SEAMAIL

Newsletter from Software Engineers Association

Volume 10, Number 1 November, 1995

目　次

編集部から		1
ソフトウェア CALS の提案	長野 宏宣	2
BMaiDvoClu モデルによる 　ソフトウエア会社のリストラクチャリング	木ノ下 勝郎	7
制作の意志とオブジェクト指向技術	熊谷 章	15
Visual Basic の光と陰	桜井 麻里	24
広島原爆 WWW プロジェクト終わる	大場 充	33

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会 (SEA) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985 年 12 月に設立されました．

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200 人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，800 余名を越えるメンバーを擁するにいたりました．法人賛助会員も 40 社ちかくを数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州，広島，東北の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます．SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

代表幹事： 山崎利治

常任幹事： 大場充　熊谷章　坂本啓司　中野秀男　深瀬弘恭

幹事： 青山幹雄　荒木啓二郎　井川裕基　市川寛　伊藤昌夫　菊地俊彰　君島浩　窪田芳夫　酒匂寛　塩谷和範
篠崎直二郎　杉田義明　高橋光裕　武田淳男　田中一夫　玉井哲雄　中来田秀樹　中谷多哉子　西武進
野中哲　野村行憲　野呂昌満　平尾一浩　二木厚吉　堀江進　増井和也　松原友夫　松本理恵　盛田政敏

事務局長： 岸田孝一

会計監事： 辻淳二　吉村成弘

分科会世話人 環境分科会 (SIGENV)：塩谷和範　田中慎一郎　渡邊雄一
教育分科会 (SIGEDU)：君島浩　篠崎直二郎　杉田義明　中園順三
ネットワーク分科会 (SIGNET)：小林俊明　人見庸　松本理恵

支部世話人 関西支部：臼井義美　江谷典子　中野秀男　盛田政敏　横山博司
横浜支部：野中哲　藤野晃延　北條正顕
長野支部：市川寛　小林俊明　佐藤千明
名古屋支部：筏井美枝子　鈴木智　平田淳史
九州支部：武田淳男　平尾一浩
広島支部：大場充　佐藤康臣　谷純一郎
東北支部：菊地俊彰　野村行憲　和田勇

賛助会員会社： 岩手電子計算センター　ＮＴＴソフトウェア研究所　ＰＦＵ　ＳＲＡ　アスキー
エスケーディ　オムロンソフトウェア　カシオ計算機　キヤノン新川崎事業所
さくらケーシーエス　サン・ビルド印刷　ジェーエムエーシステムズ　ジャストシステム
ダイキン工業　ニッセイコンピュータ　ムラタシステム　リコーシステム開発
安川電機　構造計画研究所　三菱電機セミコンダクタソフトウェア
三菱電機関西コンピュータシステム　新日鉄情報通信システム
新日本製鉄エレクトロニクス研究所　池上通信機　中央システム
東芝アドバンストシステム　東電ソフトウェア　東北コンピュータ・サービス
ＳＲＡ東北　日本ＮＣＤ　日本情報システムサービス
日本電気ソフトウェア　富士写真フィルム　富士通　富士通エフ・アイ・ピー
オムロン　ＳＲＡ中国　（以上 37 社）

ＳＥＡＭＡＩＬ　Ｖｏｌ．１０，Ｎｏ．１　　１９９５年１１月１日発行
編集人　岸田孝一
発行人　ソフトウェア技術者協会（ＳＥＡ）
〒１６０　東京都新宿区四谷３－１２　丸正ビル５Ｆ
TEL: 03-3356-1077　FAX: 03-3356-1072
印刷所　サンビルド印刷株式会社　〒１６２　東京都新宿区築地町８番地
定価　５００円（禁無断転載）

編集部から

☆

なんとか月刊ペースに戻りつつあるSEAMAILです．ああ，しんど!

☆☆

巻頭に載せた長野さんの"Software CALS"への提案は，去る7月に立山で行なわれたIPSJのワークショップで発表されたものの再録です．

SS95チュトリアルでお話された窪田さんも交えて，いずれ，SEA Forumで，われわれにとってCALSとは何であるのかを議論してみるのも面白いと思われます．

☆☆☆

次の木ノ下さんの提案も，別の視点から，ソフトウェア業界の現状改革を訴えた熱いペーパです．SEAに入られたのは最近ですが，実は15年前，アメリカはロッキー山中で開かれたワークショップで，岸田，熊谷，中園ほかのSEA会員(もっともそのころまだSEAはなかった!)たちと同宿していました．

☆☆☆☆

おなじみのクマさんからは、名古屋·南山大学での特別講義を機会に、ソフトウェア開発についての考えをまとめた論文が寄せられました．さて，他の講師の方々はいかに?

☆☆☆☆☆

毎月興味深い議論を展開しているらしいYDOCに，たまにはレポートを書いてほしいと要望したところ，7月のスピーカだったML管理人の桜井さんから，さっそく原稿が届きました．他の支部の方々もよろしく．

☆☆☆☆☆☆

SEA広島支部のニュース(on Internet)からは，大場先生のWWW原爆博物館プロジェクト·レポートを転載させていただきました．ほんとうにおつかれさまでした．

☆☆☆☆☆☆☆

ソフトウェアCALSの提案

長野 宏宣
(NYTソフトウェア研究所)

概 要

本提案では，ソフトウェアCALSへの取り組みについて，2万台のワークステーション(WS)からなるインターネット技術をベースとした大規模分散ソフトウェア開発環境を構築運用した筆者自身の経験，および十数年にわたってソフトウェア･エンジニアリングに関する標準化活動に従事してきた立場から，私的な想い入れも含めて述べてみたい．

1. はじめに

ソフトウェアはそもそも電子的な製品であり，他の産業分野の製品に比べると，計算機ネットワークを利用した分散開発および情報流通を基本とするCALSに適している．最近は，ソフトウェア･ライフサイクルを支援する分散環境を，計算機ネットワークを活用して構築し，この上で，従来紙ベースで作成されてきた仕様書などの文書が電子化されてきている．このように，製品および関連する中間製産物がほとんど，電子的に取り扱い可能となってきており，ソフトウェア産業は，本来CALSのような，電子化と計算機ネットワーク化を基本とする仮想企業の形態に乗りやすい特性を持っている．しかし，ソフトウェアをCALS的にやろうという議論は，これまではあまり表面に出てこなかった．その理由は，欧米では，すでにそのような形態で仕事をしていてあまり話題にならないからだと推察されるのだが，日本では，以下のような情けない状況が考えられる．

(1) 紺屋の白袴： CALSが，一般の製造業の仕事の過程の見直しとしてのBPR (Business Process Re-engineering)，およびネットワークを活用した情報交換の即時化にもとづくCE (Concurrent Engineering)による業務の抜本的改革を狙いとしているため，ソフト産業の関係者は，自らの仕事の見直しよりも，他産業を向いたビジネスチャンスとみなして取り組んでいること．

(2) 閉鎖系列と閉鎖環境の形成： 日本のソフト産業は，これまで，主として汎用機による大規模なシステムをビジネスのベースとしてきたため，CALSのようなオープンなネットワークを用いた共同作業よりも，従来から築きあげてきた系列個別の閉鎖的なネットワークの活用で，ことがすんできた．このことは，他産業に比べたとき，ソフトウェア産業の商用インターネットへの加入が少ないことからも推察できる．つまり，日本のソフト産業は，一部の独立系企業を除くと，メーカなどの系列を通じた人材の提供をビジネスの基本とし，その系列のお仕着せの開発環境ですませてきたため，インターネットの必要性を現時点では感じていないと推察できるわけである．そのお仕着せの環境が，従来通りの人を集中させる計算機センターであるとなれば，状況はさらに危機的である．

2. 企業グループに閉じたソフトウェア開発環境

インターネット技術が，研究者の情報交換から，種々の実用作業に用いられるようになったのは，たかだか，ここ5～6年のことである．

ソフトウェアの世界でインターネット技術を本格的に実務に適用したのは，開発環境の分野であった．これは，メインフレームによる集中型の開発環境から，Unix WSサーバとクライアントからなるLAN環境，これらを通信回線で相互に接続した企業内インターネットをベースとする分散開発環境へと移行する企業が出てきたことに始まる．しかし，これらのネットワークの運用は，あくまでも企業内あるいは企業グループに閉じたものであって，CALS等で目指すオープンなネットワークからは程遠い．その理由には，現状のインタネットのセキュリティ･レベルが必ずしも満足できるものではないこと，現時点では企業グループに閉じた運用で十分であり，関係の薄い他のグループとの情報の交換の頻度が低いため，その必要性を感じていないこと，などが指摘できよう．

とはいえ，独立系のソフトハウスなどにとっては，このように企業内に閉じたネットワークの恩恵を受けるチャンスが少なく，また，いくつかのグループとの付き合いのためには，それぞれに閉じた専用の通信回線や端末などの準備が必要であるなど，オープン化を望むグループも存在している．さらに，PCソフトを主たる業務とするソフトハウスでは，すでにインターネットを業務のインフラとし，開発から販売までのプロセスをこの上で自然に実践している会社も存在していることを挙げておきたい．

日本のソフトウェア産業全体にとっては，グループ内の専用ネットワークをオープンな商用インターネットへ接続して，本格的に多様な相手との情報流通の手段として活用すること，および，その開発/流通環境としての活用について，真剣に考えるべき時である．

3. ソフトウェア開発の国際分業化と国際標準化

ネオダマのソフトウェアの開発から利用者へ提供されるまでの流れは，これまでの汎用機のソフトウェアとは，大いに異なる．特に，その品質や仕様決定の過程は，旧来の常識とはかけはなれている．

WS や PC のソフトウェアは，フリーウェアなどの形でインターネット上のサーバにソース･プログラムとして提供されるなど，最初は無料あるいは小額の賛助金を支払うことで容易に入手･利用できるものが多い．これらのソフトウェアの仕様は，その後徐々に改版され，同時に品質も改善される．無償の時には保守上の保証もなく，グループや個人に閉じた研究や趣味のための利用が前提となっており，バグなどの責任も，利用側の問題とみなされるのが常識である．その後，そのソフトウェアの市場化の見込みが出てくると，ビジネスとして品質およびマニュアルなどの文書を整えて，通常のソフトウェアの流通経路に乗った製品が提供されるようになる．

このような形態をとるには，利用する側も提供する側も，同じオープンなネットワーク上で情報交換ができることが前提となる．最近の WWW サーバと MOSAIC などのビューワによる爆発的なインターネットの普及は，従来専門家に閉じていたこのような最新のソフトウェアの流通の仕掛けを，専門外の利用者にも広げたという意味で，画期的である．

一方，急激な円高の影響もあって，日本国内のソフトウェア技術者のコストパフォーマンスが，アジア諸国や米欧のそれと比較すると，競争力がなくなってきている．そのような中で，インドや台湾などを筆頭に，米国の高等教育を受けた優秀な頭脳が里帰りをして，米国からの高度なソフトウェアの開発を請け負っている事実は見逃せない．

他の，主として製造業の工場のアジア諸国への流出と同様に，ソフトウェア産業においても，アジアの優秀な頭脳を低コストで活用することが，今後の課題として大きくクローズアップされている．そのような状況にあって，国内でやるべきことと，経済原則にのっとって国際分業によりアジア諸国に分担してもらうこととを，業界全体の視点から，一度整理して考えてみる必要がある．たとえば，インドの数学的な能力の高さは著名であるが，その能力を活用するようなプロセスを切り出すといったことが考えられる．

そのためには，一品生産のソフトウェアであっても，その開発から運用･保守に至るライフサイクルの各作業プロセスを，IS12207(JIS 化作業中) などの世界標準の定義をもとにして，実際の作業に適用する具体的なプロジェクトを試行することから始めなければならない．さらに，その時の品質保証についても，たとえば IS9000 シリーズなどにもとづいた組織の監査制度などを有効化していく努力も必須である．IS9000 シリーズのソフトウェアプロジェクトへの適用については，その妥当性についての議論が残されていることも事実であるが，今後の国際分業を考えたとき，寄って立つべき相互理解のきっかけとしては無視できない．

この他に ISO/IEC JTC1/SC7(ソフトウェア･エンジニアリング) では，国際分業における相互理解にむけた規格化が進められている．これらの中には，ソフトウェア品質の特性の定義 (IS9126) などの枠組みの規格の他に，CDIF(Case Data Interchange Format) で進められている CASE ツール間のデータ交換規格など，具体的なインタフェイスとして活用できるような規格も含まれており，ネットワークを通じたソフトウェア開発プロセスの分業化への基礎がためが進められている．

4. ソフトウェア産業と CALS

ソフトウェアは，そのプロダクトがそもそも電子的であるため，他産業に比べて CALS という形態に乗りやすいことは，すでに述べた．しかし，日本のソフトウェア産業の現状は，いわゆるインターネットや PC などのオープンなソフトウェアの世界を除くと，個別の企業グループに閉じた文化が形成されており，ソフトウェアの再利用や流通などは別世界の文化であったといえよう．オブジェクト指向によるソフトウェアの再利用と流通性の向上などが叫ばれて久しいが，これもまだ，きわめて一部の閉じた部分に限られている．

そのような状況の下でも，閉じた世界に，WS や PC のパッケージ (特にマルチメディア関連の機能については外部の第三者のソフトウェア) が徐々に導入され，システム全体としては，利用者自身が種々の構成管理を行なわざるを得ない状況になりつつある．このために，一旦システムの障害などが発生すると，従来のようにすべてを窓口のベンダに依頼することもできず，場合によっては，エンド･ユーザが供給先と直接情報のやりとりや交渉をせざるを得ないこともある．

こうしたソフトウェア･システム全体の構成法の変化に応じて，ソフトウェアに関する種々の情報交換を，インターネットなどのオープンなネットワークで実施する動きがでてきた．１つは，WWW などの公開サーバによるソフトウェアそのものの配布，そして，それらに関する障害/修正情報の配布である．

このようなソフトウェアは，いわゆるオープンなプラットフォーム上に提供され，未完成の状態で試用品として提供されることが多い．先進的な利用者は，早めにそれらの試用品を入手して使ってみることになる．その過程で，提供されたプログラムのバグや使い勝手，仕様上のコメントなどが，提供元に電子メールなどによって送られてくる．その情報はさらに WWW サーバや電子ニュースグループに公

開され，情報の早期の公開による重複した作業のムダが削減される．このような過程を経て製品が完成し，有償のソフトウェアとしてマーケットに出てくる．利用者は機能も品質もすでに了解しており，安心して入手することができる．

すべてのソフトウェアがこの形態をとれるとは考えられないが，この方法には学ぶべき点が多い．大規模ソフトウェアにおいては，長期間の開発プロジェクトの結果，利用者の要求とは異なるシステムができてしまって，のちのちまでのトラブルが生じるような事例には，こと欠かない．そうでなくても，大規模システムの完成には何年もかかるので，着手時点と完成時点との環境の変化から，すぐにも機能変更に追い込まれることが考えられる．このような状況を打開する方策としては，システムを，切れのよいサブシステムに分割し，それぞれのインタフェイスを明確にして，それぞれを独立した形で開発/保守できるようににしておくことが考えられる．

従来，汎用機のOS開発でもこのようなやり方がとられてきたが，システム全体の結合度が高く，さらに性能要求などからインタフェイスを崩してしまうなど，結果としては全体が一つのモノリシックな形になっていた．今後のアプリケーションの構成としては，汎用機もネットワークに接続されたWSやPCなどのノードとして扱い，アプリケーションをそれぞれの機能サーバとしてのノードに分割分散して搭載することによって，インタフェイスの切れをより明確に物理的にみえる形にして行くことが考えられる．

このアイデアは，ネットワークの速度が加速度的に早くなる近未来には案外現実的かもしれないが，当面は実験的に種々の試みをしてみる段階であろう．これが実現すると，将来のソフトウェアはゲーム･ソフトのように物理的なサーバ･ボックスとして構成されることになるのかもしれない．

要するに，国内において曲がり角にあるソフトウェア産業においては，これまでのような大規模な汎用機のソフトウェア･システムのあり方を，最近のハードウェア技術の進歩やネットワーク化の進展などの技術環境の変化，および，アジアなど新興のソフトウェア産業との分担を含めて，CALSという産業構造の動きを自らの変革のチャンスとして活かしたいというのが，ソフトウェアCALSの狙いである．

2. ソフトウェアCALSの具体化イメージ

(1) 概要

では，ソフトウェアCALSの具体的なイメージについて想像してみたい．

その具体例は，たとえばNTT社内のソフトウェア開発ネットワークに見ることができる(Nagano 1)．ここでは，利用者の要求仕様の確定以降の，ソフトウェア･ライフサイクルに対応する作業のほとんどすべてが，全国に分散したネットワーク上で実施されている．このネットワークの基盤ス技術はインターネット技術であるが，しかし，このネットワークは，世の中のインターネットとは少々異なるイメージで運用されている．

具体例を挙げれば，ソフトウェアの開発に必須のドキュメンテーションは電子ニュースよって，種々の帳票は電子メールによって，運用され，各種の情報はネットワークによって，ほとんど瞬時に配付される．結果としてのソフトウェア製品は専用サーバに格納され，開発プロジェクトの手を離れて，製品としての構成管理の対象となる．また，必要な種々の計算機資源もネットワークを通じて利用できるので，従来のようにある計算機センターに人が集まって集中作業をすることは，ごく限られた部分の作業で済む．

このように，企業に閉じたネットワークでは，現時点でも，多かれ少なかれ，ソフトウェア開発の電子化とネットワークによる距離の克服が行なわれていると思われる．しかし，いざ企業を越えてこれを実施しようとすると，現在のインターネットではいくつかの不備があり，そのままではことが進まない．

まず，セキュリティの問題が最初に指摘されよう．いわゆるハッカーによる悪意の侵入や，ビールスと呼ばれるソフトウェアによる被害の問題だけでなく，企業の経営情報や製品そのものを勝手に参照される恐れが指摘される．また，企業を越えたインターネットでは，その信頼性や速度の問題も，現時点では必ずしも満足できるものではない．

しかし，それらを理由にソフトウェアCALSは実現できないというのは，いささか早計である．インターネットにおいても，セキュリティ保障，信頼性などの技術が徐々に提供されてきている．逆に，ソフトウェア産業がそれらの条件を満たしたネットワークが提供されるまで何もしないというのは，許されないことであろう．

次に，現時点で，インターネット上で本格的に使用可能な文書情報交換手段の貧弱さが挙げられよう．文献1に挙げた事例でも，インターネットの基本ツールである電子メール，ニュースがemacsによる種々の工夫によって最大限に活用されているが，図面については，当初はほとんどを文字による組み合わせで表現するなど，いわゆる日本語ワードプロセサに慣れた人達には，きわめて不満なものであった．最近では，DTPツールの活用などにより，図面等もネットワーク上で活用している．

