスプローラ初期画面

3.1　利用者インタフェース設計上の原則

3.1.1　利用者とは

早い段階で，ソフトウェアが対峙しなければならない利用者を考慮することは，非常に重要である．人間工学的な考察が必要であると同時に，想定している利用者が既に Windows 操作に慣れ親しんでいるのか，それとも従来の CUI（Character User Interface）操作に取り残されているのか，あるいはその両方を対象にしなければならないかによって，以下に述べる原則の解釈には差が生じる．想定する利用者のレベルと以下の原則の解釈について要求者との間で合意しておくことが望ましい．

3.1.2　一　貫　性

ソフトウェアの外見及び動作は，そのソリューションおよびソフトウェア全体を通じて一貫性がなければならない．また，利用者が触れる可能性のある他のソフトウェア群（例えば MS-Office）との一貫性も考慮する必要がある可能性もある．この一貫性のレベルが高ければ高いほど，機能追加時の「直感的操作」を期待できる．一貫性をどのレベルにするかは利用者との間で早い段階に合意する必要がある[7]．

1）　手続きの一貫性

ある操作によって利用者が期待する処理は，ソフトウェア全体を通じて共通であることが望ましい．印刷操作によって，ある場合には利用者が見ている画面，ある場合にはファイル全体を印字するという処理の違いは好ましくない．

2）　物理的一貫性

メニュー項目や主要なオブジェクトはソフトウェア全体を通じて，同じ外見をもち，同じ場所に表示されることが望ましい．同じ意味の情報が画面によって違う場所にあらわれることは利用者に必要以上の緊張を強いることになる．

3) 操作の一貫性

操作の結果は，ソフトウェア全体を通じて共通であることが望ましい．ある場合にはダブルクリックで実行，ある場合にはシングルクリックで実行という操作の違いは操作ミスを誘発する．

4) 語彙的一貫性

同じ名前のコマンドは，ソフトウェア全体を通じて共通の意味を持ち，同じ機能を実行する他のコマンドがあってはならない．操作の対象オブジェクトの違いによって違う名前のコマンドを用意する（「削除」,「イレーズ」等）のは，利用者に意味の違いを考えさせることになる．

5) 外的一貫性

機能追加やソフトウェア追加した場合の外見と動作は，同一の原則に従う必要がある．つまり一旦決定された使用性原則は，簡単には変えられないことに注意する．

3.1.3 単　純　性

操作が単純であることは，利用者にとって記憶しなければならない量が少なく，学びやすいことを意味する．ある処理を行うために複数の手順が必然であり，利用者が入力の順序や形式，また他のことを覚えなければならないようならば，必要に応じてウィザード形式で誘導することを考慮する．

3.1.4 柔　軟　性

これは想定する利用者のレベルに依存する．Windows 操作の熟練者と初心者の両方に対応しなければならない場合は，基本的な操作を初心者に合せて，熟練者にはショートカットの道を用意する．可能であれば構成オプションを用意し，ある程度のカスタマイズを利用者に許すことが望ましい．

3.1.5 利用者によるコントロール

ソフトウェアに制御されるのではなく，利用者がソフトウェアを制御しているという感覚を持つと，利用者の満足渡が向上し生産性が向上すると考えられている[8][9]．開発者は忘れがちであるが，利用者インタフェースの目的は操作を強要することではなく，利用者が進めたいと思っている処理を助けることにある．

1) 能動的インタフェース

利用者を受動的立場に置かないことが必要である．受動的な立場に置かれた利用者が集中を持続することは困難であり，必要なときの利用者の応答が悪くなることがある．開発者は，処理を自動化することが生産性をあげることにつながると考えるし，それが最善の策だと考えがちだが，自動化が必要な場合には，そのトリガは利用者によって与えられることが望ましい．また，自動化に要する開発コストは一般的に自動化しない場合よりも高いはずである．

2) 操作の中断

利用者が操作を取りやめることを許す．できれば全ての画面にキャンセルボタ

ンを用意し，利用者が何も操作を進めなくても終われるようにすることが望ましい．これにより，利用者は操作に躊躇することが減り，前述の能動性はさらに高まる．

3） カスタマイズ

想定した利用者のレベルの幅が広い場合は，利用者にカスタマイズを許すことが望ましい．しかしその範囲については充分な検討が必要である．カスタマイズには適当な規定値が必要であるし，カスタマイズによって，ソフトウェアの処理が影響を受けるような場合や，操作支援組織（ヘルプデスクや事務指導等）の負荷が高くなることがある．

