SEAMAIL

Newsletter from Software Engineers Association

Volume 9, Number 8-9-10 August, 1995

目　次

編集部から		1
1995 年度総会報告		2
1994 年秋～ 1995 年夏　主要イベントの記録	編集部	3
オブジェクト指向と ComponentWare	酒匂 寛	10
Domain Engineering	Lloyd G. Williams	24
聞き書き　"グループウェア技術"	林 香	41
インターネット：　もう 1 つの震災時のメディア	中野 秀男	54
広島原爆 WWW プロジェクト	大場 充	56
オブジェクト指向を正しく進化させるために	伊藤 昌夫	58
哲学的暴走･脱線顛末記 Part-II		68
禅とオブジェクト指向	栗原 貴	68
鏡の国のアリサ	伊藤 昌夫	70
オブジェクト指向への密教的接近	青木 淳	74
哀しきスーパーハッカー	青山 幹雄	85
Call for Participations		
IWSED-95		86
イベント一覧 (1994 ～ 1996: 実績と予定)		88

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)：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　伊藤昌夫　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. 9, No. 8－9－10　　1995年8月15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　1，000円（禁無断転載）

編集部から

☆

なんと! 今年になって初めてのSEAMAILです.

☆☆

その間, 阪神大震災やら, 地下鉄サリン事件やら, いろいろなことがありました. SEAのまわりでも, さまざまなイベントが次から次へと計画·実行され. 事務局兼編集部はその準備や後始末に追いかけられ通しで, 時間だけが過ぎて行きます.

☆☆☆

問題は,"プログラマはドキュメンテーションが嫌い"という公式が, どうやらSEA会員諸氏の場合にも成り立つらしく, それらのイベントからのアウトプットが, いつまで待っても編集部に届かないことです.

☆☆☆☆

さすがにこの春頃から, SEAオフィスの電話に"SEAMAILまだですか?"という問い合わせが, ちらほらかかってくるようになりました. "原稿が足りないものですから, まだ出ません. あなたも何か書いていただけませんか?"という定番の切り返しで応答すると, ほとんどの場合"ムニャムニャ..."で切れてしまうのはなぜでしょう?

☆☆☆☆☆

SEAの会費はSEAMAIL購読料ではなく, 原稿掲載権料(しかもページ数制限なし)だということをお忘れなく.

☆☆☆☆☆☆

ともかく, 数少ない投稿とイベント資料を寄せ集めて,「ごぶさた&暑中お見舞号」を編集しました. 次号は, 早ければ9月に, 去年の若手の会レポート特集の予定です.

☆☆☆☆☆☆☆

最後に. 悲しいお知らせ. SEA幹事会メンバの山崎朝昭さん(新日鉄情報通信システム)がなくなられました. 昨年秋に入院·手術され, その後経過は順調というe-mailをいただいていたのですが.... 残念!

☆☆☆☆☆☆☆☆

1995 年度総会報告

今年度 SEA 年次総会は，さる 5 月 25 日 (木)，5 月の Seminar & Forum の後，機械振興会館で開催され，新年度役員の選出，年度予決算の報告·承認が行なわれました．

1. 新年度役員

新年度の幹事会メンバーは次の通りです．

代表幹事は中野先生から山崎利治さんに変わりました．

昨年度の幹事のうち，臼井義美 (日本電子計算)，江谷典子 (富士通)，小林俊明 (コンピュータ·コンサルティング)，鳥居宏次 (NAIST)，野村敏次 (日本電子計算)，平山伸一 (さくら KCS)，渡辺雄一 (アスキー) の各氏は，今回退任されることになりました，どうもお疲れさまでした．

代わりに，荒木啓二郎 (NAIST)，井川裕基 (日本電子計算)，塩谷和範 (SRA)，野呂昌満 (南山大)，盛田政敏 (さくら KCS) の各氏が新しく幹事会メンバに加わりました．よろしく．

代表幹事
山崎 利治 (フリー)

常任幹事
大場 充 (広島市大)
熊谷 章 (PFU)
坂本 啓司 (オムロン)
中野 秀男 (大阪市大)
深瀬 弘恭 (IIJ)