このように，ソフトウェアの中間製産物のほとんどが文書であるということからしても，さらに，今日では徐々に上流のCASEツールによって複雑な図面を顧客と開発側とで交換するようになってきていることからも，より高度な文書交換の手段が望まれる．この期待に応えるのが，

SGML をはじめとする文書標準規格である．従来の素朴な電子メール等による情報交換に比べて，図面等を含めたはるかに表現力の優れた文書情報の交換が可能になり，従来一部の専門家の間に閉じていたインターネットによる情報交換の，ビジネス･レベルでの活用が現実のものとなろう．

(2) ソフトウェアライフサイクルと CALS

次に，要求定義，設計，製造，試験，提供，保守の各工程が，ソフトウェア CALS においてどのように実現されるか，そして，各工程を通じたプロジェクトの開発管理がどのように実現されるのか，社内における経験をベースに，そのイメージを述べてみたい．なお，開発対象ソフトウェアとしては，大規模な企業向けのソフトウェアを想定する．

ライフサイクルの各工程を時間的に順次実施するのが，古典的なウォータフォール･モデルであるが，近年では必ずしもそのとおりには実施されてはいない．ライフサイクルを構成するより具体的なプロセスについては，先に述べた IS12207 に定義されており，これがソフトウェア CALS のベースとなろう．ここに定義されたプロセスの組み合わせは，ウォータフォール･モデルへの反省から，その組み合わせや実施の順序についてはカスタマイズすることになっており，まさに，CE への基本的な準備ができているといえよう．

要求定義： ソフトウェアの開発においてもっとも重要な工程であるが，もっとも CALS に乗りにくい．最近の情報処理システム開発の上流は，企業活動の見直しから始まるので，システム化の対象とするビジネス･プロセスそのものの分析が必要であり，基本的には，従来のように人が出かけて行って仕事をせざるを得ない．

しかし，一旦問題が認識されると，情報交換は，ネットワーク上でより効率的に実行できる．そのためのツールとして，ワークフロー分析やシステムの要求定義を行なう CASE ツールが利用できる．これらのツールのデータを格納したデータベースをリポジトリと呼び，これが，その後のソフトウェア･ライフサイクルを通じて，すべての作業の中心となる．要求定義の結果は，図表を含めた文書となり，たとえば SGML などの文書規格によって表現すれば，企業や国境を越えたネットワークによる情報交換が，容易に実現する．リポジトリは，要求仕様から最後の出荷製品までの文書や情報を完全にリンクして，ライフサイクルにおける情報の追跡性の保証，および，文書や情報の版管理を実現する．

設計： 要求定義を受けてソフトウェアの設計を行なう．まず，パッケージ･ソフトウェアを含めて，既存のソフトウェア･コンポーネントの利用を考える (要求定義の段階で決定していることもあろう)．それらによる実現の可能性がない部分を自分で開発することになる．そのための CASE ツールやシュミュレーションツールが今後ネットワーク上に揃えられていくことになろう．設計の結果は，要求仕様の文書とのリンクを保ちながら詳細化され，リポジトリに格納される．設計のある区切りにおいては，仕様の確認のためのレビューを行なう．このレビューもネットワークの上で実施できるだろう．文献 1 の事例でも，メールや電子掲示板の運用によって，大規模な性能チューニング･システム･ソフトウェア (通信ソフトウェア) の設計レビューを，ネットワークの広がりの上で実施している．

将来は，ワークフロー･ツールによって計画されたスケジュールにそって，発注元と設計者とが，ネットワーク上の CSCW ツールを利用しつつ，コメントと解答を交えながら作業を消化していく．これらのコメントは，すべてリポジトリに保存され，提起者と回答者との間での議論と責任者による結果の承認などが記録されるようになるだろう．

製造： ここは，設計において開発の対象となったコンポーネントの，詳細な論理設計とプログラム作成を行なう．ライフサイクルの中でも，最も分業の可能性の高いプロセスであろう．製造は，オブジェクト単位に実施され，それらの単位としての仕様の確認も行なう．仕様の確認は，部分的な実行も含めて，他の部品との組み合わせについても，基本的な動作が確認される．確認事項は，プログラムの分析によって自動的に抽出され，それにもとづいたチェックのための入力を，あらかじめ整備しておかなければならない．

試験： システムが大きいときは，サブシステム単位の試験を行なう．さらに全体のシステムを既存のパッケージソフトウェアも含めて組み上げて，所定の動作ができること，性能条件を満足することなどを確認する．試験の環境も，オープンなネットワーク上に接続され，種々の環境の設定と，遠隔からの制御を可能にしておく．大規模な試験は，夜間に自動チェックされ，人間はその結果を確認する．問題があれば，その担当者へ電子メール等によって，必要な情報とともに管理帳票を自動的に送付する．同時に，管理簿としてのリポジトリには，試験の項目ごとの結果と管理帳票の番号や分析の担当者などが，記録される．このデータをもとにした分析ツールを用いると，試験全体の状況も，ネットワーク上で簡便に参照可能であり，トータルのプロジェクトの進行状況を瞬時にチェックできる．

提供： このようにして試験が済んだソフトウェアは，これもまた，ネットワークを通じて，あるいは媒体に搭載して，別途整備されたマニュアルと共に提供される．ソフトウェア製品の提供には，ほとんど物流が伴わないですむで，他の産業に比較してきわめて有利な条件である．

保守： 運用を始めたシステムは，ネットワークを通じて操作/監視され，問題が生じると，これもネットワークを

通じて情報を収集し，分析を行なう．その手順は，試験の結果の不良の修正と同様な手順となろう．ソフトウェアの改版も，ネットワークを通じて実施される．これらの状況も，構成管理を行なうリポジトリに記録され，利用者と運用サイトに応じた稼動中のソフトウェアの版管理もなされよう．

開発管理とリポジトリ： 線表管理，品質管理，原価管理，資源管理，要員管理などのプロジェクト管理も，ネットワーク上の各種のツールを活用して実現される．これまでに述べたように，リポジトリを中心とした各種のデータにもとづく分析ツールが整備され，プロジェクトの開発管理も，いながらにして可能となろう．究極のソフトウェアCALSにおいては，各サブシステム単位あるいはもっと細かい単位と開発のプロセスの単位の組み合わせで，企業や国境を越えた組織から構成されるプロジェクトとなるかもしれない．この場合も，プロジェクトの中心はリポジトリである．

すべての情報は，ネットワークを通じて交換/更新され，必要な情報については，その更新の記録が自動的に保存される．もちろん，セキュリティ制御については，種々のアクセス制御が必須となる．したがって，このリポジトリについては，意味のある単位についてのアクセス権の管理ができなくてはならない．また，リポジトリ自体がネットワーク上に分散配置されていることも，当然考えなければならないだろう．権利の関係で，ある種の情報へのアクセスには対価が必要となることもあるかもしれない．

たとえば，そのプロジェクトで利用するパッケージがネットワーク上のサーバに置かれていて，その使用料金が利用形態によってさまざまであることが考えられる．試験の間は無償かそれに近い低料金，客先での運用が始まるとその利用規模によって値段が違うなどが考えられるし，その間の契約の交渉もネットワーク上で行なわれる．また，そのパッケージの利用法や試験ツールなども周辺のサーバにあり，それらの利用にもいろいろな条件が考えられる．それらのサーバへのアクセスでは，当然暗号化された情報を解く鍵の管理などが，今後の必須技術となろう．

6. おわりに

世の中が大きく変わる時代は，その変化に乗じて難問を一気に解決するチャンスだと捉えることができる．CALSは，衰退しかけている日本のソフトウェア産業の活性化の１つの可能性を示している．ここに示した想いには，現在のソフトウェア産業界にとっては，ある意味で迷惑千万なことも含まれているかもしれない．しかし，要員派遣による日銭稼ぎとしてのソフトウェア産業は，もうこの国では成立しないと考えたほうがまともであり，そのような部分をアジアの新興パワーにゆだねることも，きわめて現実的な事象として捉えなければならない．

顧客のいる所でなければできないこと，それは上流のBPRとこれを支援するビジネス情報システムの提案である．また，最後の維持管理におけるきめの細かいサポート，いわゆる営業によるフォローは，何といっても大半が直接のコミュニケーションに依存しよう．それでも，ソフトウェアというものが標準的なインフラ上でのパッケージ主体となった時には，ネットワークによるきわめて無機質な関係による取引きが大半を占めるようになるのかもしれない．また，インテグレーションの事例の情報などをあつめたサーバやその事例のカスタマイズなどが，ネットワーク上の新規ビジネスとして考えられる．つまり，日本でもコンサルタントのビジネス規模が拡大するのだろう．

ソフトウェアCALSの技術的な実現性は，現時点でもほとんど組み合わせ技術によって完成するように思われる．したがって，我々の課題はまさに，契約や交渉のレベルにあるようだ．その第一歩は情報交換形式の標準化にあろう．まずこの課題を整理して，一歩ずつ業界の情報流通の壁をオープン化していくことが今後の我々の急務である．電子取引きなど，制度の絡む課題については先行する他の産業の事例に学ぶことにしたい．あるいは，ソフトウェア産業として現在利用できるカードによる決済などの仕組みの下で，当面の解決策を試行してみるのもよい．

最後に，もっとも困難と思われる本質的課題を指摘しておきたい．それは企業間の垣根の撤廃，すなわちオープン化と国境をも越えた企業間の共同作業グループの構成にある．本文ではほとんど触れずにきたが日本の他の産業構造と同様に，これまでグループ内への囲いこみによる閉鎖社会の効率化を追求してきたソフトウェア産業においても，この課題を直視して取り組む姿勢が求められる．これまでの企業運営の価値観の変革を求めるに等しいこの課題の解決の推進力は，受注側の企業からというより，インテグレータを含めた発注側，利用者の意識改革からしか出てこないであろう．そして，この課題の枝別れの先には，汎用パッケージの活用など，これまで指摘されてきた日本の大規模システムにおける技術的課題などの解決も見えてくる．

[参考文献１] Hironobu Nagano: "Creating An Exemplary Organization", American programmer Vol.7 No.1, January 1994.

MaiDvoClu モデルによる ソフトウエア会社のリストラクチャリング

― システムアーキテクト集団の創出に向けて ―

木ノ下 勝郎
(オーロラシステム設計事務所)

0. はじめに

万物は流転し，諸行無常，流れる水の泡沫 (ウタカタ) は，かつ消えかつ結びとどまるためしなし．朝に紅顔の美少年，夕べには白髪の老人となる．世の中の会社も又，かつ起こりかつ倒れる．敵を倒し市場を制覇したその武器が，しばらくの栄華の後には逆に自らの弱みとなる．ひとつの会社の隆盛も 20 年，30 年は続かない．

プログラマー/SE を大量に動員し得たソフトハウス企業群が，華やかに店頭公開を果たし，創業者たちはキャピタルゲインを懐にした．

それを見たあまたのソフトハウスが，大量採用に奔走し，ベンチャーキャピタルを導入した．「ジョウジョウ，ジョウジョウ」の声が社内にこだました．店頭公開時の株式値上がりを夢見た経営幹部連中は，借金までして自社株購入に精出した．そして数年後とりあえずその泡沫は消えた．

バブル時期の新興ソフトハウスは，ヒトが多くいることが強みであった．しかし舞台は暗転し，ヒトが多くいることが重荷になった．「いかにしてメシをくわせるか」が，経営者の頭に四六時中宿る．

オープンシステム，ダウンサイジングのもとで顧客の要求/姿勢も変化した．マーケットの構造が変化しつつある．昔のニーズに適応した我が体制を，これからのニーズにどう適合させるか．需要に合わせなければ滅びる．

これまでの利益の源泉が，舞台が変われば，足枷となる．たとえば，SE 1 人と数人のプログラマーのセット販売商法に，今後も需要は存続するのか？

この小論は，「システム設計を独立した業」として，そのマーケットを創出させる立場からの「ソフトウエア会社リストラクチャリング」への提案である．

要約：システム設計部門と物理実装部門の分離

今後のソフトウエア業界構造は，次のような仕事を情報システム市場において供給する部門/会社で構成されることになり，それぞれの部門/会社に即した社内組織の再編成を提案する．

1. 情報システム関連のサービスを提供する仕事
   - (1) SD 部門/会社: 設計監理契約
   - (2) SI 部門/会社: SI 契約
   - (3) SO 部門/会社: 運用保守契約
2. プロダクトを販売する仕事
   - (1) 通信関連機器ベンダ: 売買契約
   - (2) CPU & 周辺機器ベンダ: 売買契約
   - (3) 基本ソフトベンダ: 売買契約
   - (4) ミドルウエアベンダ: 売買契約
   - (5) 業務パッケージベンダ: 売買契約
3. ソフト部品作成を受託開発する仕事
   - (1) ソフトウエアハウス: 請負契約
4. 技術要員を派遣する仕事
   - (1) 人材派遣会社: 派遣契約
   - (2) 専門サービス提供会社: 委託契約

1. マーケティングの視点

会社活動の基本は，市場 (マーケット) において「仕入れて売る」ことである．ソフトウエア会社は，「ソフトウエア開発」の需要に対して「ソフトウエア技術者」を仕入れた．バブル時期に興隆したソフトウエア会社は，構造不況業種における人員リストラの受け皿にもなった．

ソフトウエア開発における「属人性の排除」と「標準化して誰でもできること」と「工業生産方式：シグマ計画」を「ソフトウエア技術」のテーマとした．その結果，「誰でもできる＝若手でも中途採用の素人でも人月何十万円で売れる」というバブル時期のマーケットが成立した．

このマーケットは，「誰でもできるように標準化された仕

事の仕方」「抱えている要員を遊ばせないためのソフトウエア開発手法」「少数精鋭ではなく若手でも素人でも金をかせげるプロジェクト管理方法論」を結果的に許した.

そこで問題は,要員をたくさん抱えているソフトウエア会社の経営者が,このマーケットが存続すると思うか,それともマーケット=客のニーズが変化したと認識するか,である.

つまり「供給側の現状体制の雇用維持」の論理に基づき営業活動をおこない「何とか生き続けよう」とするか,それとも「現状の人員体制をリストラ」して「何とかして生まれ変わろう」とするか,の経営判断である.

換言すれば,バブル時期の巨大システム開発プロジェクトは,汎用機メーカが大型ハード機器を大量に売り込む戦略が主導したのであったが,そこでのアプリケーションソフト開発技術を,どう評価するか,という判断でもある.

さらにこの判断は,日米ソフトウエアギャップの無惨な状況をどのように認識するかという,ソフトウエア会社経営者の国際的な視座を問うことにもなる.

筆者は,バブル時期のアプリケーションソフトウエア開発技術はバブルであったと判断する.

ソフトウエア開発を製造業になぞらえること,素人でも誰でもできる手法を求めたこと,属人性を排除したがったこと,コーディング作業を「製造」と称したこと,手直し/手戻り/推敲を限りになく嫌ったこと,ドキュメントをていねいにたくさん書くことと「設計」を同一視したこと,等々の「80年代イデオロギー」は,実はソフトウエア技術の空洞化しかもたらさなかった.

この判断は,「ソフト会社の規模と技術力は反比例する」ことを傍証するいくつかの「少数精鋭優良企業」が,バブル崩壊後も元気に儲かっていることにより証明できる.

2. これまでと違って何が変化しつつあるか?

(1) ベンダのいいなりになることをユーザが拒否する.

ハードベンダ主導の下で機器販売と設計と実装が密接不可分であったが,これから数年は,オープンシステム環境における要素技術 (コンポーネント) ベンダー間の水平分業時代となる.その先は分からないが,ネットワーク業者の下での一元体制になるかも知れない.

- 何でも自分の会社でやることがメリットだったが,大企業といえども何でも自前ではやれなくなった.
- 大会社が必ずしも総合力を発揮できるわけではない.官僚的縄張り意識や事業部個別採算制度の故に,技術変化への対応が鈍くなる
- 「基本設計=1円」の入札事件が発生したが,機器販売では利益がでない.
- 人月単価契約に基づくコスト見積を逆手に取った2つの経営戦略が存在した.数で勝負,人海戦術/労働集約的に数を集めて店頭公開 :今後は苦しい質で勝負,少数精鋭で一括請負を受託した小規模優良会社:今後も有望
- メーカ主導をユーザーが嫌がる.ユーザー主導実現のための「ユーザー代行/支援: SD会社」業界を創出するチャンスである.
- 従来の情報システム部門は,汎用機ベンダーと密接に付き合ってきたが,今後はSD会社との協力関係を強化すべきであろう.

(2) SIはリスクが大きすぎる.

- 大手SI業者は,いくつかのSIプロジェクトの失敗に懲りて,フェーズごとの見積提案を志向するが,細切れ見積ではSIにあらず.
- 一括請負のリスクは,主として「要件定義フェーズ」にある.そのリスクを回避するために「業務モデル/論理設計」と「物理設計/実装」を分離する必然性がある.
- 実装上のリスクは,マルチベンダ環境下における性能設計,つまりコストと性能比較の困難である.そのリスクを回避するためには,ベンチマークテスト等のためのオープンシステムテスト環境を保有することであるが,これは資金力のある大手会社だけが可能となる.
- 業務モデルと論理設計のためには,プラットフォームから独立したソフトウエア技術の自立性確保が必要条件となる.科学と工学の関係に似せて,前者が科学で後者が工学(エンジニアリング)に対応するイメージとなる.科学とは,企業の業務活動を対象にして「認識論的アプローチ」を行い,その企業の情報構造(データ構造+意味構造)を記述する,ことを意味する.これは,資金力とか企業規模というよりも知恵力の領分である.
- 「SD会社」:経営コンサルタント,システムアーキテクト,設計事務所.その役割は,業務モデル/論理設計,利用者のためのプロジェクト管理.
- 「SI会社」:資金担保情報商社,情報エンジニアリング会社,ゼネコン.その役割は,物理設計/実装/テスト,実装側のプロジェクト管理.
- ただし,小規模なSIは従来型の小規模優良会社が存続するだろう.

(3) 業務パッケージが拡大しつつある.