4） モードの排除

モード（特定の指令以外は許さない状態―モーダル―3.2.3 項参照）は極力排除することが望ましい．特にそれが，利用者インタフェースの理由だけで発生し，アプリケーションロジックやデータベースアクセスロジックからの必要性が無い場合は再考することが必要である．モードの設定が避けられない，またはそれが最善の策である場合は，そのモードにあることを視覚的に明示する．

3.1.6 応　答　性

利用者の操作に対して，ソフトウェアは何らかの反応を示す必要がある．特に，処理に時間がかかる場合は進捗状態を示す．通常，クライアントサーバ 3 層モデルにおいてプレゼンテーションロジックつまり利用者インタフェースは，アプリケーションロジックの処理の進捗を知らないため，単独では進捗状態を示せない．その意味で，必要な応答性を確保するためにはアプリケーションロジックにも影響があることに注意する必要がある．

3.2 設計上の注意点

3.1 節の原則に関してその解釈が決定したら，具体的な利用者インタフェースの設計に入ることになる．ここでは，設計を進めていく上で注意すべき点について述べる．

3.2.1 利用者インタフェース開発サイクル

Windows 上の利用者インタフェース開発における最大の課題は，設計段階に，その使用性をドキュメントに表現しきれないことにある．CUI[*5] との大きな違いがここにあり，マウス操作に伴うイベントを正確に記述するには膨大な表現が必要なだけでなく，要求者の理解を得ることは難しい．したがってこのクリティカルな工程を進めるために一般的には図 4 の反復開発サイクルが推奨されている[10]．

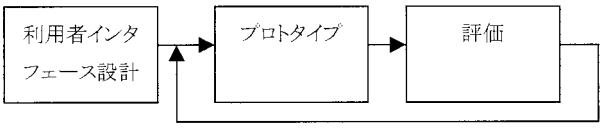

図 4　利用者インタフェースの開発サイクル

ここで注意しなければならないのは，利用者インタフェースの開発サイクルは，ソリューションを構成する他のソフトウェア（アプリケーションロジック，データベースアクセスロジック等）の開発サイクルと必ずしも一致しないことである．つまり上

記サイクルは，実際にはかなり初期の段階で繰り返される必要があり，全体工程における論理設計の段階まででプロトタイプのコードも含め，ベースライン[*6]として確定がなされるべきである．図5は，TEAMdesignによる上流工程標準モデルの内容をまとめたものであるが，最初のプロトタイプが非常に早い段階でなされていることに注目して欲しい．なお，TEAMdesignでは利用者インタフェース設計手順として，業務シナリオの記述，ユーザタスクモデル[*7]とユーザ概念表[*8]の作成を求めているが，いずれも要求定義段階で最初のベースラインを確定することを推奨している．