幹事
青山 幹雄 (新潟工大)
荒木 啓二郎 (NAIST)
井川 裕基 (日本電子計算)
市川 寛 (電算)
伊藤 昌夫 (MHI エアロスペース)
菊地 俊彰 (東北電力)
君島 浩 (富士通ラーニングメディア)
窪田 芳夫 (東京電力)
酒匂 寛 (SRA)
塩谷 和範 (SRA)
篠崎 直二郎 (NEC ソフトウェア)
杉田 義明 (日本 NCD)
高橋 光裕 (電力中央研究所)
武田 淳男 (安川電機)
田中 一夫 (山一情報システム)
玉井 哲雄 (東京大)
中来田 秀樹 (ネクストファウンデーション)
中谷 多哉子 (フリー)
西武 進 (SES)
野中 哲 (アップル テクノロジー)
野村 行憲 (岩手電子計算センター)
野呂 昌満 (南山大)
平尾 一浩 (IIJ 九州)
二木 厚吉 (JAIST)
堀江 進 (NEC ソフトウェア)
増井 和也 (東芝アドバンストシステム)
松原 友夫 (Office Peopleware)
松本 理恵 (菱信システム)
盛田 政敏 (さくら KCS)

会計監事
辻 淳二 (辻システム計画事務所)
吉村 成弘 (公認会士)

事務局長
岸田 孝一 (SRA)

2. 決算および予算

(1) 1994 年度収支

支出の部		収入の部	
人件費	8,210	新入会費	616,000
事務所費	2,794,387	更新会費	4,886,000
印刷費	4,608,562	賛助会費	4,800,000
通信費	2,747,992	イベント収入	1,736,861
会議費	283,273	雑収入	320,198
支部支援費	377,740		
国際活動費	10,000		
消耗品費	127,837		
雑費	14,395		
当期収支差額	1,386,663		
合計	12,359,059	合計	12,359,059

(2) 1995 年度予算

支出の部		収入の部	
人件費	120,000	新入会費	1,056,000
事務所費	3,000,000	更新会費	4,800,000
印刷費	4,560,000	賛助会費	4,500,000
通信費	3,600,000	EVENT 収入	1,800,000
会議費	300,000	雑収入	360,000
支部支援費	420,000		
国際活動費	240,000		
消耗品費	120,000		
雑費	18,000		
当期収支差額	138,000		
合計	12,516,000	合計	12,516,000

去年会費を値上げし，各種経費を切り詰めましたのでなんとか黒字を維持していますが，依然として緊縮財政です．

1994 年秋～1995 年夏　主要イベントの記録

事務局

去年の秋からこれまでに行なわれた SEA イベントのうち，事務局に情報が残っているものについて，簡単に概要をまとめてみました．

1. 全国縦断 Forum on ComponentWare

この連続 Forum のきっかけは，昨年 5 月 25 日の SEA 総会にあわせて行なわれた月例 Forum "New Metaphor はソフトウェア開発を変革しうるか ?"でした．

単なる座学ではなく,"オブジェクト指向" の大衆化を狙った (?) 新しいツール群を一堂に集めて，プレゼンテーションやデモを行ない，最後をパネルで締めるという形式の Forum は，内容的になかなか面白いし，一度だけで終らせるのはもったいないというので，各ツール･ベンダの御協力を得て，9 月の横浜を皮きりに，北九州，仙台，名古屋，(再び) 東京，琵琶湖 (SS95 の Tutorial，展示，BOF)，松本と巡回してきました．

開催時期	場所	参加人数	備考
95/5/25	東京	29	
9/30	横浜	25	
12/2	北九州	32	
95/2/8	仙台	80	共催:TISA
4/14	名古屋	60	
5/25	東京	50	
6/13	琵琶湖	23	Tutorial
7/20	松本	25	

今後さらに，広島 (9 月末)，九州，関西，北海道，北陸 etc での開催が予定 (企画) されています．

いずれ，コーディネータの中来田さんからまとまったレポートが出るものと期待していますが (?)，この号には，Forum の基調講演者であり，SS95 では Tutorial の講師をつとめられた酒匂寛さんの解説論文 "オブジェクト指向と ComponentWare" を収録しました (pp.10 ～ 23)．

2. 第 12 回若手の会 (9/21 ～ 24)

テーマは CSCW．参加者は 36 人でした．レポートは，いま，実行委員長の筏井美枝子さん (SRA) と熊谷章さん (PFU) が取りまとめ中．

今年の若手の会については，まだ何もアナウンスが出ていませんが，これは,「そろそろこのワークショップの企画･運営がマンネリ化してきた．一度，若手の人たち自身に最初のプランニングから運営までをまかせて，モデル･チェンジを考えよう!」ということが幹事会で決まり，とりあえず 20 代の会員のうちの希望者 10 数名で mailing list を作って，いま，討論が始まったところです．