- 広範なビジネスルールの蓄積ができつつある．ベストプラクティス部品群．
- カスタマイズ用 API が洗練されつつある．オブジェクト連携の標準化．
- 人海戦術/大量動員のプログラム開発需要が激減する
- 派遣型ソフト会社が没落し，ユーザ企業に再就職したプログラマ/SE 社会全体の情報リテラシイの底上げに貢献する．
- パッケージに業務を合わせたほうが，結果的に得する企業も出てくる．
- 競争優位を獲得するための戦略的業務と定型ルーティン業務との識別が重要となる．定型業務はなるべくパッケージを使う方向になる．

(4) 開発環境，ツールが充実しつつある．

- 画面や帳票の設計支援およびコード生成ツールの進歩が著しい．
- 大量の帳票/画面プログラム作成が，これまでの若手 SE およびプログラマの仕事であり，それが人海戦術型ソフト会社の米櫃であった．かし，これらのプログラムはツールによりユーザ自身が作成できるようになる．EUC つまり DIY(Do It Yourself) は，すべからく必然である．電話交換オペレータ，改札口での切符切り，お抱え運転手，半導体検査女工など歴史的事例には事欠かない．
- 今後はツールパッケージベンダが隆盛する．そこに，本物の専門プログラマたちの職場となる．日米ソフトギャップの克服の可能性がここにある．
- しかし，日本における大学教育制度および有名大会社指向の中にあっては，真に先端的なプログラム開発力を有するかどうか疑問もある．インドや中国の人材との連携が鍵になるかも知れない．

(5) ユーザの拡大に伴い運用保守の需要が増える．

- 従来の派遣プログラマ/SE は，運用保守アシスタントになる．
- SO 会社は，設備産業＋労働集約産業となり，規模のメリットにより情報処理業界の骨格を形成する．
- 高齢化社会における雇用吸収の場となり，さまざまな運用保守サービス業が発生する可能性が高い．
- EUC が増えれば，支援業務への需要も増加するであろう．
- 不景気の中にあって，「何とか経営を維持しているソフト会社を支えている柱は，オペレーション代行部門である」というケースが多い．

3. なぜシステム設計と実装の分離が必要か?

(1) ベンダ主導の場合は全部まかせればよかった．

ユーザは，ベンダから質問されたことだけに答えた．そして総合テストの段階で文句をいっても，時すでに遅い．結果的に，ユーザはベンダのいいなりにならざるをえなかった．そこで，ベンダ主導からユーザ主導へ転換するためには，次のことが必要となる．

- ユーザ側が主体的に自分の業務要件をベンダに提示すること
- ベンダ選定/プロダクト選定基準を明確にすること
- 物理設計/実装プロジェクトの進捗と品質を管理すること

(2) システム設計＝業務モデリングと DVO 論理構造設計．

これは，製造業や建築業における「設計図面」に相当する．

- システムの実装とは，「要素プロダクトの統合と検証」のことである．「コーディング＝製造」ではない．情報システムの「製造」の本質は，ユーザによる品質検証工程である．
- ユーザ主導のためには，ユーザ (又は，その代理人＝SD 会社) がプロジェクト工程全体に継続的に参加し得る方法論が必要となる．段階的開発とかスパイラルアップということになるが，段階的に要素を統合していくための「統合化設計」の原理が必要となる．それは，システム構造 (アーキテクチャ) の開発工程への展開である

  → (Mai/Dvo/Clu/*AMP の要諦)
- 従来のウォータフォール型プロジェクトでは，ユーザは「最初の要件定義と，最後の総合テスト」フェーズにしか参画できなかった．

(3) システム利用者とベンダの立場は違う．

- ベンダに全部「おまかせ」は，高級寿司屋と同じく値段は高くなる．
- 建築業のゼネコン汚職事件以降，企画設計と建築が分離する方向にあり，そのほうが数割方安くなるらしい．
- ユーザは，一定資源 (期間，コスト) 内で「最大品質」を求めるが，ベンダは，一定資源内で「最大利益」を求める．ここに「品質定義」に関する矛盾が存在する．
- 要求品質を定義する作業と品質を実現する作業も，両方とも SI ベンダに丸投げするのはおかしい．発注者責任と受注者責任をはっきり定義できる方法論が必要となる．
- それは，システム設計/論理設計と物理設計/実装の

分離である．システム設計/論理設計と物理設計の違いは，次のような性格をもつ．システムを利用する立場での設計：業務モデル/機能/情報構造の論理設計．

4. システム設計と物理設計/実装の分離は可能か?

(1) 理論的な観点からは可能である．

- ある会社の業務は，パソコンLANでもUnixでもオフコンでも汎用機でもIBMでも富士通でも現実に実装可能である．この事実は，特定の実装物理環境から独立した「業務要件と論理構造」が存在することを示唆している．
- しかし，これまでは「業務要件定義書」「論理構造設計書」を記述する一般的な技術体系が存在しなかった．その主たる理由は，「トップダウン機能分解主義」と「ハードベンダが決定してからのアプリケーション設計」および「業務分析における科学的アプローチの未熟さ」などである．
- 今後は，オブジェクト指向の流行に見られるごとく「モデリング：対象構造の抽象化」の概念は一般化してくる．

(2) ベンダの経営的な観点からは，設計と実装の分離に抵抗する会社もある．

- ハード機器を売るためには，上流工程も取り込む必要がある．しかしハード機器ベンダの主導権は弱まる．ハード機器ベンダ自身「ソリューションビジネス，サービスの有償化」に必死である．
- 設計業務だけでは，若手要員の仕事がない．戦力にならない若手は，付加価値の高い年寄りが喰わせなければならない．若手が年寄りを養うのではなく，年寄りが若手のめんどうを見る時代．45過ぎたかっての腕力のある元SEたちが現場に復帰して，30代のバブル時期の促成SEを鍛えるべきであろう．
- 少数精鋭の優良会社の利益の源泉は，一括請負である．一括請負でやれる会社は，今後も存続できる．

5. これまでなぜ設計と実装が分離されなかったか?

(1) ハード機器ベンダが業界を主導したので，ソフトはハードに従属した．ベンダが決定した後，プロジェクトが発足した．

(2) 業務内容が定型的で単純であったので，実装技術だけを外部に依存した．業務内容を知っている会社プロパーの情報システム部員が参加した．

(3) 大規模プロジェクトでは，質よりもプログラマの動員力を重視した．第3次オンラインプロジェクトの実態は，人海戦術であった．

(4) 機能分解トップダウン思考により，設計と実装の切れ目が判然としなかった．要件定義作業が，プログラム仕様書作成まで続いた．

(5) コード化されたプロダクトの「組み合わせ＝実装」というよりも「プログラムコーディング＝製造」が，主要な仕事であった．業務パッケージのカスタマイズではなく，特注品の手作り．

(6) 設計ツールが不在で，コード表現しか信用できなかった．論理設計と物理設計に分けることは，無駄であった．

(7)「概念/論理/物理」の区別が一般的ではなく，「概要/詳細」「外部/内部」が一般的であった．属人性を嫌い，標準化/マニュアル化すれば，誰でも「設計」できると錯覚した．

6. 今後はどうなるか?

(1) ハード機器ベンダーが業界を主導したので，ソフトはハードに従属した．実装環境から独立した仮想ハイパー言語によるソフトウエア設計書の記述が可能になる．業務システムの表現言語，つまり情報システム論理図面規約が必要となる．これが上流CASEツールの発展方向であろう (業務モデル)．

(2) 業務内容が定型的で単純であったので，実装技術だけを外部に依存した．システムの対象業務が拡大し，全社的な情報の共有化と統合化のためには専門技術が必要となる．企業活動のダイナミズムを把握して，その構造を記述する専門性の確立のためには，従来の経営コンサルタントや上級SEの職種規定を越えた専門集団が必然となる (システムアーキテクト)．

(3) 大規模プロジェクトでは，質よりもプログラマーの動員力を重視した．コーディング力よりも市販プロダクトの組立能力 (エンジニアリング力) 重視に変化する．要素技術の開発力と統合組立技術は，その技術基盤が異なる．統合組立技術は，業務システム構造からコンピュータシステム構造への変換技術に相当する．汎用機中心の集中制御型システム設計に比較して，オープンシステム環境における分散型協調システムの設計は，はるかに難しい．

(4) 機能分解トップダウン思考により，設計と実装の切れ目が判然としなかった．要件のCLU化とDVO構造により，判然とした切れ目が可能．システム設計/論理設計における「機能」は物理設計/実装概念において継承/存されるが，形態や構造は，変換される．従来のウォータフォール型では，業務システムにおける「形態と構造」が明示的な設計対象になっていなかった．

C 要件 → 機能設計
LU 要件 → 構造設計

にほぼ対応する.

(5) コード化されたプロダクトの「組み合わせ=実装」というよりも「プログラムコーディング=製造」が, 主要な仕事であった. 強力な業務パッケージが普及する. 従来の画面プログラムや帳票プログラムを COBOL でコーディングする仕事の約 8 割は消滅する.

(6) 設計ツールが不在で, コード表現しか信用できなかった. プログラミング作業はツールによる自動生成と業務パッケージのカスタマイズになる. システム設計/論理設計を支援するツールが発展する. 物理設計/実装は, つぎの 3 つの組み合わせとなる.

レディーメイド:
パッケージに業務を合わせる
イージーメイド:
パッケージをカスタマイズする
オーダーメイド:
自社独自のカスタムソフトを開発する

(7)「概念/論理/物理」の区別が一般的ではなく,「概要/詳細」「外部/内部」が一般的であった. 開発工程がますますツール化/自動化され, 概念/論理設計の比重が高まる. システム開発は, 建物や機器の工場生産に模擬するよりも, 属人性を重視した文筆業と印刷/出版業に似てくる.

7. 設計と実装を分離するための技術的課題

7.1. 要件定義

当面は, 上流 CASE ツールと称されるものを使って OCD, FDD, DFD, ERD, STD ダイアグラムと定義書を業務モデルとして作成する. それぞれのダイアグラムには, 欠陥もある. ツールの機能も充分ではない. 業務上の意味表現規約もまだまだ十分ではない.

OCD: Organization Context Diagram
組織情報関連図
FDD: Functional Decomposition Diagram
機能構成図
DFD: Data Flow Diagram
データフロー
ERD: Entity Relationship Diagram
実体関連図
STD: State Transition Diagram
状態遷移図

将来は,「業務の仕組み」を記述するための仮想ハイパー言語 (マルチメディアダイアグラム) を開発する.

業務の世界を,「誰か」が「何かに」対して何か「する」ことで構成されると見れば, 歴史的には次のような過程を経てきた.

70 年代まで: 手続き規則.「する」に着目してプロセス定義を重視した. アルゴリズム, 処理手続き規則, 構造化手法.
現在まで: データ項目.「何」に着目してデータ定義を重視した. データ中心設計 (DOA) とオブジェクト指向.
これから: 情報 (=データ+意味) →行動指針.「誰が」に着目して人間の仕事の仕方を重視する. 情報処理支援クライアントシステム. 情報とは「データ」に対する「意味付与」であり, 価値判断と意志決定のトリガーである.

従来の事務処理システムは, 組織で決定されたデータ処理ルールをコンピュータで代行するものであった. 個人個人の顔が見えないシステムであった. 個人裁量の業務ルールではなく誰がやっても同じ処理をするルールを「要件」と称した. 今後は, この要件は「サーバーシステム」として実装される.

「誰が要件を決定するか」「誰の何のための要件か」ということに関する方法概念が「PDC」である.

P: 個人 (Person, People)
D: 部門 (Division, Department)
C: 会社 (Company, Corporate)

要件の定義属性からの方法概念として次の 3 つを区別する.

目的要件 → 情報システムの評価基準
情報要件 → 業務モデルとして記述する情報構造
制約要件 → 期間, 金額, 体制等に関する資源

情報要件の定義可能性と実装上の構成要素分割および段階的開発 (統合化設計) の方法概念が「CLU」である.

C: 正しさ (Correctness)
L: 見やすさ (Layout)
L: 使いやすさ (Usage)

今後のクライアントシステムの特徴は,「個人の顔が見える」システムである. そこでは人間の行う「情報処理機構」「人間同志のコミュニケーション機構」に関する基礎論なくしては, クライアントシステム要件定義の糸口が見つからないと思う.

これまでの汎用機中心のシステムは, 自動化, 効率化, 集

中化を特徴とした結果，コンピュータシステムは人間にとっては巨大なブラックボックスとなった．そこには生身の人間の姿はない．人間にとってはひとつの「疎外態」として君臨してきた．

組織における人間の仕事のまっとうな復権を支援することが，クライアントシステムの本質であると思いたい．GUI で快適なインタフェースを提供することは，必要ではあっても充分ではないのである．

業務要件の決定責任は，業務の世界にある．システム技術者は，その世界とどのようにかかわり合うか．システム技術者の専門性は，業務要件に対する「視点」「意識」にある．その視点とは，システム構造モデルつまりアーキテクチャである．アーキテクチャとは，事務処理系情報システムを構成する要素とその関係に関する体系である．

実現構成要素の枠組み (フレーム，テンプレート) を用意して，業務世界を意識すること，意識により抽象された現象を記述すること，これがシステム技術者としての専門性である．お客から聞いたあれこれを単に分かりやすいよう絵にしたり文章にしたりすることには，専門性は不要である．従来の「業務 SE のシステム技術者の専門性とは，いったい何だったのだろうという思いがうずく．

7.2. DVO 構造設計

当面は，データ中心設計による「データ定義」を主体にした「DVO/*AMP 構造」をシステム構造とする．その構造の特徴は，次の 3 階層協調構造である．

(1) クライアントシステム：
　　物理ビュー制御処理
(2) 業務サーバーシステム：
　　論理ビュー (トランザクション) 制御処理
(3) データベースシステム：
　　内部データ一貫性制御処理

情報要件がこの DVO/*AMP 構成要素に対応つけられて，物理的なコード表現として実装される．

オープンシステム，マルチベンダー，コンポーネントウエアに関するまず最初の設計概念が，「AMP」である．

A：業務層
M：ミドル層
P：プラットフォーム

「入力→処理 (プロセス) →出力」というプロセス中心システムモデルから「入力→データベース→出力」というデータ中心システムモデルのキーワードが［DVO」である．

D： Data Base(内部データ)：
　　データの内部格納構造
V： View(外部データ)：
　　システム環境とのインタフェース
O： Operation(操作)：
　　外部データと内部データの交換移送機構

分散構造設計の方法概念が前述した［PDC」である．

システム設計の本当の難しさは，性能設計にある．性能設計は，その一つ下のレイヤーに位置するソフトウエアの性能とハードウエア性能に依存するからである．信頼性設計 (障害対策) と効率設計 (時間および資源) において，これまでの経験上ほんとに論理的に難しいソフトウエア設計課題は，次のようにいくつもはない．

(1) 同一資源に対する排他制御
(2) 同期通信におけるタイムアウト制御
(3) 最近の分散 DB の同時更新

データ圧縮とかスケジューリングとかメモリー管理とかそれぞれの難しさはいろいろあるが，実時間系の複雑な制御システムの難しさも原理的にはこの 3 つに集約される．

事務処理系システムの構造設計の要諦は，結局は「性能設計」である．データ量と処理性能との関係は，当然ノンリニアであり，ある種のカオス現象ともみなせる．

つまり私たちは常に所与のレイヤーの上で自分たちの責任を果たしてきたわけであるが，よって立つそのレイヤーが他律的に変化してきた．

「バッチ処理」： 紙テープ，パンチカードシステム，磁気テープ，ディスク，ジョブ制御言語，マルチジョブ，ジョブスケジューリング管理などの技術が次次にでてきた．

「即時処理」： オンラインリアルタイム処理の登場でシステム技術に質的な変化が訪れた．オペレーティングシステムも通信制御システムも OLTP モニターも発達し，集中制御と信頼性と自動化と効率を追求した汎用機システムは肥大化の一途をたどった．金融機関システムにおける全国ネットのトランザクション制御の仕組みは，実に壮大なシステムである．

「OA 処理」： 決定論的データ処理から取り残されたこまごました身の回りの仕事のためにパソコンとワークステーションが登場した．個人の顔が見えるシステムである．決定論的手続きではなく個人秘書としてお手伝いしてくれるクライアントシステムが要求されている．グループウエア，ワークフロー制御の仕組みの時代になった．機械論的システム設計技法から有機的システム設計技法の転

換点であると思う．「システム論」から「情報論」へ力点をシフトさせねばならない．

このように自分達の技術が，もし「技術というものがあったとして」，それが成熟する充分な期間もなく翻弄されてきたと思う．しんどいことです．

現行システムを再構築するここ数年の間は，DVO構造で対応可能である．しかし，秒数百メガビットのB-ISDNが広く利用される次の次の時代のアーキテクチャがどうなるかは，少なくとも自分の当面の課題ではない．

先端技術の先物市場に玉石混交をあさり，予言者風にいささかの評論をなすのは自分の任ではない．

7.3. 進捗品質管理

当面は，CLU別段階的統合化テストのスケジュール管理を重視する．納期と工数の制約要件の範囲内で，いかに品質を最大にしていくようにプロジェクトを管理するか．その方法概念がCLUである．これは「必要最小限の設計ドキュメント」とは何かに関する答も準備する．

従来のウォータフォールモデルでは「無駄なドキュメント」を書きすぎた．最近のパソコンボーヤにプログラムを作らせれば，全くドキュメントを書かない．プロトタイプだけでシステムが実現するはずがない．

C要件はきちんとドキュメントする．LU要件は動かしながら評価する．これが「ウォータフォールモデル」と「スパイラルアップ」の止揚である．

将来は，ユーザー検証用仮想ハイパー言語の対話型インタプリター(3次元アニメータ：バーチャルリアティ駆動エンジン)を開発し，業務モデルそのものの動きを評価し，実装環境に自動変換することになる．

7.4. 方法論とツール

方法論の洗練は，実践の場で鍛えられるしかない．ツールの設計/開発には，まず「資金」がなければならない．資金をどうやって調達するか? 日本のソフトウエア産業にあって，ツール開発の担い手のありように何か根本的な問題があるように思われる．「大企業信仰」「舶来信仰」「アカデミズ信仰」「完全主義」「共同体的平等主義」「実績重視伝統主義」....

参考文献

弊著；「システム再構築プロジェクトの計画と管理」(丸山学芸図書，3800円)

center, box; c.

*** 付 録 ***

真夏の夜の夢

「システムアーキテクト」集団 形成へのメッセージ

95/8/10

(株) オーロラシステム設計事務所
木ノ下勝郎

謹啓

ますますご健勝のこととお慶び申し上げます．かねてよりご指導ご援助を賜り厚く御礼申し上げます．

さて，わたしたち10名足らずの集団は，「システムアーキテクトを中心としたシステム設計事務所」を自称しておりますが，「システムアーキテクト」も「システム設計事務所」も，いまだ世の中に認知された「専門職」「業種，業態」ではありません．

いっぽう上場企業クラスの多くの会社では，不景気の影響もあって，償却済みのシステムを継続運用しております．そのシステムは，一般に特定メーカの大型汎用機を中心にしたシステムで，最近のオープンシステム技術に比べて「高い，遅い，使いにくい」の非難にさらされております．経営者もユーザも情報システム部もそれぞれの立場で，「現行システムをいづれ何とかしなければならない」との思いが強くなっています．

つまり「現行システムの再構築需要」のエネルギーは，確実に蓄積されつつありますが，業界の受け皿構造との矛盾も増大しつつあると認識します．

再構築にあたって，発注側ユーザ企業は，まず最初にどこに相談されるのでしょうか?