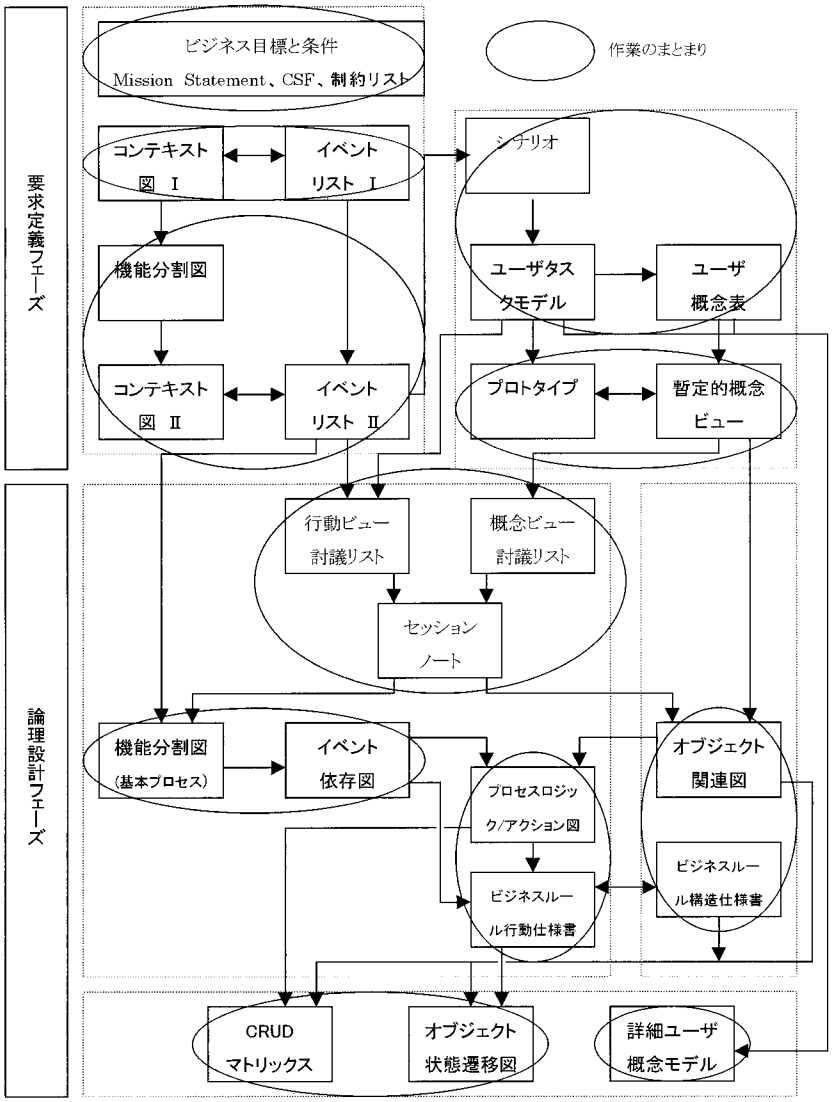

図 5　TEAMdesignの上流工程作業と成果物の関連図

3.2.2　ウィンドウモデルの選択

Windows上で利用者インタフェース設計をする場合にまず考えなければならないのは，利用者の作業環境（そのソフトウェアあるいはソリューションと利用者が会話

するための作業場所）を Windows デスクトップ上にどう表現するかということである．これは，そのソフトウェアの扱おうとしているオブジェクトまたは仕事をどのように表現したいかに関わる．例えばトランザクション処理の利用者インタフェースを構築する場合，同時に複数トランザクションを扱う必要があれば，MDI（Multiple Document Interface）モデルを選択するのが一般的である．この場合，図 6 でわかるように子ウィンドウという形で表現される概念（Document）は個々のトランザクション要求に対応し，親ウィンドウはサーバとのセッションに対応すると考えて良い．またこの場合は各トランザクションに共通の処理（セッション制御，トランザクション選択，共通ヘルプ等）や情報（セッション状態，利用者情報等）を親ウィンドウで扱うことができる．

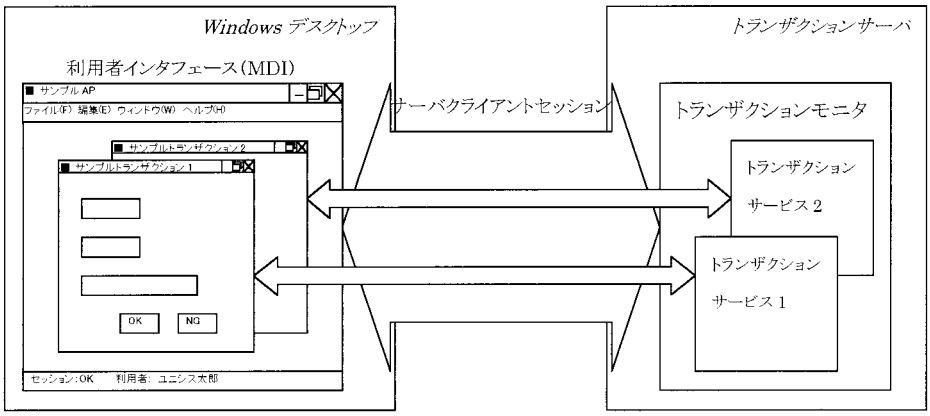

図 6　MDI（Multiple Document Interface）モデルの例

これに対し，トランザクションを同時に複数扱うような必要が無い場合はセッション制御とトランザクション制御が結果的にほぼ同義になるため，図 7 のように SDI（Single Document Interface）モデルを採用することもできる[*9]．