3. 10 月の Forum (10/18)

テーマは,"障害者と計算機＋ネットワーク"．SEA としては初めての試みでした．参加者は 36 人．障害者の方々も何人か参加されましたが，時間の関係で，それらの方々の肉声が聞けなかったのは残念でした．

取材を兼ねて参加されたフリーライターの広重隆樹さんのレポートを，パネリストの 1 人だった竹中ナミさんがやっておられるボランティア団体プロップステーションの機関誌から転載します．

SEA Forum に参加して

広重 隆樹

障害者の自立のためにコンピュータを活用しよう－　そうしたプロップの考え方に惹かれ，その活動を外野から見守ってきた．障害者のコミュニケーション手段としてのコンピュータや OA 機器にも少しばかり関心がある．かつて障害者雇用の取材の現場で職場との連絡に FAX をフル活用する聴覚障害のオフィスワーカーに出会った．ときどきアクセスするパソコン通信の朝日ネットで，誤字の一つもないメッセージをアップした人が，実は全盲の障害者だったと聞いて驚いたこともある．

コンピュータや通信が障害者の社会参加を広げるツールになっていることを実感する．けれども，そうした環境を整備するメーカーや技術者の視野に，障害者のことがどの程度見えているのか，それが以前から疑問だった．

そんな折り，竹中さんに誘われて，東京で開かれた

「SEA Forum October94·障害者と計算機＋ネットワーク」(94年10月18日·後楽園会館)に出かけることになった．不勉強で，実はSEAのことはこの会合に出るまで知らなかった．ソフトハウスや大学，研究所などにいるソフト技術者が，組織の壁を越えて経験や技術を交流し合うための場で，設立は9年前に遡るという．

代表幹事の中野秀男先生の司会で，まず日本障害者雇用促進協会の丹直利さんが，「障害者とコンピュータ·ネットワーク」と題して，障害者のパソコンやパソコン通信利用の現状と課題について講演した．特に僕が興味を惹かれたのは，マルチメディアと障害者というテーマだ．丹さんは，「今のマルチメディアは『目が見える』ことを前提に動いているような気がする．産業優先でかけ声ばかりが先行しているが，今の段階で障害者の利用という視点を組み込んでいかないと，盲聾者が情報から取り残されることになる」と指摘した．

僕も最近，ダイヤルアップでインターネットに繋ぐことができるようになり，WWWサーバーにアクセスしては「うーむ，面白い！」とかウナっているのだが，その時はすっかり視覚障害者のことは忘れている．マルチメディアは「感覚の組み換え」(西垣通氏の言葉)を促す新しいメディアだとされる．けれどもそこでいわれる「五感」とは，やはり健常者のものだけではないのか……．

そんなことをツラツラ考えていると，いきなりナミねぇの大声で目を覚まされた．

竹中さんの演題は「身障者の就労を支援するボランティア団体の活動」．つまりプロップの活動の紹介なのだが，桜井さんの就労をめぐる大阪府との交渉，朝日新聞へのアドバイスの話など，「ぶっちゃけ話」を含めてかなり具体的なものだった．彼女が強調したのは，「これまで障害者関係の集まりというと，いわゆる福祉系の人ばかり．技術者と交流することで，新しいフェイズが生まれる」ということである．

後半は，プロップメンバーで日本DECに就職した安東直哉さんを交えてのパネル·ディスカッション．中野先生が上手にまとめていたけれども，議論のテーマは「障害者にとって便利なコンピュータとは」「障害者のコンピュータ利用をめぐって官庁，企業，ボランティアの役割とは」「ネットワーク社会と障害者の問題」「障害者の就労の仕組みをどう変えるか」 — そんなところに絞られた．

安東さんは，使いやすいマウスやトラックボールを探し求めて日本橋や秋葉原を歩いた経験を披露してくれた．ナミねぇが「障害者向けのコンピュータ開発のために国の助成は少なすぎる．フリーウェアの開発などは助成の対象にならない」と指摘すると，中野先生や丹さんも「技術者，研究者同士の横のつながりが足りない」と課題をあげていた．会場の技術者からも「まだまだソフト技術者と障害者の接点が見えない」という声があがったりした．

けれども，今回のフォーラムのように，障害者と技術者が共通のテーマで語り合うのは初めての試みだ．こうした経験の積み重ねが，いつかは企業の開発現場に反映され，行政をもつき動かすことになるだろう．その萌芽が生まれつつある，と思った．