- これまでのように，まずシンクタンクや経営コンサルタントに稟議書の作成支援をお願いするのでしょうか?
- あるいは従来通り，大型汎用機メーカの主導の下で，プロジェクトが仕切られるのを望むのでしょうか?
- はたまた，大手SI認定企業に「一括で頼むからよろしく頼む」と丸投げするのでしょうか?
- そうではなくて，さまざまなオープンシステム要素技術を，ユーザ自らインテグレーションするのでしょうか?

わたしたちは，まず最初に「システム設計事務所」に相談いただきたいと願うものです．その武器は，システム再構築プロジェクトをすすめるにあたっての方法論 (MaiD-voClu；マイデボクル) であります．

その方法論にもとづき，今後のシステム開発においては，「システムアーキテクトを中心にしたシステム設計事務所」が，「上流工程における要件定義とアーキテクチャ設計およびその後のユーザ側に立ったプロジェクト管理」を引き受けるべきではないかと考えます．

その趣旨背景は，ソフトウエア業界も他の経済社会環境と同様に「いま大きな構造的変化のただ中にある」との認識です．

ソフトウエア業界の過去を振り返れば，次のように常に新しい「業種，業態」が創出してきたことが分かります．

1960 年代以前：
　ハードメーカが誕生
1960 年代中期：
　共同利用計算センタが創立
1970 年代初頭：
　ソウトハウス/オフコンディーラが誕生
1970 年代中期：
　オペレータ派遣 FM 事業が創出
1970 年代後期：
　ソフトパッケージ販売/流通業が確立
1980 年代中期：
　プログラマ/SE 派遣会社が乱立
1980 年代後期：
　ソフトウエア出版事業が隆盛
1990 年代初頭：
　ゲームソフト/PC ソフトベンダが勃興

このような歴史的視点にたてば，1990 年代中期の現在は，バブル景気の崩壊と来るべきマルティメディア/高速通信時代の谷間に位置すると思われます．オープンシステムの名のもとで，プラットフォーム技術の標準化やツール市場構成は，いまだ流動的であります．こういう状況にあって，約 5,000 社，60 万人が従事しているといわれる「情報処理サービス業」の雇用構造も流動化しつつあります．大型汎用機中心の人海戦術型プロジェクトが激減したからです．

谷間と混乱の時期にこそ次の時代の新たな展望がうまれます．歴史の流れの大きな方向は，「ベンダ主導からユーザ主導へ」「生産者重視から消費者重視へ」「人間中心のクライアントシステムと組織中心のサーバシステムの協調処理へ」「対話型情報処理支援と自動運用型データ処理代行の協調処理へ」であろうと思われます．

情報システム技術を企業活動にどのようにして役立てていくか，というテーマに対して，真にユーザの立場に立った「業種，業態」を創出させる必然が存在するのではないか?

この思いが，「システムアーキテクト集団」「システム設計事務所」業界の創出を展望する基本的な動機であります．しかしながらわたしたち 1 社では，社会的認知を得るのはいかんともしがたく，多くの「システムアーキテクト」と「システム設計事務所」の出現を願うわけです．

そのために「あたしも，俺もシステムアーキテクトだ」と自称するシステムアーキテクトがあちこちに勝手に誕生することを期待するものであります．ご支援ご協力をお願い申し上げます．

以　上

制作の意志とオブジェクト指向技術

― 南山大学特別講義のためのノート ―

熊谷 章
(PFU 研究所)

概 要

コンピュータシステム，ソフトウェアシステムの研究開発は，ものごとの制作行為そのものである．この講義では，人間が行う制作行為そのものをテーマにして，それと自然界がさまざまなものを生み出すことと，どのような違いや類似があるか，といった基本的なことがらを考察してみたい．それは，科学，哲学，工学，思考といったわれわれの行為を対象とし，アーキテクチャを考えることにつながる．このような観点から，オブジェクト指向がどのような視点と技術を提供しているかを考察する．そして，オブジェクト指向が抱えている限界と問題点を指摘し，それらをどのようにして解決できるかという方向性を示してみたい．基本的な視点は次の3つである：

- 制作の意志
- オブジェクト指向の基礎とその動向
- Image Based System 開発のための
  オブジェクト指向の適用

1. 制作の意志

古来，人間は自分の手でものを作ってきた．ものを作るという行為はなんなのだろうか? このきわめて単純な視点から始めて，最後は表題である”制作の意志とオブジェクト指向技術”まで話を続けてみたい．この話を書いてみようと思い立ったきっかけは，次の感慨からである．それは，このコンピュータ産業という仕事に就いてから，20数年間私の心にまとわりついているもので，日本ではなぜ新しいコンピュータ技術が生まれ，発展しないのだろうか? という疑問である．テーマが少し大きすぎると思われるかも知れないが，この傾向は他の分野でもよく話題になる．

特に，外国人と会って，各国の特徴を話題にすれば，決まってこのテーマが現れる．「日本人は猿真似が好きで上手だから，発明ができるイギリス人とペアを組めば，世界中で適う相手はいない」という英国人．「設計をアメリカ人がやり，それを高品質で製造するのが日本だ」と考えている米国人．「日本人はお金はあるが，哲学や国際政治力がない」と思っている中国人．かれらは，日本人を一人前の人間として評価していないように，私には見える．6月2日付の朝日新聞のコラムには，戦後の日本人の精神年齢をマッカーサーは12歳と判断した，と書かれていた．コラムの筆者は，それでは1995年の日本人の精神年齢は何歳と世界の人びとが判断するであろうか? と締めくくっていた．

現代を代表する(?)思想家の一人である吉本隆明にいわせれば，日本古来の文化は，たとえれば掘立て小屋だそうだ．外来から来る嵐に出会えば，それまでの建造物は跡形もなく壊れてしまい，またそこにすぐ建てられる掘立て小屋を作り，それで間に合わせ，次の嵐が来ればたちどころにまた壊されてしまう．日本人の文化の歴史はこの繰り返しだという．哀しいかな，この指摘はかなりあたっているように思われる．だいたい，ものを考えたり，表現したりする文字そのもが，外来文字を使っているのだから仕方がないといえばそれまでだが，ここら辺には何か深い原因がありそうだ．

コンピュータ産業の掘立て小屋と嵐の関係を表1に示す：

この表では，基本的なハードウェアとソフトウェアだけをあげたが，実はこの現象はミドルウェアとか，付加価値の高いアプリケーションでも起きている．具体的にあげれば，Spreadsheet, database, CAD/CAM, Computer Languages, Window system, 清書プログラム，desktop publishing など，数え上げればきりがない．いいかえれば，日本には，嵐が来ても壊されないくらいの強固なものはただの一つも存在しなかったことになる．

表1 コンピュータ世界の堀立て小屋

掘立て小屋	嵐	説明
独自の汎用機	IBM Main Frame	1973 年頃
独自の OS，言語，アプリ	IBM OS360	コピーが流行
独自のミニコン	DEC PDP	米国流のマシンが出回る
独自の OS，言語，アプリ	RSTS, VMS	なかなかいい勝負
IBM 互換機	IBM Main Frame, Cluster	完全互換を目指す
独自のオフィスコンピュータ	IBM PC	IBM PC 市場に介入
独自の OS，言語，アプリ	CPM, MS-DOS, Basic	米国ソフトベンダー台頭
独自のパソコン	IBM PC, Macintosh	日本市場への攻勢
独自の OS，言語，アプリ	MS-DOS, Windows, Mac OS	Microsoft 独壇上
独自のワークステーション	SUN WS, HP WS	一時 SUN WS 一色
独自の OS，言語，アプリ	UNIX, Realtime UNIX	UNIX

この異常な現象の原因を，私は次のように考える．古来，われわれ日本人には「制作への意志」がなかった．「制作への意志」は，建築への意志，または ' Will to Construc' と表現される．なぜ，日本では制作の意志が生まれなかったか? といえば，日本社会がもっている下記の３つの特徴が効いていると思われる．

(1) 閉鎖社会
(2) 交通量が少ない
(3) 物事の制作よりも自己表出の方が好き

この日本の特徴と対局にあるのが，現代の米国と唐時代の中国である．その社会構造はほとんどネットワークであり，典型的なオープンシステム (open system) であり，それぞれノード間にはすさまじい量の交通が存在する．そして，そこでは，生産のための努力が常になされており，技術革新が起きている．これは，マルクスがいっていることだが，交通量が決め手なのである．交通量の多いところに生産技術と制作への意志が生まれる．

次に，自己表現というキーワードを明確にするために，言語と意味について考える．言語を道具とみる考え方と，自己表現そのものとみる考え方がある．前者は，われわれがいますでにもっている思想を使って定義されるもので，コミュニケーションや意味伝達のための道具として使用されるものを指す．その典型的な例は，科学的記述にみることができる．これに対して，後者は言葉を自己表出に用るものを指す．典型的な例は，詩や歌などにみられる．ここで，表現 (representation) と自己表出 (expression) の違いを明確に認識することが重要である．

この主題に関係すると思われる２つの思考方法を，次に紹介する．これは H.Bergson が唱えているもので，分析的方法 (Analytic method) と形而上学的方法 (Metaphysical method) との２つである．これらの比較表を次に示す．

表2 ２つの思考方法

形而上学的方法	分析的方法
無シンボル，無座標系	シンボル，無座標
直観	定量的測定
絶対的な理解	相対的な理解
系の中	系の外
当事者の視点	観察者の視点

分析的方法は，近代科学の分野において一般的に用いられているもので，その基礎とみなされている物理学に，その典型をみることができる．ある対象を認識しようとすれば，その系の外から，なんらかの測定用機器を用いて，その対象の特質や振舞いを測定する．このとき，観察者が持っている座標系と，対象を表すシンボルを使用する．そして，多くの測定結果から，対象およびそれを含む現象のルールを見つけだす．そのルールと現象との間の再現性が確認されれば，対象を理解したことになる．

この方法の特徴は，観察者が系の外にいることであり，それにともない対象の理解が相対的になることである．つまり，観察者は，それぞれが独自の座標系を持っているため，それにもとづく観測は，必然的に相対的なものになる．この方法の原理は，有名なデカルトの「方法序説」に書かれ

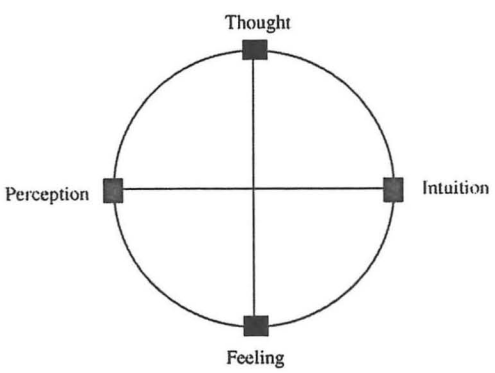
ている通りである．

一方，形而上学的方法は，Bergsonが，ものごとを理解する方法として提唱しているものである．簡単にいえば，この方法では，対象を理解しようとする人が，直観により対象を内部からビジュアルにみるということになる．具体的には，まず，対象に対して共感や親愛心を持つことから始まる．次に，意識をその対象の中へ投入する．そして，次第にその度合いを増してゆき，最後には，対象そのものになりきるのである．いったん，対象そのものになりきり，それを完全に理解したら，その意味を他人にわかるような形式で表現すればよい．

この方法の特徴は，当事者が系の中にいて，絶対的な理解が可能だということである．

次に，この２つの方法と人間の心の構造について考えてみたい．C.G.Jungがその著書「Man and His Symbols」で次のように述べている．

図１　Jungの図式



上図の各点は，それぞれ対極を持っている．たとえば，思考 (Thinking) に対しては感情 (Feeling) が対応しており，思考型の人間にとって最も未発達な部分が感情であることを示す．これらの４つの機能のタイプは，意識が経験の方向づけを行う傾向に対応している．感覚 (Perception) は，われわれに何かが存在指定るを告げ，思考は，それが何であるかを告げ，感情は，それが快感を与えるか否かを告げ，直観 (Intuition) はそれがどこから来てどこへ行くかを告げる．思考から感情の線は，価値判断をともなう合理的な機能であるが，感覚と直観を結ぶ水平線は，精神的な原因よりは身体的な原因に依存しているもので，自然発生的でかつ内的，外的な環境に依存する．

ここでの私の主張は，従来コンピュータを使用する世界では，思考･感情で表現される垂直線のみを対象としてきたが，これからは感覚･直観で表される水平線をも対象とすべきだということである．これによって，コンピュータの利用形態が，従来の論理的な表現 (representation) から，自己表出 (self-expression) へと変革できるものと考えられる．

次に，制作の意志 (Will to Construct) と力への意志 (Will to Power) について考察する．前者は創造型 (Creation View)，後者は進化型 (Evolution View) と見られる (表 3)．

表３　創造型の視点と進化型の視点

Creation View	Evolution View
Creation(建築)	Generation(生成)
Product(生産物)	Text(テキスト)
意味の形式化	Order の生成
Representation(表現)	Expression(自己表出)
独立した Order と構造	Chaotic かつ Genetic
Calculus(Newton)	Taoism(老子)
Will to Construct(Des Cartes)	Will to Power(Nietzsche)

プラトンは「饗宴」の中で，制作 (creation) の意味を，広義に，次のように定義している．非存在から存在への移行をすべて制作と呼ぶ．この視点，つまりすべての生成を建築とみるものの見方は，建築家のそれである．ここには，西洋の知と表現できる最も特徴的なものの見方，建築への意志がはっきり見られる．ここでは，超越論的な意味の外化とその再現が最も重要課題となる．具体例としては，ガリレイの解析幾何学やニュートンの微積分をあげることができる．また，建築への意志は，必然的に真理への意志につながる．つまり，世界は究極的に可知的であり，秩序を持つという信念が，そこにある．そして，ついには,'God as The Great Architect' という考えにたどりつく．ここまで来れば，かれらが何かを作るとき，自然に依存する必要がなくなり，自己完結した世界がもたらされる．

一方，Niezscheがいうところの権力への意志 (Will to Power) は，先に述べた建築への意志とはまったく異なり，われわれの日常生活を貫いているドライビングフォース (driving force) の源泉ともいえるものである．ここでは，超越論的な意味や構造を超出して自ら意味を産出することができなければならない．それが，Productに対するTextであり，Orderの生成の意味するところである．そして，われわれは，自然をある秩序にしたがってその詳細を記述し説明できるという大いなる誤解から抜け出し，われわれ自身が自然の中にいて，すべてのことがらが自己言及的になっていることに気づく必要がある．

われわれ日本人は，上で述べた建築への意志とは，縁遠い世界で生きてきた．そして，好みとして内的な世界を対

象とした自己表出をこよなく愛したと考えられる．したがって，自己表出の典型といえる芸術には，時代を通した日本人の注力があり，傑作をたくさん生み出している．一方，大量生産に代表される文明の利器としての生産物に対しては，あまり注力せず，すでにあるものの模倣で終始したといえないだろうか? そして，意外にも自己表出としての器用さが，大量生産の技術にもうまく適用できて，予想外の日本的工場社会ができあがってしまった．

しかし，建築への意志が稀薄またはほとんど存在しないために，自らの手で創造物を作り出す行為を，社会的に価値あるものとは，ついには見なさなかった．さらに，価値観として，外来物である金を最上位に置いたため，本来最も重要だと認識していた自己表出技術を支えていた美的価値観さえも，忘却のかなたに追いやってしまった感が強い．

ここで，コンピュータを自己表出の媒体として使用することを提案したい．少し乱暴かも知れないが，コンピュータを従来の言語と同じ役割を持ったものとしよう，という考えである．いわば,'A Computer as a Language' というもので，従来，コンピュータをわれわれは論理的な表現システムとして扱って来たが，これからは自己表出 (Self-expression) として使い，新たな世界を築くことを意図している．

このアプローチの重要な課題は次のものであると考える．

- 形而上学的な方法 (Metaphysical Method)
- 権力への意志
- 自己表出のための３つの基本要因
  - 下半身モヤモヤ (rhythm)
  - ミゾおちワクワク (melody and harmony)
  - 頭クラクラ (concept)

2. オブジェクト指向の基礎とその動向

2.1. 最近の動向

ここでは，コンピュータ･プログラミングを中心に，最近注目される動向について簡単に述べる．

(1) プログラミング言語

Smalltalk-80 を契機として一世を風靡したオブジェクト指向技術は，一時いくらかなりを潜めたが，ここ１～２年のあいだにふたたび注目され，新しい言語があらわれはじめてきた．それらを以下にあげる：

Smalltalk 系：
- Smalltalk-80,
- Objectworks,
- Smalltalk/V,
- Distributed Smalltalk,
- SmalltalkAgents

C 系：
- Objective C,
- C++

Lisp 系：
- CLOS,
- Euslisp

PPS(Parcplace Smalltalk) による主流は，Smalltalk-80, Objectworks である．後者には，開発ツールが含まれており，いまでは，ほとんどのマシン上で動作できるようになった．Smalltalk/V は，Digitalk 社が開発したもので，低価格でかつパソコン上で動作するシステムを狙っている．Distributed Smalltalk は，HP と IBM が，分散環境で動作するものとして開発したものであり，CORBA の仕様に準拠している．SmalltalkAgents は，QKS(QuasarKnowlegde Systems, Inc) が開発したもので，今後予想される分散システムに対応した統合型のオブジェクト指向ダイナミック言語開発環境である．その特徴は，仮想マシンがSmalltalk のみでなく，C, SQL, TeX, Prolog, Dylan などの言語をサポートしているらしい．

Objective C は，Next が Nextstep のシステム記述言語として使用している C 言語版のオブジェクト指向言語である．これとほぼ同じなのが AT&T から出た C++ である．

CLOS, Euslisp は，いずれも，Common lisp にオブジェクト指向の機能を追加したものである．前者は ANSI によって制定されたもので，後者は電子技術総合研究所で考案され作成されたものである．

(2) オブジェクト･マネージメント･システム

オブジェクト･マネージメント･システムは，さまざまな形態の情報を格納するデータベースである．最近ようやく世界的規模での標準化が行われている．この技術とオブジェクト指向とは，作成したオブジェクトを格納しておくという点で強い関係がある．現在では次の３つの系統がある：

CORBA(Common Obeject Rerquest Broker):
- OMG(Object Management Group)

PCTE(Portable Common Technology Environment):
- ISO, ECMA, NIST

OODB:
- 各メーカが開発した OODB
- (例，Gemstone, O2, ODB II, Objectstore, etc)

(3) 並列分散システム

オブジェクト指向システムから並列分散への拡張は，比較的すんなりと考えられる．次のものがある：

ABCL(An Object-Based Concurrent Language):
- 米沢明憲 (東京大学)

Act3:
Actor 理論に基づいた言語，C.Hewitt(MIT)