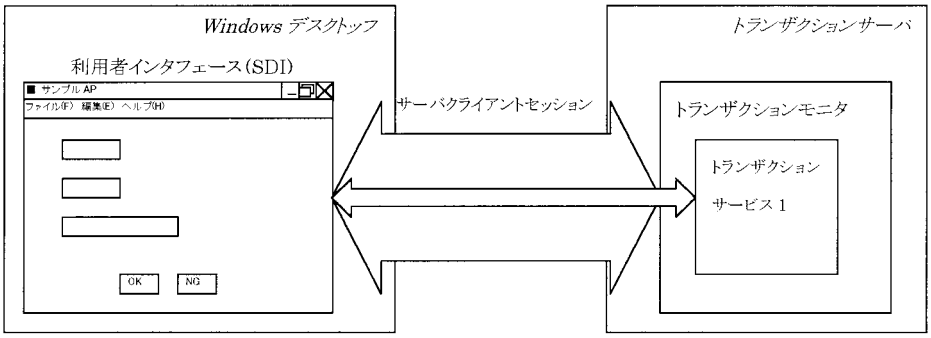

図 7　SDI（Single Document Interface）モデルの例

SDI の良いところは，各画面をそれぞれ実行ファイルとして生成してしまえば，画

面選択等の制御を Windows のシェル自体に任せてしまうことができる点にある．一方で複数の画面共通の制御や情報共有のために別の仕組みを考えなければならないこと（経験的には画面遷移の制御がスパゲッティに陥りやすい），Windows 上の他のソフトウェアと作業環境を共有しなければならない（MDI では親ウィンドウが作業環境となり，それを超えて子ウィンドウを制御できない）ことなどは問題になる可能性がある．

SDI あるいは MDI，いずれのモデルを選択するかは一般的に開発者の責任として扱われることが多いが，実は前述トランザクション処理の例のように業務要求によって選択の基準が示されていることが多い．どちらにするかで実装は大きく異なるので，なぜそのモデルを採用したのかという論理的な文脈を失わないよう常に留意する必要がある．また，この問題を先送りしどちらかのモデルに仮置きしてプロトタイプ等を進めてしまうと結果的に次工程で大きな手戻りの可能性があるため，要求者および利用者と合意しておくことが望ましい．

なお，SDI，MDI 以外にも Workspace，Workbook，Project などのモデルが定義されているが[11]，特別な理由が無い限りはどちらかにするべきである．これは後に機能追加やソフトウェア追加を行った場合に特殊なモデルでは表現ができなかったり不自然であった場合に，全体のモデル変更か使用性の不統一に陥るからである．

3.2.3 モーダルとモードレス[12]

利用者インタフェース設計をする場合に考慮を要するのが，どのようにモーダルな画面遷移とモードレスな画面遷移の表現を設計書上に表現するかである．Windows 以前にはシングルウィンドウシステムであり，基本的に全ての制御がモーダルに進められたため，特に考慮は必要無かった．しかし，Windows 環境においては，あるウィンドウで一連の操作が要求されるときに他のウィンドウに制御を移すことは可能であり，それを許すかどうかはソフトウェアの責任である．また，フォーカスを移動したときにウィンドウの共通領域（例えば MDI のメニュー）のどの項目を使用不能（Disable）にするかもソフトウェアの責任である．したがって利用者インタフェースがある状態のときに，それがモーダルな状態かモードレスな状態かは設計段階で決定しておかなければ，使用性の解釈で齟齬が生じるか成果物の品質低下をもたらすことにつながりかねない．なお，3.2.1 項で触れた TEAMdesign ユーザタスクモデルは，表記法として両者を区別することが可能である．

3.2.4 抽象化と再利用性

利用者インタフェース設計段階でユーザタスクモデルを作成すると，抽象度の低いオブジェクトと操作（「預金口座を一覧表示する」，「金利を一覧表示する」等）が明らかになる．プロトタイプの作業に入るためには，これらを Windows 標準コントロールのレベル（テキストボックス，リストボックス，コンボボックス等）に写像する（抽象化する）必要があり，一般的に，この過程は利用者インタフェース開発者の知識に依存する．しかし，これら UI 部品（利用者インタフェース構成要素）を TEAMdesign のユーザ概念表（図 5 参照）に展開することで，経験に依存しないで抽象化の指針とすることができる．これは対象オブジェクトとそれに対する操作を表形式にしてオブジェクトや操作の種類で分類する方法であり，結果として「預金口座」を「一

覧表示」するのと「金利」を「一覧表示」するのは非常に似たUI部品を必要とすることが導き出される．もちろん業務的には預金口座と金利は違うものであるが，業務的なオブジェクトやルールを考慮しなければならないのは主にアプリケーションロジックやデータベースアクセスロジックの責任であって，利用者インタフェースにとっては「…を一覧表示」するという概念が重要である．言語によってはUI部品についても継承という概念を支援するものもあるが（例えばPowerBuilder），VBではそれができない．そのため，さらに汎化を進めて言わばスーパー部品としてWindows標準コントロールを導出するよりは，既存のUI部品（SWEETS/DROP）の在庫を参照し，似たものを探し出して再利用するほうが重要になる．これがSWEETS/CORNの提供する機能である（2.3.3項参照）．

3.3 利用者インタフェースの評価

3.3.1 使用性評価基準の設定

プロトタイプあるいは最終成果物としての利用者インタフェースは，機能性だけでなく使用性についても評価されることになるが，その場合の評価基準は3.1節で述べた原則と3.2節でさらに詳細化した利用者インタフェース仕様が実現されているかどうかということになる．これを一般的な範囲でチェックリスト化したものが，SWEETS/SUGARの利用者インタフェースチェックリスト[13]である．

利用者インタフェースチェックリストは以下のものがあり，チェック項目例として示したような確認内容が設定されている．各々のチェック項目はそれぞれ判定としてOK，NG，あるいはペンディングで採点する．実際にこれらのチェックリストを適用するためには3.1節で述べた各々の原則についてどのレベルで準拠するかの議論を反映する必要がある．

1） ヒューリスティックチェックリスト*11（TEAMdesign Heuristic Walkthroughより）
チェック項目例：
「利用者の立場から見た場合に，動くはずなのに動かない機能はないか？」
「設計者側から見て，利用者の要求を変更したり削除したものはないか？」等

2） 利用者インタフェース設計基準についてのチェックリスト
チェック項目例：
「ビジュアルレイアウトは直感的になっているか？」
「利用者がカスタマイズできるようになっているか？」　等

3） （狭義の）使用性に関するチェックリスト
チェック項目例：
「通常の操作を行う場合，キーボードとマウス間の手の移動量が最小限になっているか？」
「キーボードだけで全ての操作ができるか？」等

4） 一貫性に関するチェックリスト
チェック項目例：
「全体を通じてカラーの使用方法に一貫性があるか？」
「ダイアログボックス上のボタンの配置はMicrosoft標準に合致しているか？

メッセージボックスを除く全てのダイアログボックスはすべてヘルプボタンを持っているか？」等

5) 用語に関するチェックリスト
チェック項目例：
「使用されている用語がヘルプファイルに含まれているか？」
「エラーメッセージには問題点とともに対処法が記述されているか？」等

6) フィードバックに関するチェックリスト
チェック項目例：
「重要な処理をする場合, 音または映像により事前に利用者に知らせているか？破壊をともなう処理を実行する前に，利用者の確認をとっているか？」

3.3.2 品質基準の設定

利用者インタフェースが CUI から Windows になった時点で，設計の原則が大幅に拡張されたことと同様に，開発者の戸惑いを誘ったのは最終成果物としてのソフトウェアの品質確保の問題であろう．少なくともコード量が圧倒的に増えたことで（1 章参照），それに比例した品質基準（例えば不具合検出数/KLOC）を採用している場合，それを満たすために必要なテスト項目数は当初想定できなかったほど飛躍的に増加したはずである．しかし実は，この現象は決定的に設計原則の拡張に依存しているのであって避けて通る方法はないと考えるべきである（テストツールによるテストの自動化はある程度期待できるが，必要なテスト項目が減るわけではない）．したがって，設計原則の準拠度合いと設計基準が決定した時点で，品質基準の設定については見直されなければならない．また，3.3.1 項で見たように利用者インタフェースの開発サイクルが反復型であることから，不具合検出数のカウント方法についても開発プロジェクトで統一した基準をもつことも必要である．