Actor 理論は，Open system つまり，解放され連続的に進化するシステムの研究開発のために考え出されれた．その特徴は，並列計算のパラダイムであり，関数プログラミングとオブジェクト指向プログラミングを統合している点である．ABCL は，オブジェクトが concurrent にかつ並列に動作できるようにした言語で，異なったコンピュータ上で動作する．

(4) ComponentWare

昨年春ごろから，オブジェクト指向技術はすでに終焉し，ConponentWare の時代が始まったと，一部で主張されている．簡単にいえば，Class Hierarchy のないコンポーネントを基本として，それらを組み合わせてプログラムを作ろうというものである．

次のようなプロダクトがすでに登場してきている：

PARTS (Digitalk)
Host 言語: Smalltalk/V
スクリプト: PARTStalk
HyperComposer (PFU)
Host 言語: Smalltalk-80 or C++
スクリプト: なし
VisualAge (IBM)
Host 言語: Smalltalk-IBM
スクリプト: Smalltalk-IBM
NextStep (Next)
Host 言語: Objective-C
スクリプト: Objective-C
IPAD (富士通，日立等)
Host 言語: C++
スクリプト: Modifier(C-like)
Visual Basic (MicroSoft)
Host 言語: Basic
スクリプト： Basic

これらの動きの中で注目されるのは，アメリカの企業が，これからのプログラミング言語の基底言語として Smalltalk を採用し，その上に新しい言語を構築していることである．

(5) Agent Orientation

日本の FRIEND21 プロジェクト (通産省，1988-1994) に刺激されてか，世界中で Agent がブームになっている．また，General Magic 社の Telescript と Magic Cap も，このブームに拍車をかけた．FRIEND21 によれば，次のカテゴリーがある：

Interface Agent:
人と外界とのインターフェース
DAI Agent:
分散 AI でいう Agent
Multi-modal Agent :
Multi-modal UI, Agency model

また，日本の人工知能学会，人工知能基礎論研究会によれば，次のカテゴリーがある：

1) 概念レベルの用法

a. 自立知能を目指すもの
Autonomous agent:
各自の意思決定原理，機構に基づく
Intelligent agent:
心的状態 (mental state) をもち問題解決や学習機能を有する

b. 分散知能を目指すもの
Society of mind:
知性を実現する単位を agent(Minsky)
Multi-agent:
協調や交渉などの相互作用

2) 応用レベルの用法

a. Software agent:
ネットワーク中に存在する自立的なソフトウエアの総称
b. Interface agent:
コンピュータの利用者とコミュニケートするもの．擬人化されたものが多い．
c. Believable agent:
迫真性を備えた agent．情感をもち通常のソフトウエアを超えている

3) 実装レベルの用法

a. Agent oriented programming:
オブジェクト指向の発展形
b. Mobile agent:
ネットワーク内を自由に動き回り情報を収集
c. Telescript agent:
General Magic が提供するスクリプト言語

2.2 フォーマルな技術としてのオブジェクト指向

(1) オブジェクト指向の特徴

オブジェクト指向の特徴は，抽象データ型，インヘリタンス，ポリモフィズムという言葉で表わされる．この特徴により下記のテーマに合致している：

- さまざまの分野における異なった現れを同じ表現方法に統一 (オブジェクト)
- コンピュータ·アーキテクチャ (OS, ハードウェアなど)

- Programming Languages:
  Class and Object
- Programming Methodology:
  Modularization and Information Hiding
- Database Model:
  ER(Entity Relation) to Object Model
- AI の研究:
  Object Oriented Abstraction (Theory of Frames)
- 哲学と認知科学:
  Process or Object(Greek), Object-oriented View (Descartes), Agent(Minsky)

(2) 抽象データ型

抽象データ型は, Pascal にある User-defined Types から拡張されたもので, カプセル化 (encapsulation) の機能が追加されている. この機能は, Simula 67 に初めに実装された.

抽象データ型は, データの型を隠蔽 (information が iding) するのみでなく, その型に対する操作をも含んでいる. つまり, 外部から直接データをアクセスすることを禁止している. オブジェクト指向では下記のように表現される.

オブジェクト ＝ データ ＋
手続き ＋
メッセージ送受信機能

あるオブジェクトにアクセスするときは, そのオブジェクトに手続き (操作) をメッセージで渡す. オブジェクトはそのメッセージを受け取り, 必要な手続きを実行しその結果を依頼してきたオブジェクトに通知する.

(3) インヘリタンス

インヘリタンス (継承) は, オブジェクトの特性を一般化する機構である. 実現方法には, subclass によるものとと delegation によるものとがある. データ構造と手続きのインヘリタンスがあり, そのメカニズムには single inheritance と multiple inheritance がある. is-a 関係と is-kind-of 関係で分類し整理された class は, 木構造で表現される.

別のいい方をすれば, インヘリタンスとは, class に, その superclass の機能を再利用させる機構だといえる. これにより, 差分プログラミング (differential programming) が可能になる.

(4) ポリモフィズム

ポリモフィズムは, 同じメッセージに対して, 異なった class がそれぞれ独自の方法で応える機構である. オジェクトのデータの種類が何種類あっても, それを画面に表示するためには, display という要求を, オブジェクトに送ればよい. これを受け取ったオブジェクトが, 自分のデータに合った手続きを選び実行する. これを遅延束縛, メソッドサーチと呼ぶ.

2.3 オブジェクト指向技術を用いたシステム開発

(1) オブジェクト指向の準備

オブジェクト指向技術を用いてシステム開発を行うためには, 下記の条件を満たすことが望まれる：

- 手続きプログラム言語の経験:
  ポインタ, 抽象化, 論理化
- データベース設計の経験:
  ER モデル, 意味データモデル, 汎化, 集約, 類型, 連関, 静的視点
- リアルタイムや並列処理:
  並列分散, 状態遷移, ペトリネット, 動的視点
- 哲学, 心理学, 社会学的センス:
  抽象化, 認識, プロセス, 視点

(2) プログラミング環境

オブジェクト指向技術を用いたシステム開発には, 次の機能を持つ環境が必要とされる：

- メッセージを送っているオブジェクトを捜し出す
- あるメッセージに答えられるオブジェクトを捜し出す
- あるオブジェクトの内部状態を変更するメッセージを捜し出す
- 汎化-特化関係をブラウジングし整理できる
- 全体-部分関係をブラウジングし整理できる
- 例題によりプログラミングを支援する

2.4 オブジェクト指向技術の論点

オブジェクト指向に関する議論は数多くあるが身近で起きている論点を紹介する.

(1) オブジェクト指向に欠けているもの
(by D. Bjorner)

- 抽象化, 詳細化
- 検証
- 引数化
- 修正 (enrichment, derivation)
- 主体性 (subjective)

(2) Formal Method とオブジェクト指向の比較
(by 山崎利治)

フォーマル	オブジェクト指向
理性論	経験論
集中	分散
還元	反応

(3) OOP と AOP (by Y.Schoham)

Items	OOP	AOP
Basic unit	Object	Agent
Prameters defining State of basic unit	Unconstrained	Beliefs, Commitment, Capabilities, Choice,...
Process of communication	Message passing Response method	Message passing Response method
Type of message	Unconstrained	Inform, Request, Offer, Promise, Decline,..
Constraints on methods	None	Honesty, Consistency,...

3. ある System 開発でのオブジェクト指向の適用

3.1. 概要

ここでは，ある特殊なコンピュータの応用領域において，オブジェクト指向技術がいかなる影響と効果をあげたかを，経験にもとづいて論ずる．

その経験とは，1986 年から 1988 年にわたる 3 年間，PFU の研究開発センターで実践された "KingFischer" というプロジェクトである．このプロジェクトの技術的なタイトルは，"Image-Based System"であった．

このプロジェクトでは，従来にないコンピュータ･システム，つまりイメージを基本としたシステムを研究開発し，コンピュータの新しい市場を開拓することを狙った．このプロジェクトの中で，筆者はソフトウェア･グループの "Playing Manager" をつとめた．

このプロジェクトでは，目的とするシステムを設計し作成するためのシステム･プログラミング環境として，Smalltalk-80 を選んだ．

その結果，Smalltalk-80 はわれわれに多大な影響を与えた．開発方法論，プログラミングの仕方，ものごとの思考方法，組織の仕事のやり方などで，ほとんど革命的な変革が起きた．

本稿では，このことがらを，次にあげる技術的な観点から考察したい：

特定の応用領域としての "Image-Based System"
"Image-Based System" に要求される機能
システムプログラム言語としての Smalltalk-80
Smalltalk-80 の採用で何が変わったか?
特定応用領域とオブジェクト指向技術の関係

そして最後に，特定応用領域指向アプローチとオブジェクト指向技術に対する設問を，いくつか示すことにしたい．

3.2. 特定の応用領域としての "Image-Based System"

このプロジェクトの目的は，新しいコンピュータシステム，"Imaged-Based System" を開発することであった．

これまでの伝統的なコンピュータシステムは，いわば"Code-Based System" だと呼べよう．そこでは，コンピュータが取り扱う基本単位が，あかかじめ決められた形式にしたがっている．たとえば，コンピュータの命令セット，数値データの表現形式，つまり整数表現や浮動小数点表現がそれである．

これに対して，"Image-Based System" では，コンピュータが扱う基本単位を，イメージを含めたマルチメディアのチャンク (chunk) とした．そして，このチャンクを載せて取り扱い可能とするために，シート (sheet) の概念を導入した．

シートの上には，多くのチャンクが載り，このシートを組み合わせ操作することにより，マルチメディアデータを対象としたシステムが構築できるようにした．いまでいう "hypermedia system" に酷似したものである．しかし，10 年前には，まだこのようなシステムは実現されていなかった．われわれのキー技術は，このシートに代表される．すべてのチャンクは，シート上に表現され，そのシートの構造を表現できかつそれを実行できるような機構を考えた．

最も重要な問題は，いかにして表現されたシートを高速に実行するかであった．目安として，一秒間に 30 枚のシートの実行を目標とした．

3.3. "Image-Based System" に要求される機能

このシステムを設計し研究開発するためにはつまりマルチメディアデータを表現し操作するためには，強力な新しいメタファが必要とされた．

われわれは，従来のコンピュータ言語を含めて，いくつかの実験を行った．下記のあげるリストがその経過である：

UNIX と C
Lisp
Prolog
LIPs
Smalltalk-80

C 言語は，あまりにもプリミティブすぎて，表現能力が貧弱である．

次の Lisp と Prolog は，パワフルではあるが，このプロジェクトの目的には合致しなかった．その理由は，これら

２つのコンピュータ言語は，強く "Physical Symbol System" に結びついていることであった．それは，記号処理や人工知能の分野に近いものである．

一時，このプロジェクトでは，統合プログラミング環境，LIPs(Lisp, Ingres, and Prolog) を開発した．これは，手続き部分を Lisp で，宣言的な部分を Prolog で，そしてデータベース部分を Ingres で記述しようとしたものであった．

しかし，われわれの狙いは，マルチメディア･データを直接操作し，それらを使ってアニメーションを実現することであった．論理的な記号処理だけを目的としていなかった．

最後に，われわれは Smalltalk-80 に出あった．そして，プロジェクト･メンバーのほとんど全員が Smalltail-80 システムに満足した．

このときわれわれが念頭に置いていた機能要求を次に示す：

動的で，かつビジュアルな表現ができる
オブジェクト指向プログラミング技術
マルチメディアデータが取り扱える
ネットワーク構造のシートが構築できる
(Hypermedia System)
Visual Programming ができる
特定の分野に対してメタモデリングができる
大規模な情報システムを対象にできる

3.4 システムプログラム言語としての Smalltalk-80

われわれは Smalltalk-80 をシステム記述言語として採用し，これを用いてシステム設計を行った．Smalltalk-80 は，前節であげた機能要求をほぼ満足した．われわれは，このシステムで実行可能なシートを設計し，構築し，実行させることができた．ここでわれわれが遭遇した困難は，技術的なものと，研究開発の組織に関するものとの，２種類であった．

技術的な困難は，Smalltalk-80 は機能的には満足だが，実効速度がきわめて遅いということと，システムが閉じており，他のシステム (たとえば，Unix や DOS) と連携できないという点でであった．

この困難に対してわれわれが立てた解決策は，CTALK という新たな言語処理システムを作成することであった．それは，簡単にいえば，Smalltalk-80 で開発したシステムを自動変換して，C 言語レベルに変換し，速度とシステムの閉鎖性の両方の問題を解決しようと試みたのである．

もう一つの組織的つまり社会的な困難は次の通りであった：

日本のコンピュータ社会構造はオブジェクト指向技術開発に向いていない．
新しいコンピュータ言語を習得するのは難しい．
アニメーションのためには新しいプログラミング言語が必要である．
システムデザイナーにはコンピュータ知識よりもドメイン指向知識が必要とされる．

最初の困難は，日本企業がとっている極度なトリー構造の管理組織に起因している．上位から下位への一方向の情報伝達は効率よく実行されるが，両方向の，かつ動的でフレキシブルな議論がうまくいかないのである．

オブジェクト指向による開発では，チーム全員参加の実証的なアプローチが要求されるので，従来の官僚的な組織では多くの問題が露呈される．

２番目以降の困難は，ここではあまり重要でないので，議論は省略する．

3.5. 何が変わったか?

Smalltalk-80 の採用で何が変わったか? に対する答えは，"すべてが変わった!" である．それまでのわれわれが実践していた"Desktop Metaphor" は" 紙と鉛筆" であった．そして，システム設計のやり方は紙をベースとした "Water-Fall " 型であった．レビュー，工程管理，定期的な技術検討会，進捗管理など典型的なものであった．

これに対して，Smalltalk-80 では，"Desktop Metaphor" がコンピュータになり，紙ベースからコンピュータベースになり，"Waterfall" がいわゆる" Object-Oriented Methodology" に変わった．

われわれは，コンピュータ上に表現されたオブジェクトそのもを共有しながら，それをベースとして考え，議論するようになった．そして，このプロジェクトのような目的を達成するためには，何度にもわたる "try and error" を繰り返しながら，目標とするシステムの概念設計をすることが重要であると理解した．

3.6. 特定応用領域とオブジェクト指向技術の関係

システム･アーキテクチャは，ターゲットとするシステムに強く依存している．特に，応用プログラムと，それを実現するシステムプログラムは，応用分野からの要求に依存すべきである．われわれの "Image-Based Sysytem" の場合，Smalltalk-80 を使って対象システムのプロトタイプを開発した．そして，Smalltalk-80 言語を，設計言語，プロタイピング言語，表現言語，それにシステム開発言語として，全面的に使用することができた．これは，対象としたシステム，つまり特定のドメインによって，要求される機能が異なることを示している．

表現を変えれば，ソフトウェア･システムの開発アプローチには，大きく分ければ２つの分類があり，今回のプロジェ

表4　分析的アプローチとIA的アプローチ

比較項目	分析的アプローチ	IA的アプローチ
computer science	mathematics	experimental
main tools	formal specification	experimental tools
basic challebge	clear thinking	better tools needed
methodoplogy	verification before make	design is error crrecting
problem	well-structured	ill-structured
solution	best solution	satisfying solution

クトはその一方の典型と考えられる．２つの分類とは，Colorado大学のG.Fischer教授が提唱しているもので，一方は"分析的アプローチ"，もう１つは"IA (Intelligence Augmentation) 的アプローチ”である．

これら２つのアプローチの比較を表4に示す．

3.7. まとめ

われわれはSmalltalk-80を用いていくつかのコンピュータ応用システムを開発した．たとえば,"Image-Based Sysytem", "PWB(Prototyper's WorkBench)", "Active Book", "Hyper Composer"などである．

このうち２つのシステムを，最初はSmalltalk-80で記述し，その後CまたはC++言語に書き換えた．その主な理由は，プログラムの実効速度と，できあがったシステムのオープン性にあった．前者では，実行時のガーベジ･コレクションが問題になり，後者では，既存システムとの連携が問題となった．そして，この傾向は世界的な主流になりつつあるが，私にとっては本筋の解決策とは考えられない．

最後に下記の問題提起をし，本項のまとめとしたい．

- C++とそのライブラリーは真の解ではない．
- ObjectWorkを拡張または敷衍し新言語を開発することが真の解である．
- システム開発の方法論はそれが使われるドメインに極度に依存する．
- システムアーキテクチャは時代の要請に基づいている．
- オブジェクト指向をベースとした並列分散システムアーキテクチャがいま必要とされている．

参考文献

プラトン,「饗宴」, 世界文学大系, 筑摩書房, 1995年.

C.G. ユング,「人間と象徴　無意識の世界」, 河合隼雄監訳, 河出書房新社, 1975年.

H. ベルグソン,「形而上学入門」, 世界の名著.

F. ニーチェ,「権力への意志」, 筑摩文庫.

G. Booch, Object-Oriented Analysis and Design With Applications. The Benjamin/Cummings Publishing Co. Inc. 1994年.

青木淳.「オブジェクト指向システム分析設計入門」. SRCハンドブック, 1993年.

J.Martin, Principles of Object-Oriented Analysis and Design. Prentice-Hall International Editions, 1993年.

Y.Shoham, Agent-oriented programming. Artificial Intelligence volume 60, pp.51-92. Netherlands. Elsevier, 1993年.

G.Fischer, "Artificial Intelligence and Human-Computer Communication - Does Software Engineering Have a Future Without Them? , SRA Seminar Text, 1989年.

Visual Basic の光と陰

― あるプロジェクトでの開発経験から ―

桜井 麻里
(SRA)

SEA 横浜支部 (YDOC) 1995 年 7 月 28 日の例会

こんばんは! このYDOCのMailing Listの管理人としてみなさまおなじみの桜井麻里です. 今夜は「Visual Basicの光と陰」という題で, お話をさせていただきます.

はじめにお断りしますが, 私は, VBのすべてを知りつくしているわけではありません. 昨年のゴールデン・ウィーク明けから約1年間, あるプロジェクトに助っ人として参加したのですが, そこでVBを使って比較的大きなアプリケーションを作ったことがあるというだけです. 今日は, そのときの経験をもとにお話します. VBの応用としては, かなり偏った使い方をしましたので, 感想も偏っているかもしれません. ご了承ください.

はじめに, VBを使ったことのない方もいらっしゃると思いますので, その紹介を簡単にさせていただきます. VBの入門講座ではないので簡単にすませます. VBは実物を見ないとわかりにくいので, のちほど, 時間を取ってデモをお見せしたいと思います.

OHP-1 [Visual Basic のうわさ]

- GUIアプリケーションが簡単に作成できる.
  - ーウィンドウ・ベースのイベント駆動プログラミングが簡単.
  - ービジュアル・プログラミング, インタプリタ+構文エディタ
  - ー文法が簡単でなじみやすく, だれでも簡単にプログラミングできる.
  - ー標準部品は多くないが, 市販のカスタ・ムコントロール (VBX) を使えば結構いろんなことができる.
- 足りないところはCやC++で書いたDLLで補えるので完璧!
- Windows上のアプリケーションを短期間で開発するにはこれしかない!
- ExcelやAccessで使えるマクロ言語もVisual basic. おぼえておいて損はない!