4. SWEETS の課題

SWEETS プロジェクトは，Windows NT 上の利用者インタフェース開発を前提とする原則や規約の提供，VB 部品の提供，VB 部品をコンテンツとする必要最小限の管理機能の提供という三つの目標を掲げて実行された．ここで，上記前提条件の範囲で目標毎に課題を整理しておく．

4.1 各種規約・ガイドの整備

この目標は，具体的には，ソリューションあるいはソフトウェア間で使用性が統一されていない状況を打破するために設定された．これについては原則だけでなく，具体的な設計基準を半ば強制的にでも統一する必要があるだろう．好みの問題はあるだろうが MS－Office を構成する各ソフトウェアは，使用性がかなり統一されているだけではなく，直接的に使用性には関わらない部分の Look ＆ Feel の類似性にも気が使われている．ソフトウェアの起動→（色調，サイズ，表示場所の統一された）ロゴの表示→（ロゴ表示中になされる）MDI の初期化→デフォルト Document の表示という最初の段階で，いかにもこれらのソフトウェアが Suite であることを思い知らされる．これは，表面的には単純なことであるが，本稿で論じた原則以上に一歩踏み込んだ設計思想の標準化と部品化のための規約作りがなされていることを示している．

TEAMdesign やその他の標準的な開発方法論との整合という面で，ユーザタスクモデルやユーザ概念表という技法との関連はかなり整理されたが，具体的に SWEETS/CORN で提供している SWEETS エクスプローラのビューとユーザ概念表（例えば Excel シート）の対応付けができれば， 上流工程での生産性向上に役立つはずである．今一歩の議論が必要であると思われる．

また品質の均一化ということに関しては，次節の部品化の推進もさることながら，設定された規約の遵守をどのように計測するかということのほうが大きい．チェックリストを提示できたことは評価できるが，さらにメトリックスを明確にする必要があると思われる．

4.2 再利用可能な金融ソリューション用共通部品の整備

これについては，現状金融ソリューションにおける利用者インタフェースの範囲での網羅性はかなり高いと思われる．しかし，現状では VB 部品に特化していることことから，JAVA 部品や COM 部品等，今後利用者インタフェース開発において主流になると思われるものの整備が必要かもしれない．

4.3 部品の管理機能

プロジェクト立上げ当初想定していた単純な部品の格納場所という機能範囲に比べて，かなり大幅な拡張が行われた．これは単純な部品の格納では実開発時に手間こそ増え，何も生産性の向上につながらないと判断したためである．結果的に構成管理（Configuration Management）ツールと機能的に重複する部分が発生したが，VB に特化したことで運用が単純化された点は使いやすいという評価も可能である．反面，想定したプロジェクト規模が利用者インタフェース開発で 200～300 人月程度までということで，大規模開発や分散開発環境が作れないという中途半端さもある．また，構成管理ツールとしてみた場合，標準的な開発工程が定義できなかったため，上流成果物の格納や下流成果物との関係の保持等の機能が不充分で拡張の必要がある．

5. お わ り に

SWEETS は適用事例がまだ少なく成長過程の緒についたばかりであるが，前章で述べたプロジェクトの前提条件そのものを見直すことが必要かもしれない．対象言語を VB に限定したことや対象システムを利用者インタフェース（プレゼンテーションロジック）に限定したことが，SWEETS の発展に対する足かせになることも考えられる．この点については今後も継続的に議論をしていきたい．

しかし一方で，不充分ではあったが利用者インタフェース開発にあたっての原則やチェックリストをまとめられたことは，現実的に役立つという意味で大きな成果であった．

2 章でも触れたが，ソリューション全体から見ればとかく二次的な問題と捉えられがちだった利用者インタフェースがようやく注目されてきている中で，この成果が役立てばこれに勝るものは無い．

多くの方々の協力無くしてこのプロジェクトは成り立たなかった．SWEETS プロジェクトのチャンスを与えていただいた方々，その遂行に関わった全ての方々に対し，ここに感謝の意を表する．

＊1　本稿の中でソリューションと表現する場合は，業務的な特定の問題領域に対する解決手段となるベンダ提供のソフトウェア群あるいは企業情報システム全体を示し，ソフトウェアと表現する場合は，そのソリューションの一部を構成するコンピュータソフトウェア，あるいは特定のソリューションに限定されない一般的なコンピュータソフトウェアを指す．

＊2　これは，筆者の経験してきた金融機関向け勘定系端末ソリューションの歴史においても明らかである．当社の提供する金融機関営業店向けソリューションは，おのおのの時代で機能拡張はあったものの，1970 年代から 1980 年代前半の金融専用ハードウェア時代に 5,000 ステップ前後，1980 年代後半から 1990 年代前半の CUI（Character User Interface）ベースの汎用ハードウェア時代に 50,000〜150,000 ステップ程度であったが，Windows NT 上で動作するソリューション FBA Navigator では 1,000,000 ステップに及んでおり，GUI（Graphical User Interface）化/Open 化が進むにつれてコード量が飛躍的に増大している．

＊3　ユニシスでは，インフォメーションサービス部門がサービスを提供する上で使用する手法，技法およびツールを体系化して TEAMmethod（チームメソッド）方法論と呼んでいる．TEAMdesgin（チームデザイン）方法論は，この TEAMmethod の構成要素であり，情報システム設計について体系化したものである．TEAMmethod では，他にも，情報化計画，ビジネスプロセスの再設計，情報システム構築およびプロジェクト管理についても体系化している．

＊4　現在，SWEETS で用意しているメッセージ関数は，同一 CPU 内にアプリケーションロジックが存在することを想定しており，アプリケーションロジックの位置透過を実現するものではない．アプリケーションロジックの位置透過のためには，メッセージトランスポートに何を使用するかを決定した上で現状インタフェースに合せて仕組みを拡張する必要性がある．

＊5　ここで述べる CUI はキャラクタベースのシングルウィンドウという外見を持ち，キーボードによる操作を前提とした利用者インタフェースのことである．PC 上でキャラクタベースのマルチウィンドウの外見を持ちマウス操作できるプラットフォームや，マルチウィンドウ，マウスインタフェースをその内部に実装したソフトウェアも存在したが，特に日本では一般的とはならなかった．

＊6　プロトタイプには 1）プロトタイプに適した言語を用いて，外部仕様の確認だけを目的に実装し後の工程で使用しない方法，2）プロトタイプを実際の開発のベースとして使用する方法，の 2 種類がある．一般的には後者が生産性の面からも良いと考えられており，この場合プロトタイプの成果物は使用性のベースラインとなるだけでなく，最終成果物に向かっての中間成果物として扱われることになる．

＊7　ユーザタスクモデルは，TEAMdesign の採用している主要技法のひとつで，実際の業務シナリオを利用者の観点から視覚化したものである．視覚化の目的は利用者から見た業務の流れを明確にすることと利用者インタフェース上の使用性について包括的な合意を得ることである．この過程で業務シナリオの文章表現の不備や不整合が明確になるはずである．また，このモデルで着目する「フォーカス」という概念は，論理設計工程における「機能分割」の「基本プロセス」のヒントになる．さらに，このモデルで導入される「モーダル」「モードレス」という概念は視点の移動の明示と共に，後に分析するイベント依存性（「前置」，「トリガ」，「保護/制御」）という概念に結びつく．

＊8　ユーザ概念表は，TEAMdesign の採用している主要技法のひとつで，汎化という形で「利用者インタフェース構成要素」の抽象化を取り扱う．しかし，その分析結果をどのように利用するかは言語に依存するところが大きい（VB は利用者インタフェース構成要素の継承を支援しない等）．また，ユーザ概念表で論じられる「継承」や「再利用」という概念は開発者の都合で導出されるものであり，業務要件からくるものではないことに注意．

＊9　MDI を採用した代表的なソフトウェアの例としては Microsoft Excel，Word，Access，PowerPoint などの MS-Office 製品，SDI の例としては Windows に付属のメモ帳がある．一般的にオブジェクト指向論者には SDI が好まれてきた（SDI では対象オブジェクトを選択した場合に処理が決定されるのに対し，MDI では処理―Word 等―を選択してからオブジェクト―文書―を選択するという印象が強いためである）．しかし，MDI にも本論で述べたように捨てがたい魅力があり，論理的な意味付けさえあれば MDI モデルでも良いと筆者は考える．MS-Office シリーズの使用性に統一感があるのは，このモデルの統一が大きく寄与してことは否定できないであろう．なお，最近では，SWEETS エクスプローラでも採用した Explorer モデルのものも多くなっている．