これは私が適当に書いたので, ほんとうにこういう噂が立っているかどうか知りませんけど, 雑誌などを見るかぎりでは, こんな噂が立っているんじゃないかと思います.

こういうわけですので, 近ごろでは, 営業さんがVBの仕事を持ってくることが, けっこう多いわけです. みなさんご存じのように, 好むと好まざるとに関わらず, 巷ではWindows 3.1がはやっています. そうすると, 必然的にVBの仕事もふえてくるわけです.

今回, 私が助っ人として担当したプロジェクトも, そういう背景の中で, 某社の営業さんが「Windows3.1でVB」と決めてしまった仕事でした.

OHP-2 [VB の概要]

- 話のベースはVersion 2.0日本語版
- 手続き型プログラミング言語
- 構造化プログラミングを支援
  - ーモジュール化と3つのレベルのスコープ・ルールをうまく使えば情報隠蔽も可能.
  - ー工夫すれば複数名のチームによる開発も可能.
- ウィンドウ・ベースのイベント駆動プログラミングを支援
- インタープリタ, 構文エディタ, ビジュアルな開発環境, コンパイラを装備
- Visual Basicで扱うもの.ti -2 ・いたるところにある64Kの壁! [参考文献 1]
  - ー Version 4.0 ではWindows95対応なので64Kの壁はなくなるという噂.

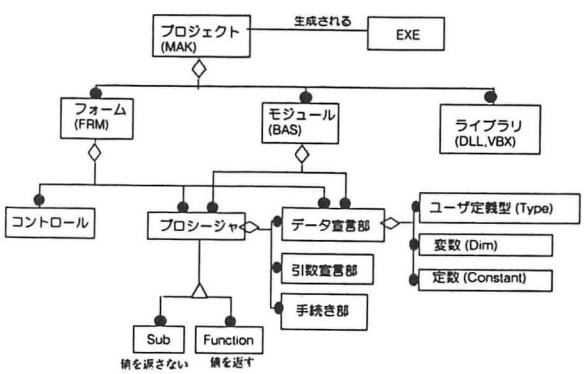
アメリカでの最新版は3.0ですし, そろそろWindows95対応のVisualBasic Version 4.0がで出てくるころなのですが, 日本ではまだ最新版は2.0ですし, 私たちが使ったものも2.0日本語版でしたので, そのお話をします.

文法はここでは説明しませんが, 手続き型で構造化プログラミングを支援した言語だといえば, 大体おわかりいただけるでしょう. If Then Else Endifとか, Do Whileとかが使えます. DOSの世界では当然のこととして, やはり, 大文字小文字の区別はされません.

VB で提供されている環境を使えば，構文エディタなので，編集しているそばから文法チェックをされてしまうのですが，それがきらいな人は，お気に入りのテキストエディタを使って編集しておいてから VB インタプリタを起動しても，いっこうにかまいません．私たちのプロジェクトでも，そのようにしている人が何人かいました．

Visual Basic が扱うものを次の図に示します．

OHP-3　Visual Basic で扱うもの



この図で，EXE , MAK, FRM, BAS と書いてあるのは，DOS でいうところのファイルの拡張子です．用語は VB の用語です．

この図では，VB でいうプロジェクトとは，１つの EXE にするファイルの集まりの単位であり，プロジェクトは，0 個以上のフォームと 0 個以上のモジュールと 0 個以上のライブラリから構成されると書いてあります．

そのように書くと，プロジェクトは何もなくてもよいように見えてしまいますが，たぶん，フォームかモジュールのどちらか１つぐらいはないとダメだと思います．フォームだけでモジュールがなくてもよいことはたしかですし，モジュールだけでフォームがなくてもよい (ただし，EXE は SubMain から始まる) のが....

それから，この図では描くのを忘れてしまいましたが，フォームとコントロールは１つ以上のプロパティを持ちますので，コントロールという四角の下には，プロパティという四角を作って，フォームおよびコントロールとプロパティを線で結び，その線のプロパティ側に 1+ と書いてください．

フォームというのは，いわゆる画面定義です．コントロールというのは，X-Window でいうところの Widget のようなものです．標準で用意されているコントロールには，ラベル，リスト･ボックス，テキスト･ボックス，スクロール･バーなどがあります．

この図の右側に，DLL と VBX を「ライブラリ」として一つにまとめて書いてありますが，いわゆる手続きのライブラリは DLL のほうです．C とか C++ でライブラリを作って，ダイナミック･リンク･ライブラリとして実行時に結合することができます．VBX というのは，カスタム･コントロールと呼ばれていて，コントロールのライブラリのことです．

プロシージャとしては，Sub と Function があります．ここには，単純に，Funtion は値を返す，Sub は値を返さないと書いてありますが，それは，そのプロシージャそのものが値を返すかどうかという話でして，引数として渡したものが値が変わって返ってくるかどうかとは別のことです．引数の扱いとしては，VB のデフォルトはアドレス渡しです．ですから，デフォルトでは，呼ばれたほうのプロシージャ内部で渡された引数の値を書き換えてしまうことができます．値渡しにしたい場合には，プロシージャの引数定義のときに，引数名の前に ByVal というキーワードをつけます．

Funtion getOperatorName (ByVal operatorID As Long) As String という感じです．

情報隠蔽を促進し保守性を高めるためには，この ByVal が有用なのですが，残念なことに，ユーザ定義型には ByVal をつけることができません．ユーザ定義型は既存のデータ型よりも当然大きくなりますので，引数コピーにコストがかかるとか，ただでさえ，スタック･エリアが 20KB しかないのに，ユーザ定義型を ByVal でコピーしていてはすぐにあふれてしまうという理由で，許されていないのでしょう，きっと．

それから，この図だと，ユーザ定義型はどこにでも書けてしまうようにみえますが，モジュールの先頭にしか書けなかったような気がします．

OHP-4　[スコープ･ルール]

- ･3 つのレベル (グローバル，モジュール，プロシージャ) のスコープがある．
- ･プロシージャ (Sub, Function) は，Private をつけて宣言しないかぎり，グローバルとなる(同じ EXE 内のどこからでも呼べる)．Private をつければそのモジュール内部のみで有効となる．
- ･変数および定数の宣言は，プロシージャ内部での定義であれば，プロシージャ内部のみで有効．Global をつけて，モジュールの先頭で定義してあれば，グローバル．Global をつけずに，モジュールの先頭で定義してあれば，モジュール内部のみで共通に有効．

表1 ソースコード量

機能単位名	モジュール数	モジュールファイル行数	大域プロシージャ数	Privateプロシージャ数	フォーム数	フォームファイル行数	フォームコントロール数	フォームプロシージャ数
基本制御	20	7,287	263	89	7	1,342	69	38
障害管理処理	14	4,061	109	50	2	1,925	108	22
詳細情報処理	1	1,273	7	42	1	1,186	75	6
アイドル取込処理	1	706	14	23	1	434	28	5
指示処理	1	660	14	21	1	235	12	6
検印処理	4	2,787	66	58	2	1,673	101	27
承認処理	4	1,839	56	43	1	756	45	14
システム情報処理	3	1,527	51	34	2	606	29	18
システム起動終了部	1	1,054	19	29	1	162	5	3
振分け＆フック＆メイン部	4	2,892	191	3	1	202	7	8
端末内共通部	5	2,013	58	9	1	192	10	5
通信共通部	8	2,577	27	10	1	417	4	8
その他の共通部	4	497	13	0	5	1,248	45	29
officer.exe小計	70	29,173	888	411	26	10,378	538	189
sysstart.exe小計	13	3,020	44	22	5	1,109	48	22
sysend.exe小計	9	1,189	21	6	7	1,114	62	19
合計	92	33,382	953	439	38	12,601	648	230

・フォーム内で定義されているプロシージャは，そのフォーム内部でのみ有効 (フォーム外部からは呼べない).
・フォーム内で定義されているコントロールやプロパティは，フォーム名をつけて指定すればグローバル・レベルでアクセスできる.

では，このへんで，私たちが何を作ったかをお話しましょう.

それは，一言でいってしまえば，いわゆるATMの自動監視システムです．ATMの自動監視以外にもいろいろな機能はあったんですけれども....

利用者は某地方銀行の照会センタの人たちです．私が担当したのは，その中の「役席端末」という端末です．それは，偉い人というか権限のある人が使う端末で，システム全体で1台しかありません．いわゆるクライアント・サーバ型のシステムでして，サーバとしてはUnix WS，端末としてWindows 3.1を使っていました．各端末は，このシステム専用に使われているので，余計なプログラムは一切動かしていませんでした．電源をいれるといきなりこのシステムの機能が立ち上がるようにautoexec.batに記述していました．このシステムのサーバ・プロセス自体も役席端末から起動/停止するようになっていました.

ソース・コードの量を機能単位ごとに調べてみたので，ご覧ください (表1).

ファイルの行数というのは，もちろん，テキスト形式で保存したときの行数です．フォーム・ファイルの行数はあまり意味がないかもしれませんが，コントロール数が同じなのに行数が大きかったら，フォーム・プロシージャが長いというふうに解釈できるという意味で，載せています.

詳しい説明はしませんけれども，それぞれ独立した機能がたくさん詰まっていることがおわかりいただけると思います．量としても，VBで書いたアプリケーションとしては大きいほうなのではないかと思います．これらの機能を並列に動かしていました．ユーザに対して一度に表示できるのは1画面ですけれども，裏では他の機能のためにDBと通信するスレッドが非同期的に複数動いていました.

Unix WSは，DBも含めて2台あって，それぞれで，DBサーバ，ノティファイア，ピアなどのサーバ・プロセスが動いていました．端末はマスタ側のピアを介してDBサーバやノティファイアと通信するわけです．サーバとの通信に関しては共通のお約束があり，それを実装した部分がシステム共通で提供されていました．それが，通信共通部です.

あらかじめ，端末立ち上がり時に，自分が興味あるテーブルのレコードやカラムを指定して，挿入があったら教え

てくださいとか，更新があったら教えてくださいという形で，ノティファイアに登録しておきます．そうすると，実際に挿入があったときに，ノティファイアが教えてくれるわけです．もちろんピア経由です．端末としては，ノティファイアからの通知がきたら，それをトリガとして，さらにレコードを取得したり，画面に表示したりという手続きをするようにプログラムしておくわけです．

複数のスレッドが並列に動いているので，ノティファイアからの通知が来たら，自分の端末の中のどのスレッドがお願いしておいたものかを判断して，該当するスレッドに処理を飛ばさなければなりません．それをやっているのが，この表の「振分け＆フック＆メイン部」の中の振分け部です．ここのコード量が他に比べて多くなっているのは，ひたすらメッセージを解析してあちこちに振り分けているからです．私も含めてメンバーのほとんどは，こういうシステムをやった経験がなかったので，この部分でずいぶん苦労しました．

このソース･コードの量は，私がシステムをかなりスリムにしたあとで計ったものです．できてすぐの頃は，もっと多かった．たとえば，障害管理処理は，当初，BAS ファイルだけで 11,000 行ぐらいありました．ところがあまりにも遅かったのと，通信量が多すぎて通信バッファがオーバフローしてしまったことが何回かあったこと，本来，Unix 上でデーモンにしてしまえるような機能であったことなどの理由により，今年になってから 6,000 行分ぐらい削って，その機能を Unix 上のデーモンにしてしまいました．DB とのやりとりは Pro*C を使って書いたので，端末でやるよりずっと楽でしたし，実行速度もずっと速くなりました．はじめからデーモンにしていればよかったのでしょうが，気がつくのが遅かったんです．私が仕様を理解し終わらないうちにプログラマが来て，コーディングを開始しなければならなかった．とにかく，急かされていましたから．夏ごろには，デーモンにすべきだ気がついたんですけど，もう，遅かった．でも冬になって，正式運用を始めたあと，遅いとか，通信がらみで落ちやすいとうことでクレームがついたのと，仕様変更要求がたくさん来たのでそれにからめて時間を取りやすかったということで，デーモン化に踏み切ることができました．同様に，この表の一番上にある基本制御系のスレッドも，ほんとうはデーモンにすべきだなぁと開発途中から思っていたんですが，こっちは私が設計した部分ではなかったので，いまだによく理解していませんし，いまさら面倒なのでやっていません (内容は，ATM と公衆回線経由でやりとりする装置の制御です)．

この表で，他に比べて Private プロシージャ数の比率が高い機能単位はすべてモジュール数が 1 だということにお気づきでしょうか? 逆にいえば，モジュール数が複数だと Private にしようと思ってもしにくいので，Private の比率が低くなるということです．それは，VB のスコープ･ルールで，モジュール (BAS ファイル) 単位より上位のものとしては，いきなり，グローバルになるからなのです．今回みたいなアプリケーションだと，機能単位が違えば扱うデータもぜんぜん違うので，情報隠蔽のための枠組みがもっと豊富だったら，もっと隠蔽させることができたんですけどね．

OHP-5 [感想 1 ― 苦労した点 (1)]

- 「メモリ不足」のエラー頻発．64K の壁に泣かされた．
  - ―リスト･ボックスの容量が 64K なので，すなおに使うことができなかった (たかだか 40 バイトのデータを 5,000 件程度表示させたかっただけなのに...)
  - ―グローバル･データのサイズ制限にひっかかった．
  - ―グローバル･シンボル･テーブルのサイズ制限にひっかかった．
- 単純なはずなのに適切なコントロールが見つからなかった．
  - ―いわゆる一覧表示のところで，カスタム･コントロールの MhSpread を使おうとしたが，1,000 件程度のデータでもまともに扱えないほど遅く，容量的にもきびしいことがわかったので，あきらめた．
  - ―結局，活躍したのはラベル･コントロール．
- スタック･サイズ 20K の制限にビクビクした．

最初にひっかかったのは，通信共通部用のユーザ定義型の定義ファイルでした．すべてのユーザ定義型で，1 つずつ Global 変数も定義してあったんです．DB のスキーマで定義されているものがすべてあったので，それだけで 64K の大部分を占めてしまっていました．不要なものを削除することで対処しました．私が担当した端末は最後まで仕様が決まらなかったので，他の端末に比べて一番開発が遅れていました．だから，他の端末チームの人たちが苦労したあとで楽をさせてもらえると思っていたのですが，そうはいきませんでしたね．64K の壁に関していえば，私たちがまっさきにひっかかってしまい，最後まで苦しみました．

扱うデータ量でも，私たちの端末が一番多かったのできびしかったように思います．他の端末が安直に ListBox や MhSpread を使っているようなところで，私たちはラベルを行数分ならべて，その横に垂直スクロール･バーをつけて，自前でカーソル制御やスクロール制御をしていたわけですから．

OHP-6 [感想 2 ― 苦労した点 (2)]

- Windows3.1 は，やっぱりよく落ちた．Winsock も落ちや

すかった.
・VB とは直接関係ないが, 通信関連のトラブルには苦労した.
・1つの EXE の中で, 仮想的に複数スレッドを並列に動かさなければならなかったので, コーディング・スタイルを確立するまでに苦労した.
・最初は分担作業のしかたが (分割のしかた) がよくわからず悩んだ.
・エラー・ハンドリングが簡単そうで難しい. インタープリタで使用しているときは, エラーが起きた箇所で止まってソースが表示されるのでわかるが, EXE をリリースした後は, どこでエラーが起きたのかつかめず, デバッグに苦労した.
  ーデバッグ文をログ・ファイルに吐かせる機能はついているが, ログ・レベルが1でも, あっというまにログ・ファイルが 10MB 以上になってしまうので, 現実にはログをとってもらえない.

便利だった点も1つぐらいいわなければいけないので, まとめれば;

OHP-7 [感想3ー便利だった点)]

・はじめて VB を使った人が多かったが, なんとかなった.
・開発初期はどんどん作成できる. ある程度の形になるまでは簡単.
・画面まわりは初期開発も修正もほんとうに簡単.
・VTSOCKET の扱いも簡単だった.
・モジュールやフォームのソース・コードをファイル単位でテキスト形式で保存できるので, なにかと便利だった.
  ー分担作業がやりやすい.
  ー Unix 上のいろんなツールが使える.

今回のプロジェクトでは, 端末プログラミングの最盛期 (去年の6月から8月) には, いろいろな部署や協力会社から, たまたま手の空いていたプログラマが何十名か呼ばれて, 客先のタコ部屋に詰め込まれていっきに作ったわけですが, それでもなんとかなったのは, VB が馴染みやすく簡単な言語だったからだともいえるでしょう.

むかし, Macintosh の HyperCard/HyperTalk を使って2人で1つのスタックをプログラミングしたことがあります. Apple 社が出しているスタック・デザイン・ガイドによれば, 1つのスタックの開発では, HyperTalk プログラマは1人にするようにと書いてあります. それをあえて2人でやってみたのですが, やはり大変でした. その1つの理由としては, スタックをテキスト・ファイルとして書き出すことができないということがあったように思います. 別ファイルにすることもできないので, 毎日忘れないうちに2人で確認しあいながら, お互いにやった作業をマージしなければなりませんでした. それに比べると, VB はファイル分割できて, しかもテキスト形式でセーブできるので, 分担作業がしやすいというわけです. HyperTalk と VB を比べるのはナンセンスかもしれませんが, GUI を持ったアプリケーションを簡単に開発できる道具立てとしては, 比べてもよいと思います.

テキスト形式で保存できるので Unix 上のいろいろなツールが使えるとはいっても, rcs など, diff をベースにしたツールは, そのままではうまく使えませんでした. なぜかといえば, VB の構文エディタは, 保存するときに, モジュール内のプロシージャの順番を勝手に換えてしまうし, インデントも VB の環境設定で変わってしまうからです.

テキスト形式で保存して実際にやったこととしては, 通信共通部用のユーザ定義型の定義ファイルから不要なものを削除するとか, フォントの定義を抜き出して一覧表示するとか, シンボル・テーブルが 64K を超えないかどうかあらかじめ簡単にキーワードを抜き出して計算してみるとかですね. いずれも必要に迫られたので, perl を使ってそういうスクリプトを書いてました.

最後に, チームで VB を使って開発する場合の秘訣をまとめておきます.