＊10　モーダルとは，「ウィンドウまたはダイアログボックスで他のフォームやダイアログボックスへフォーカスを移せるようになるまでに，利用者に対しいくつかのアクションを要求するもの」を指し，モードレスは「ウィンドウまたはダイアログボックスで他のフォームやダイアログボックスへフォーカスを移すために，特別な利用者のアクションを要求しないもの」を指す．

＊11　発見的ウォークスルー（Heuristic　Walkthrough）は，TEAMdesign で採用している主要なレビュー手法のひとつで，製品が使いやすく役に立つかどうかをレビューするものである．このレビューは開発プロセスを通じて，同一基準で何度も繰り返す必要がある．

参考文献

［1］ Microsoft Corporation, The Windows Interface Guidelines for Software Design, Microsoft Corporation web site（www.microsoft.com）, 1997, User-Centered Principles.
［2］ Microsoft Corporation, PC 97 Hardware Design Guide, Microsoft Press, 1996.
［3］ Intel Corporation and Microsoft Corporation, PC 98 System Design Guide, Intel Corporation web site（www.intel.com）, 1997, Part 2 : Chapter 4, System Design Issues.
［4］ Compaq Computer Corporation, Dell Computer Corporation, Hewlett Packard Company, Intel Corporation, and Microsoft Corporation, Network PC System Design Guidelines, Intel Corporation web site（www.intel.com）, 1997, pp 10 Network PC Hardware Requirements.
［5］ IBM, CUA Basic Interface Design Guide, IBM Corporation web site（www.ibm.com）, 1989.
［6］ Microsoft Corporation, Windows ユーザインターフェイスデザインガイド，アスキー, 1995.
［7］ 日本ユニシス, SWEETS/SUGAR 金融入出力標準化ツール画面作成規約, 1998, pp 6.
［8］ Microsoft Corporation, The Windows Interface Guidelines for Software Design, Microsoft Corporation web site（www.microsoft.com）, 1997, User in Control.
［9］ 日本ユニシス, SWEETS/SUGAR 金融入出力標準化ツール画面作成規約, 1998, pp 6
［10］ Microsoft Corporation, The Windows Interface Guidelines for Software Design, Microsoft Corporation web site（www.microsoft.com）, 1997, Design Cycle.
［11］ Microsoft Corporation, The Windows Interface Guidelines for Software Design, Microsoft Corporation web site（www.microsoft.com）, 1997, Selecting a Window Model.
［12］ Microsoft Corporation, Visual Studio 6.0, Visual Basic section of the MSDN Library, Microsoft Corporation web site（www.microsoft.com）, 1988, Definition.
［13］ 日本ユニシス，SWEETS/SUGAR 金融入出力標準化ツール画面作成規約，1998，pp 28-34.
［14］ J. マーチン他, オブジェクト指向方法序説　実践編, トッパン, 1997.
［15］ M. ローレンツ他, オブジェクト指向ソフトウェアメトリクス, オージス総研, 1995.
［16］ I. ヤコブソン, オブジェクト指向ソフトウェア工学 OOSE, トッパン, 1995.
［17］ J. マーチン他, オブジェクト指向方法序説　基盤編, トッパン, 1995.
［18］ 米沢明憲他, モデルと表現, 岩波講座ソフトウェア科学, ［展望］17, 1992.
［19］ J. ランボー他, オブジェクト指向方法論 OMT, トッパン, 1992.
［20］ Microsoft Corporation, Visual Basic Programmer's Guide, Microsoft web site（www.microsoft.com）, 1988, Designing with the User in Mind.
［21］ Tandy Tower, The Human Factor : The Top 10 Windows 95 User Interface Design Errors, Microsoft Corporation web site（www.microsoft.com）, 1995.

執筆者紹介

おお津昌三（Shozo Otsu）
1982 年近畿大学商経学部経済学科卒業．日本ユニシス(株)入社．オブジェクト指向型金融機関営業店ソリューション FBA　Navigator の企画・開発を経て，1997 年より SWEETS の開発に従事．現在，ビジネスソリューション四部開発三室に所属．米 PMI 会員．