OHP-8 [感想4ー VB による開発の秘訣]

・仕様を理解した時点で, センスのある人が介入して, 命名規則を決めたり, モジュール分割するのが理想的.
  ー VB にかぎったことではないが, 分割のしかたが悪かったり, 配慮の足りない人が命名規則や共通部を作ると, VB の場合は, すぐに 64K の壁にぶちあたる.
・シンボル・テーブルなどに関する64K の壁は, モジュール強度を高くし, 情報隠蔽を促進することである程度回避できる. 名前を短くするのはその後でよい.
・仕様の理解が間に合わないうちにプログラマがたくさん来てしまったときは, 画面定義や共通になりそうな部分の洗い出しなどをしてもらって, 時を稼ぐのも手. フォーム定義は1人で地道にやってもらったほうが統一がとれてよい.
・フォームのプロパティはグローバルなので, どこからでもアクセスできてしまうが, フォームを扱うモジュールは, それぞれ1つに決めておいたほうがメンテナンス性がよい.

参考文献 1: 峰晴美,"プロフェッショナルのための Visual Basic ケーススタディ", 技術評論社, 1995.

Q & A

会場: VB ではマクロというのは使えないんですか?

桜井: はい. マクロという概念はないと思います. 定数宣言の Const ですら, コンパイラあるいはインタプリタのシンボル·テーブル領域を食ってしまうぐらいで, CPP のようなプリプロセッサは通してないようですし, 手続き的なものはすべてプロシージャ (Sub か Function) にするしかないと思います.

会場: 64K の制限を回避できるようなリスト·ボックスもあると思いますが,....

桜井: 探したんですが, あの時には見つからなかったのです. カスタム·コントロール·セットは 1 つ購入してあったんですけど, それにはありませんでしたし, fj や NIFTY に聞いても適当なのが見つからなかった. あまり時間的な余裕もなかったので, ラベルの配列とスクロール·バーで, ごりごりコーディングしてすませてしまいました.

会場: たしかに去年の 6 月ごろじゃ, あまりなかったかもしれませんね. いまなら, いろいろ出ていますけれども....

桜井: えぇ.

会場: スレッド·マネージャのような (汎用性の高い) ものは作らなかったんですか?

桜井: そういうのを一から設計している暇はなかったので, ごりごり作ってしまいました. 実は, 他の端末は (たぶん) 単一スレッドでしたし, 大きくなかったんですよ. 複数スレッドを並列に動かす必要があったのは, われわれが担当した端末ぐらいだったし, 他のチームとはあまり意思の疎通がなかったので, 結局, 自分たちだけでできる方法でごりごりとやってしまいました.

会場: わざわざウィンドウ·システムを使って GUI にする必要がなかったようにみえますが, どうして GUI にしたんですか?

桜井: 文字を大きくしたかったんですよ (笑). 年配の方が使うので, 文字は特に大きくするようにといわれました. 詳細情報画面とか障害管理画面のように表示する情報量が多い場合は 10 ポイントで勘弁してもらいましたけど, 他は 19 ポイントとかを使っています. あと, マウスだけで操作できるということでしょうかね. 一度, 実際に運用されている所を見に行ったんですが, キーボードはしっかり机の奥に隠されていました. それから, このシステムでは, 最初に「Windows であることは見せないように」と指示されました. ですから, Windows のタイトル·バーも見せていません. 必要なものだけをひたすら大きく表示しています. あ, それから, スクロール·バーを使えるというのも GUI を使う利点でしょう. そんなところでしょうか. 落ちたときとかに, Windows のせいにすると, エンドユーザは,「うちは Windows を使って欲しいなんていってないもんね」というわけですが, 私たちだって「Windows を使いたくて使ったわけじゃないもんね」といいたい (笑). Windows3.1 はほんとうによく落ちたし, 通信でも随分悩まされたので, Unix 上で X-Windows で作ったほうが楽だったんじゃないかと, いまでも思っています.

会場: お客さんにどうしても Windows で作ってくださいといわれたら, どうしますか? やっぱり, VisualBasic を選びますか?

桜井: うーむ. Windows だったら, やっぱり, VB でつくるでしょうね. 簡単ですから. VisualSmalltalk とか PARTS Workbench はどうでしょうか?

会場: 今回みたいなアーキテクチャだったら, あまり変わらないかもしれませんね. でも, デバッグは楽になるかもしれない. さっきマルチスレッドのお話がありましたけれども, それを実現するために, 状態遷移機械のようなものをたぶんあちこちにコーディングしたんじゃないかと思いますが, そういう場合には, PARTS だと状態遷移機械クラスを作っておけるので楽になると思います. 他に Visual C++ で作るという選択もありますよね.

桜井: ありますけど, VC++ はやっぱり面倒なんじゃないですかね. 今回みたいに, いろんなところからたくさん人を集めて,「せぇの!」ってやるんだったら, やっぱり, VB でしょう. わたしは VC++ は使ったことがないので, 使ったことのある方, どうですか?

会場: えっ! 突然ふらないでくださいよ. そうですね. 突然人を集めるんだったら, VB かもしれないですけど, VC++ も結構いいですよ. VB だとバイナリで提供されていますから, ライブラリとかカスタム·コントロールでトラブったらお手上げでしょ. でも, VC++ だったら, シンボル·テーブルを見たりして自分でデバッグできますもん.

桜井: そういうことする人は VC++ でもいいでしょうけど....

会場: 今度, 私も Windows を使って, EXE から別の EXE を起動したりすることになるんですが, そういうのは大丈夫なんでしょうか? ウィンドウが表示されちゃったりするんですよね?

桜井: 私たちの端末でも, EXE から EXE は呼んでいますので大丈夫でしょう. VB からだと, Shell という関数で呼ぶことになります. そうするといわゆる DOS 窓がオープンされます. Shell の引数指定で DOS 窓をアイコン表示にしたりすることができます. 実は, DOS 窓を何回も開くと

おかしくなるようなので，何回も呼ばないように気をつけています．おかしくなったときは，端末ごとリセットするしかないです．

会場: モジュールよりも先にフォームを定義したほうがよいんですか?

桜井: ええ，まぁ，アプリケーションにもよると思いますけど．同じようにみえる画面でも異なるフォームの定義のしかたができますし，それによって，コーディングが変わりますから．たとえば，私たちの場合は，共通ヘッダーの部分を独立したフォームに分けておいたわけですけど，そういう風にしたのは私たちだけで，他のチームはすべてヘッダー部分も各画面にいちいち定義していました．そんなことすると面倒なのは見えていたので，私たちはヘッダーを独立フォームにしたんですけどね．いずれにせよ，そのフォームを触るモジュールを１つに限定しておけば，そのあとでフォームの詳細が変わっても影響を受けるモジュールは１つになりますから，変更は楽だと思います．

会場: DB は二重化してあったんですか?

桜井: そうです．Unix WS を２台置いて，そのそれぞれに，Oracle のデータベースをおきました．そして，それぞれで Oracle を動かして，その上で，自前で作った dbserver, notifier を動かしました．検討した当時は Oracle の二重化機能はちゃんとしたものがなかったそうですので，自前で二重化したんです．端末とそれらの DB との間はすべて自前の関数を用いてマスタ･サーバ側のピア経由でやり取りしていました．

会場: ということは，どっちがマスタかという状態を端末のアプリケーションが持っていたということですか?

桜井: マスタがどっちかという判断とか，その状態は，すべて端末側の通信共通部で担当していました．アプリケーションは気にしなくてよかったんです．開発当初は，アプリケーションがいちいち二重化の状態を気にしなくちゃならないかという話もあったんですが，それじゃあんまりだということで，通信共通部でやってもらうようになりました．これをやっているのは，VB で書いた部分です．えーと，この表には書いてませんけれども，通信共通部には C で記述して DLL にした部分もありました．やっていることは，ほとんど構造体のキャスティングです (場内爆笑). VB は C みたいなキャストができないんですよ．そのために，パケットの組立部分は C で書くことになりました．

そろそろ時間ですね．せっかく準備していただいたので，ちょっとしか時間がないですけれども VB のデモをやりましょうか．

司会: そうですね．お願いします．というわけで，これでお開きにします．みなさんどうもありがとうございました．

付録 (その後の YDOC ML でのやりとり)

文責: 編集部

H: 麻里さんのレポートを読みましたが，よく細かな所まで覚えてましたね．テープか何かに録音してたのでしょうか？

S: う〜む．まるでほんとうに (!) テープ起こしをしてできたような原稿ですね :-) (録音してませんでしたよね? 野中さん)．当日の雰囲気がよく出ています．

桜井: 「細かいことまでよくおぼえていましたね」といわれてしまいましたが；もちろん，テープ起こしをしたわけではありません．

もともと，Macintosh で Persuasion というソフトを使ってほとんどテキストで OHP を作っていたので，OHP の部分をテキストにするのは簡単でした．それを見ていると，当日のことがどんどん思い出されてきました．それから，当日にはプロシージャ数の欄が埋まっていなかった「ソースコード量」の表も完成したので，その表を見ていると，いろいろといいたくなってきて，当日いっていないことまで書いてしまったのでした．(^^;

N: ちょっと補足．あのときの私の質問の真意は，"GUI にする必要がなかったようにみえますが" ではなく，"Windows を使う必要がなかったようにみえますが" なのでした．つまり，表計算とかワープロを使わせない専用端末にするなら，Doc 上のの GUI ライブラリのようなもの (があるのかどうか知らないのですが) でも充分だったのでないかと思ったのです．

桜井: そうだったんですか! それは，失礼いたしました．でも，単純に，「Windows を使う必要がなかったようにみえますが,」にしてしまうと，「Windows を使わずに X-Window とかを使えば良かったのに」というふうに読めてしまいますよね．

S: 実は恐ろしい :-) ことに VisualBasic には DOS 版もあるのです．もちろん画面は Windows ぽく作れたりするようです．でもイベントシステムがどうなっているかは知りません．

DOS 上の GUI ライブラリといえば DOS 上で使う (S)VGA 用の Xlib なんてのが世の中にはあって (shareware?)，これを使うと X のプログラムの移植が簡単だとか ... :-p お察しの通り，このライブラリを使用すると X-server は不要です．Client/Server のすべてのコードが含まれているわけですね (うーむ)．

桜井: それを使えば，文字も大きくなりますか? :-) フォントの扱いが X に準じているなら大きくなるとは思いますけど．でも，ビットマップフォントだったら，いちいち大

きなフォントを用意しないといけないとかいうことないですか? いまなら Windows 上の Tcl/Tk を使うという手もありますね. あれでも 64K の壁は避けられないけれど.

N: 私は, Visual BASIC は結構使いものになるという印象をうけました. VB は, C が書けると言いながら, バグしか作れないような人でも, もうちょっとまともに作れると言う点で十分に実用になると思います. Mac でこれに相当するのは, いまのところ HyperCard くらいしかないかな? AppleScript もインターフェースビルダを中心に据えた展開をしないといけないかもね.

D: まあ, COBOL よりはましでしょう :-< でも, IntelligentPad のほうがまだましだと思います. VisualWorks の PowerMac 版がもうすぐ手に入る予定です. 来年になると, VisualWorks のエンジンの上に PARTS Workbench がのった統合製品が出ます. UNIX, OS/2, Windows, Mac で同じ製品が動きます. 駆逐米英, 駆逐 VB!!! :-)

Y: VB の基本的な部分に関する説明がちょっと多い気がしました. もうちょっと奥の深い話しが聞ければもっとよかったです.

桜井: そういわれても, まぁ, 仕方ないことですね. あのぐらいは説明しないと, VB をまったく見たこともない人が話しについていけませんから. あれでも, 少なくしたつもりだったんですよ.

VB そのものに関する奥の深い話ですか? それとも, あのプロジェクトに関するもの? プロジェクトに関してだったら話せば長くなりますけど, VB そのものに関してだったら, わたしはあの程度しか話せない気がします. 本当に偏った使い方しかしていませんし, プロジェクトに 1 年いたといっても, 前半はまわりのひとが VB を使っているのを横目で見ながら, nemacs を使って日本語ばかり書いていたので, 私自身が VB を使ったのはメンバーが解散して保守フェーズにはいった 11 月から 4 月末までぐらいなんです.

Y: 話の中で「モジュールやフォームのソースコードをテキスト形式で保存できるので便利だった」とあるのですが, これがちょっとクセモノでコントロールの高さがデザイン時と微妙にずれることがあるのです. バイナリで保存するとこんなことは起きないので, 内部形式からテキスト形式に変換する時に何らかのバグのようなものが潜んでいるような気がします.

桜井: わたしのチームのメンバーは, ずっとテキスト形式で保存していたので, 私は, 高さに関しては微妙にずれるものだと思っていました. (^^; フォントサイズだって勝手な値を指定しても強制的に直されてしまうでしょ. ですから, その種のものだと思っていました. 位置に関しては, VB 環境下でコントロールの位置を揃えるというやり方を知らなかったということもあり, 大体の位置を決めてテキスト形式で保存した後, テキストエディタで位置のプロパティ値を直して揃えていました. (^^; 実は, 他のチームでは, みんなバイナリ形式で保存していたんですよ. だから, せっかく, 所定の Unix ディレクトリにおいてあっても, 他のチームのは, Unix 上のテキスト・エディタで参照することができなかったんです. 参照できていれば, いろんな技が盗めたかも知れないのに..... でも, そのようにテキスト形式で保存していなかった端末チームも, あとになって Unix 好きと共同作業をするようになってからは, テキスト形式で保存して Unix に置くようになりました.

テキスト形式で保存した場合の VB の不具合については, 他にも, VB を使ったことがあったメンバから聞いていました. 文字列型だったと思いますが, 初期値設定を明示的にしていない場合に, デフォルトの初期値が設定されるはずなのに設定されないそうです. これは, 明示的に設定すればすむ話なので気にしませんでした.

Y: これは知りませんでした. 幸い C の癖がついていて, 初期化は必ずするようにしてるので, 遭遇しませんでした. また, これはバグではないのですが, やはりテキスト形式で保存したときに, 関数が ABC 順で保存されるのですが, ファイルが大きくなってくると, その順番が崩れて保存されます. RCS 等のリビジョン管理ツールを使用すると差分が異常にふえて *.BAS,v や *.FRM,v のファイルサイズが, どんどん膨らんでしまいます.

桜井: 大きくなってくると順番が崩れて保存されるのですか? ほんとうですか? 私の実感だと;

> ファイルの大きさに関わらず, VB 環境下でなにかを変更した後に, そのファイルにエラーがあれば, 関数はアルファベット順ではなく, 変な順で保存される. そうやって変な順序で保存されたものや, あらかじめ別のテキストエディタを使って好きな順序で保存しておいたものを VB 環境下で読み込んだ場合, VB 環境下で変更を加えなければ, 保存し直されることがない. VB 環境下で変更を加えると, エラーがなければ関数がアルファベット順で保存される.

なんですけど.

大きいってどのぐらいでしょう? 行数に関係あります? それともシンボル数かしら? そういえば, コメントもあまり長いのをつけると「メモリが不足です」になりましたね.

順序が変わらなくても, タブ幅などの環境設定が保存した時の環境によって違っていると, 保存されたテキストファイルは変わってしまうので, diff をとってもしょうが

ない状態になります．ですから，私たちはRCSを使うのはあきらめてました．所属会社も力関係も複雑な5人で，一つのEXEを開発していたわけなので，エディタのタブ幅を統一しろなどとはいいにくかったんですよ．

Y: そういわれてみると，心あたりがあります．エラーが発生したらセーブできませんね．どうも，むかしのBASICのイメージがあって，こまめにセーブしないとせっかくの作業が台なしになる恐怖感から，何でもすぐにセーブしてしまってましたから．ファイルの大きさとは関係ないかもしれない．ちなみに，大きいというのはファイルサイズの話で，まずいと思いつつ，128KBytes位だったと思います．

Y: 今回のYDOCの題目から多少それてしまうのですが，VBとOracleとのやり取りとかももう少し詳しく聞ければよかったと思います．

桜井: それは....，プロジェクトの内幕の話をしてしまうことになるので，話すと長いんですよ．dbserverとnotifierとpeerという話ですから．ここだけの話ということで書きますね．

あのプロジェクトでは，少なくとも端末チームとしては，Oracleを使っているということは全く意識していなくて，私たちが呼ばれる前にサーバ･チームと通信共通部作成者によって用意されていたインタフェイス関数を呼んでいただけです．用意されていたインターフェース関数群はきわめて貧弱で，SQL文は，db_open()という関数の引数としてしか書けなかったんですよ．しかも，select*fromしかダメだった．最終的にselect*fromになるのであれば副文の中ではjoinやprojectionを使うことは構わなかったし，もちろん，whereも使えたわけですけど．そのように制限されていた理由は，VBがキャストできないということと関係が深いと思ってます．db_open()したあとは，当然，db_fetch()をぐるぐる回して，db_close()です．更新／削除／挿入は，ノティファイアが認識する必要があったので，db_update(), db_col_update(),db_delete(),db_insert()などのインタフェース関数が用意されていました．

.....以下省略.....

Y: お忙しいだろうに，丁寧なご返事をありがとうございました．他の人たちがどのようにやっているかが，よくわかりました．

われわれの場合は，notifyする必要がなかったので，もっと単純でした．Unix側はOracle標準SQL*Netのサーバを使用し，PC側はOracleのDDEMGR（DDEマネージャ）という製品を使用してます．これはPC側でDDEを使用してSQLの実行，結果の受け取りを行います．VBにおけるSQLの実行はSqlExec()という関数1つで，SELECT, UPDATE,DELETE,その他全部を行ってます．FETCHはオーバーヘッドを少なくするためにWHERE句の条件に該当する行を1回で全部転送してます．結果の受け取りはVBに標準で付いてくるGRID.VBXを使用して2次元配列のようにしてます．

それではまた．

広島原爆 WWW プロジェクト終わる

大場 充
(広島市立大学)

4月の末から5月の始めにかけて日本経済新聞，中国新聞，読売新聞で報道されたように，5月8日に，約1ヶ月の活動予定で広島原爆 WWW プロジェクトが発足し，5月12日には，プロジェクトの参加者のうち24名が集まって，第1回目のプロジェクト発足会議を，日本IBM広島営業所の教室を借りて開催しました．この全体会議には，広島テレビがカメラを持ち込んで取材をしました．

この全体会議に続いて，5月13日には，原爆ドーム前の広場で小中学生のインタビュー，基町クレド前の広場で市民のインタビューを実施しました．小中学生のインタビューでは，1時間ほどで，市内の小中学生や，他府県から見学に来ていた小中学生11名を，インタビューしました．このインタビューも，広島テレビが取材し，報告しました．また，市民のインタビューでは，一般市民の協力がなかなか得られず，市民の非協力的な態度に，地元出身のボランティアの女子学生が泣き出す始末でした．プロジェクト開始早々，前途多難な先行きを暗示していました．

その後は，2人の被爆者のインタビュー，平和資料館の展示資料の写真撮影，被爆直後の状況を物語る写真の収集，原爆の被害の実態に関する情報の収集など，比較的順調に推移しました．平和資料館での展示資料の写真撮影のときには，RCC の取材があり，暗い資料館での写真撮影ではテレビ撮影用の照明ライトで資料を明るくしてくれるなど，われわれの活動にも協力してくれました．撮影の依頼から，撮影までの日数がきわめて短く，さらに天皇·皇后のご来館で忙しいなか，資料館側のご協力によって，どうにか予定のスケジュールで仕事を進めることができました．

その間，5月のデイリー·ヨミウリの記事を見たオランダ放送局の記者から取材の申し込みがあり，平和資料館の資料撮影を行なった日に，広島市立大学へ取材に来ました．この取材には，ベンソン先生が対応しました．ベンソン先生はRCCテレビの取材に対しても，英語でインタビューに答えました．通訳を担当したのは，大場でした．RCC は，われわれの会議の様子も収録していました．

5月の後半には，ホームページの内容も少しづつ充実して来て，何とか見られる形になっていましたので，研究室内にある作業用のサーバへのアクセスを解放しました．広島大学医学部の佐々木先生は，当初からこのホームページにアクセスしていた方のお一人です．また，このころ，東京にあるデザイン会社の社長さんから，プロジェクトのお手伝いができればとのありがたい申し入れがありました．現在のホームページのタイトルとその背景に使われている写真のデザインは，この社長さんとその社員ボランティア1名によって作成されたものです．

6月に入ると，広島市の平岡市長のインタビューと，2人の被爆者のインタビューが続きました．インタビューそのものは1時間から2時間程度のものでしたが，問題はそのあとの英文テキスト作成とその校正作業です．日本語で話されたものを英語に翻訳する場合，文化的な背景の違いなどもあり，直訳では本当の意味(ニュアンス) が通じない場合があります．また，日本語では，あまり論理性を問われることがないのに対して，英語では，論理的に不完全な文は全体の意味がわからなくなるという問題があります．したがって，翻訳は直訳ではなく，ご本人の意図を理解した上で，新しく英語のテキストを作るという方法をとりました．そのため，ご本人が何分もかけて説明したことが1つのセンテンスで表現されていたり，ご本人が簡単に述べられたことが1つのパラグラフになったりしています．そのようなテキストを読んで，さらにベンソン先生に表現を修正していただいたので，実際にインタビューで語られたものと英文のテキストとのあいだには，表面的にはかなりの違いが出ています．

市長のインタビューが終わって最終テキストが完成した6月11日，広島大学医学部の佐々木先生と三島食品の三島社長に，ウォークスルーをお願いしました．お2人とも大変お忙しい方々なので，スケジュールの調整には手間取りましたが，お2人のご協力で，日曜日の夕方にもかかわらず，広島市立大学の大場の研究室で，5時から7時半までの約2時間半にわたって，レビューをしていただきました．佐々木先生からは「エノラゲイに乗り組んでいた人たちの証言を載せられないか?」とのコメントが出され，三島社長からは全体の構成に関するコメントが出されました．両氏とも南京大虐殺の WWW をリンクしていることに関しては，公平感が出るという観点で，好意的な意見でした．この時点で，三島食品から100名前後の社員のアンケート結果をいただけると，三島社長からのご報告がありました．

6月の中旬から下旬にかけて米国へ出張し，成田に到着して自宅へ電話をかけると，「NHK から毎日のように電話がかかってくるので，すぐに NHK へ電話をしてほしい」と

のこと．6月27日にNHKへ電話をすると，「29日の朝のラジオ番組で，電話でインタビューしたい」との申し入れでした．28日に出勤するとすぐに，NHK広島放送局のテレビ･ディレクタから電話で取材の申し入れがあり，その日のうちに研究室へ訪ねて来ました．「インターネットのさまざまな利用方法を紹介したい」というので，アメリカにおけるインターネットのビジネスでの利用方法を説明し，「IIJと相談してマツダの自動車販売促進にWWWを利用するような実験や，市民団体による原爆情報のWWWを利用した提供，学生や小学生による平和教育にインターネットを利用している例などを紹介したらどうか」と提案しました．ディレクタは乗り気で，「すぐに放送局へ帰ってみんなと相談してみます」といって帰りました．

ラジオのインタビューは5分程度の簡単なものでした．企画担当および放送担当のアナウンサと，計2回電話で話をしていたため，生放送であってもそれほどの緊張感はありませんでした．インタビュー担当のアナウンサも慣れていたせいか，5分間を上手に使って，こちらのいいたいことを大体いうことができました．それに比べて，テレビの取材は大変でした．Power Macの画面をビデオに採るために，スキャン･コンバータを実験室へ持ちこんで，モニタ画面とインタビューを別々に録画し，後で編集するという方法を採用していました．インタビューでは，「なぜこのプロジェクトを始めたのか?」「インターネットを使うメリットはどこにあるのか?」「何を一番訴えようとしているのか?」などを聞かれました．

NHKテレビは，地元の広島放送局でまず放送しました．提案したものとは少し違っていました．まず，地元のIIJのオフィスが紹介され，インターネットの波が中国地方にも押し寄せられていることを印象づける画面が流れました．その後，広島市立大学のキャンパスが映し出され，「ここでもインターネットを利用した新しい試みが行われています」とのナレーションに続き，原爆WWW博物館のホームページが大きく映し出されました．そして，原爆WWW博物館の説明が続き，最後に作成者として大場が紹介され，インタビューの一部が流れました．何を目的にやっているかの説明のあと，「インターネットは他のメディアと違って双方向性があるので，原爆のような問題を取り扱うのに向いていると思う」というコメントが流されました．

実は，このコメントには長い前置きがあったのですが，時間の関係で割愛されたようです．その前置きとは，「原爆の問題を米国人よく理解してもらうための最良の手段は出版なのですが，出版にはお金がかかります．このプロジェクトは，ソフトウェア技術者協会と中国四国インターネット協議会のボランティアによるものなので，そのような資金力はありません．そこで，次善の手段として従来の出版によく似たWWWの利用を考えました．WWWなら情報さえ整理すれば，あとはほとんどお金をかけずに，全世界に対して情報を提供できます．また，インターネットには他のメディアと違って...」というものでした．

このテレビが放映されてすぐ，中国四国インターネット協議会のA先生から苦情が来ました．それは，「中国四国インターネット協議会が支援しているにもかかわらず，そのことが一言も触れられていなかった」というものでした．実は，番組で紹介された活動を展開している人たちが，IIJであったり，広島市立大学であったり，安田女子大学の学生であったりで，従来からインターネットを支えて来ていた人たちとは限らなかったことも関係していたのでしょう．問題が大きくなって収拾がつかなくなる前にとの判断から，NHK取材の概要を説明した電子メールを流して，どうしてそうなったかを理解していただきました．

NHKは，同じビデオを関西でも朝の番組で流しました．その後，もう一度同じディレクタから追加取材の申し込みがあり，今度は研究室のワークステーションの前でインタビューを受けました．そのとき，「核実験反対の署名運動をインターネットでやっているから一緒に紹介してくれないか?」とお願いしましたが，興味を示してくれませんでした．この時点になると，アメリカ人やアジア系の人たちからのきびしいコメントが来ていましたので，それについてインタビューで説明しました．また，今度流すときは中国四国インターネット協議会とソフトウェア技術者協会のボランティアによる事業であることを明確にしてくれるようにお願いしました．すると，「でも，実際にやられているのは大場さんでしょう?」と反論して来ました．「入力作業はわれわれでも，取材の段階や情報収集では多くの方々が参加しているのだから，私1人がやっているという印象を与えるのはまずい」というと，「そういう印象を与えないようにします」と約束してくれました．この録画は，8月の3日ごろ，全国放送の朝の番組で放送されました．

その後，すぐにBS放送でやっている英語のニュース番組でも流されました．7月の中頃から8月10日ごろまでは，8月6日の原爆記念日をはさんで，ほとんど毎日のように取材がありました．読売新聞，朝日新聞，雑誌，BBC，南アフリカのFM放送，メキシコのテレビ局，そしてスミソニアン博物館(博物館が出版している雑誌)などでした．特に印象に残ったのは，南アフリカのFM放送局のインタビューでした．このインタビューは，電子メールと電話で行われました．インタビューの質問については，重要な質問がメールで送られて来ました．回答をメールで送り返したのち，電話番号についての問い合わせが来て，それに答えた日の夕方の6時半ごろ，アナウンサから電話が入って，「1時間後に生放送でインタビューしたいが問題はないか?」とたずねられたので，「待っていましょう」と答えました．

放送が始まる10分前ぐらいに，もう一度アナウンサが電話をして来て，「もうすぐ，女性に変わるから，そうしたら本番直前で，始まったらかの女がつなげて」と説明してくれました．放送が始まってから困ったことの1つは，アナウンサの英語が聞き取りにくいことでした．放送のため少しエコーがかかってしまっている上に，英語の訛りが強くて，何を聞かれているのかわからないのです．しかたがないので，「もう一度質問をいってください」と聞き返したことが2度ほどありました．インタビューは長く，10分ほどだったと思います．例によって，「何のために始めたのか?」「どんな人たちが参加しているのか?」「反響はどうか?」「どんな困難があったか?」「フランス核実験再開についてどう思うか?」などの質問を受けました．「どんな困難があったか?」の質問については，日ごろ感じていたことをかなり率直に話しました．この放送の結果かどうかは分かりませんが，原爆WWW博物館への南アフリカからのアクセスは，ドイツからのアクセスと同じくらいでした．

この原爆WWWプロジェクトで重要だったことが，いくつかあります．もちろん最初の段階では，情報の収集と入力です．次の段階で重要だったことは，2人のアメリカ人のレビューでした．アメリカ人の目で見て読めるものであるかどうかは，非常に重要なことでした．英語の問題ではなく，内容の問題です．問題の性質上，反アメリカ的な色彩が出るのはどうしようもありません．しかし，そのためにアメリカの一般市民が反感をもってしまっては，読んでさえもらえません．そこで，2人のアメリカ人を選び，レビューをお願いしました．結果はまずまずでしたが，修正しなければならないことがいくつか出てきました．7月の10日から23日ごろまで，修正作業が続きました．幸いなことに，この間にニューズウィークの特集が出され，修正作業の参考にしました．

最後の段階で非常に重要だったことは，他のホームページとのリンクでした．特に，アメリカで有名なホームページからリンクされることが必要なため，いろいろ研究しました．まず最初に行ったことは，原爆WWW博物館の性格を明確にすることでした．だれのためのものか，何のためのものかを，もう一度考えました．そして，学校の社会科の授業でサブテキストとして使ってもらうことが一番であるとの結論に達しました．そこで，アメリカで社会科のインターネット教材を紹介している3つのホームページの管理者にお願いして，原爆WWW博物館を社会科の教材として紹介してもらうことにしました．現在では，その3つのホームページと，もう1つ大きな教員団体の教材紹介のホームページにリンクされています．「8月5日と6日の2日間にわたって，子どもと原爆WWW博物館を見ました．いろいろ勉強できてよかった」というアメリカからの電子メールに感激しました．

もう1つ忘れられないことがあります．それは，スミソニアン博物館のエノラ·ゲイ展示のホームページとの相互リンクの確立でした．7月21日だったと思います．エノラ·ゲイ展ホームページがあることを知りました．そこで，原爆WWW博物館の『エノラ·ゲイ』からエノラ·ゲイ展のホームページをリンクし，管理責任者宛に電子メールを書きました．「拝啓管理責任者様，まず，われわれのホームページからエノラ·ゲイ展のホームページをリンクしたことを報告いたします．われわれは，広島に住む市民のボランティアで原爆の被害に関する実態を客観的に記録し，世界の人々に知ってもらうために活動しているグループです．残念ながら日本と米国とのあいだには，原爆投下の是非についての社会的なコンセンサスが成立していません．その理由の1つに，正しい情報が知られていない現実があると思います．今後の日米関係を考えると，この問題を放置しておくことはよくないと思います．両国の次世代を担う国民のために，お互いの情報を開示し，参照できるようにすることをここに提案します」．翌日，エノラ·ゲイ展のホームページから原爆WWW博物館がリンクされました!

7月29日ごろから，原爆WWW博物館へのアクセスは急激に増加しました．それにつれて，賛成意見だけでなく，反対意見も寄せられるようになりました．賛成意見の代表的なものは，「これまでリビジョニストが絶えず試み，失敗してきたことを，あなたたちはやってのけた．公平な視点と客観的な記述の成果である」というようなものです．反対意見は，ヨーロッパ系アメリカ人のものとアジア人やアジア系アメリカ人のもので違いがあります．ヨーロッパ系の人は，「もし，原爆が落とされなかったならば，多くのアメリカ人が本土上陸作戦で犠牲になっていた．それを考えれば，原爆投下は一概に悪いとはいえない．私の親戚にもそんな兵隊たちがたくさんいて，喜んでいた」と，主張します．また，「真珠湾攻撃で死んだ私の兄弟のことを考えると，複雑な気持ちだ」といった人もいます．

アジア系の人たちは，もっと攻撃的です．「日本人は，自分たちの都合のよいように自分たちの歴史を修正してしまっている．日本人が，中国でやったこと，朝鮮人に対してやったことをなぜ正直に教えないで，原爆の被害者であることだけを主張するのか? もともと戦争を始めたのは日本人であり，原爆はその結末であったはずだ．この文脈から原爆投下の事実だけを抜き出して，その非人道性を主張するのは片手落ちである．その前に，日本人は他のアジアの国々に対して謝罪すべきである．あなたがたは，日本の兵隊が南京で行った行為について何を知っていますか? それについてどう思っていますか?」というような感じのものです．もう少し婉曲ないい回しのものもありましたが，主旨は同じようなものでした．このようなメールを見ると，われわれ日本人自身の過去に対する姿勢を反省せざるをあ

りません.

海外に住む日本人からもコメントが来ています. それらは英語で書かれていて, 大体「この原爆 WWW 博物館で提示している写真は, 実際の原爆の悲劇に比較すると随分と和らげられており, 被爆の実態を物語ってはいない」というような主旨のものです. これらのコメントに対する, 編集者の回答は,「原爆 WWW 博物館は, アメリカの小中学生にも原爆の威力がどのようなものであり, どのような悲劇を巻き起こしたかを客観的に理解してもらうことを主目的としたもので, ことさら原爆の痛みを示す黒焦げの死体などを提示する必要はないと考えています. また, そのような写真は個人の所有物であり, 掲載には多額の使用料が必要であり, ボランティアによる活動の域を越えています」というものでした.

アメリカ人のコメントの中には, われわれの活動に同情的なものも多くありました. それらは,「記述は客観的であり, 公平性を保とうとしたため, 原爆の是非に関する道徳的な判断を留保している. しかし, この問題についての立場を明確に表明していないことが致命的な問題であろう. このように複雑な問題を客観的に整理して, 完全な道徳的判断を下すことはだれにも不可能なことである. したがって, 判断を留保するのではなく, 自分たちの立場を鮮明にして, 批判を受け付けたほうが結果的に建設的である」というものです. また,「私は, 本を読んで涙するような人間ではない. しかし, 昨日このホームページを見たとき, 涙が止まらなかった. 繰り返してはならない悲劇である. しかし, 個人としての感情を別にすれば, もっと戦争自体が『悪』であることを直接的に訴えるべきでると思う. その点について不満を感じた」という意見もありました.

ホームページに対するコメント以外で興味深かったものは, われわれ (ホームページ管理者) に寄せられてくる各種の質問です. 若い韓国系のアメリカ人から,「平和公園内の記念碑の一つにハングル文字で書かれたものがあるが, どうしてハングルで書かれているのか?」との質問があり, その質問に答えるとすぐに「ところで資料館への入場料はいくらぐらいか?」との質問が来ました. そしてその回答の後,「広島市長の平和宣言を読みました. いままで日本人の政治家がこれだけの歴史観を示したことはなかったと思う. 是非, 市長に感謝状を送りたいので電子メールのアドレスを教えて下さい」という依頼が来ました.

また, ヨーロッパ系アメリカ人の作曲家から,「いま, マンハッタン計画をテーマにしたオペラを作曲中です. オペラの背景として原爆資料館の資料の写真を使いたいのだが, どうすれば手に入るか教えてほしい」という依頼もありました. 別のイリノイに住む人からは,「ナショナル·ジオグラフィクの特集に掲載されているものと似た写真があるが左右が逆になっているのではないか?」との指摘がありました. 平和文化センターに問い合わせると, その通りでした. さらに, その人から「お忙しいとは思うが, 妻が廃墟と化した教会の写真を見て, 宗派を知りたがっているので教えてくれないか?」との依頼が来ました. 教会に電話すると,「日本キリスト教団の所属で特に宗派はありません」との事務的な答えが返って来ました. さらに「明治時代に設立されたときにいた牧師さんはどの宗派の牧師さんだったのですか?」と質問して,「当時は, メソジスト派の牧師がいたそうです」との答えがやっと得られました.

アメリカの学校の先生や学生からや質問が寄せられました. カリフォルニアの大学の先生から,「化学·物理の来年度の授業で, 原爆 WWW 博物館の資料を教材として利用したいので, 許可して欲しい」との要請がありました. バークレーの学生からは,「原爆の歴史について研究したいので, 参考書を紹介して欲しい」との要請がありました. 同じくカリフォルニアの高校生からは,「マンハン計画の機密性についてのレポートを書きたいので教えて下さい. 日本政府は原爆投下前に原爆の存在またはその可能性についてどの程度の情報をもっていたのでしょうか?」この質問については, 個人的に大学時代の指導教官が理化学研究所で仁科博士の部下であったことから, かなり詳細な事情を知らされていたので, 率直に書きました. その高校生は, あとで大変ていねいなお礼のメールをくれました.

フランスの核実験が再開される 1 週間ぐらい前から, ふたたび新聞社 (中国新聞, 朝日新聞, 読売新聞) や NHK の取材が続きました. 新聞社の取材は, 時間に関係なく, また, 休日であるかないかに関係なくあるので, 休みの日に取材が入ることも少なくありません. また, 電話での取材が 1 時間以上になることもよくありました. 一度は, 法事で岡山の親戚に行き, ホテルに帰る途中, 自宅で留守番中の中学生の次女に電話をすると,「すぐに新聞社に電話をして欲しい」とのこと. 公衆電話から新聞社に電話をすると,「長くなるかもしれないので, コレクトコールにしてかけ直して下さい」といわれ, かけ直したら 1 時間ほどの取材になってしまったこともありました. それでもその 1 時間後,「確認したいのですが,....」との電話がホテルに入って来ました. これも広報活動だと割り切る以外にありませんでした. 妻は, 電話の横でいつも「公私混同だ.」と苦情をいっていました. 家族の理解と助手の佐藤さんの協力なしには不可能なことだったと思います. 小学生の長男が「パパはテレビに出られていいな!」と, ただ 1 人積極的に応援してくれたのが救いでした.

ソフトウェア技術者協会
〒160 東京都新宿区四谷3-12 丸正ビル5F
TEL.03-3356-1077　FAX.03-3356-1072