なお，SEA関西支部には，その後，プロップステーションとのジョイントで"チャレンジド分科会"が結成され，障害者とコンピュータとの関わり合いについて，いろいろな角度からの討論が続けられています．

4. 全国縦断 Forum in 広島 (10/27)

常任幹事の大場充さんがIBMから新設の広島市大に移られたのをきっかけに結成されたSEA広島支部最初の公開イベント．"情報産業ニューウェーブ"をテーマに開催．次のように盛り沢山なプログラムでした．参加者は54人．

ネットワークの発展と情報産業
　西武 進 (SES)
マルチメディア技術のソフト開発への応用
　長野 宏宣 (NTT)
新技術の導入と組織学習 (プロセスの視点から)
　坂本 啓司 (オムロン)
オブジェクト指向技術は発展するか?
　藤野 晃延 (FXIS)
Panel 討論ソフトウェア技術者と新技術
　司会： 大場 充 (広島市大)
Panelist: 上記講演者ほか

SEA広島支部発行のネットワーク·ニュース (1994年11月30日付第3号) から，大場充先生 (広島市立大学) によるレポートを転載します．

去る10月27日午前10時から，西区ミクシスビル内1階研修室でSEA広島フォーラムが開催され，日本全国から約60名の方々が参加されました．このフォーラムでは，情報産業における技術の潮流と技術者の自己研鑽をテーマに，4つのセミナーとパネル討論が行

われました.

情報分野での技術は日進月歩, 数年前の最先端技術が今では石器時代の技術であるかのように語られるのが, この世界です. そのような技術の陳腐化の早い状況の中で, われわれ技術者は, 技術をどう考え, どう学び, どう育ててゆくべきかの哲学を, 日々の仕事から離れた環境で冷静に考えてみようというのが, このフォーラム企画の趣旨でした. 首都圏の SEA の会員の多大なご支援もあり, 地元の技術者を中心に関西, 九州, 四国, 北海道から, 全部で 54 名もの参加者がありました (講演者を除く). 会員 1,000 円, 非会員 3,000 円という安い参加費を設定したため, 最低でも 40 人の参加者がないと会場費と電話仮設料などで赤字決算という背水の陣で望みましたが, コンピュータ·メーカや大規模ユーザのご支援で, 何とか黒字決算にできました. SEA 本体からも一部資金援助をいただきました.

特に, セミナーの講師やパネリストをお引き受けいただいた方々には, 手弁当でのご講義ということでご迷惑をおかけしました. 講師のほとんどが, 首都圏に在住の方々で, しかも各種の有料セミナーの講師ばかりにもかかわらず, 交通費も自前でご負担いただきありがとうございました. 特に, 藤野さんには, 事務局の不手際で交通の便に関する情報が不備だったために, 航空運賃と同じくらいのタクシー代を支払っていただくという結果になってしまいました. 申しわけありませんでした.

前半のセミナーでは, インターネットを取り巻く話題 (中国四国インターネット協議会事務局·西武氏), これからのネットワークを利用した分散ソフトウェア開発の話題 (NTT ソフトウェア研究所·長野氏), 新技術導入やプロセス改善に関する戦略的な取り組みの話題 (オムロン EFTS 統轄本部·坂本氏), オブジェクト指向技術の実践に関する話題 (富士ゼロックス情報システム·藤野氏) を講師として, 最先端の技術の動向とそれらの技術に関する問題点について学びました. このような著名な方々のご講演を一度に拝聴できる機会は, 東京や関西でもまずないということで, 遠く九州, 関西, 北海道からも何人かずつの参加者がありました. 関西からは, 奈良先端科学技術大学院大学から 3 名の大学院生が一晩自動車を走らせて駆けつけるなど, 主催者としては成功を実感しました.

フォーラム後半のパネル討論 (司会: 大場) では, 主として産業界で活躍している管理者の方々から, 技術者と技術の問題について議論していただきました. パネリストとしては, 地元の産業を代表してマツダ情報システム本部の吉永氏, ソフトウェアの標準化活動で世界的に活躍している NTT ソフトウェア研究所長野氏, キャノンで新しく創造環境センターを作り新しい技術への戦略的な対応を模索している浅野氏, ピープルウェアなどのソフトウェア工学関係の名著の翻訳で活躍しているコンサルタントの松原氏などの 4 氏の外に, SEA 事務局 (SRA) の岸田氏も飛び入りで参加されました.

5 人のパネリストのプレゼンに続く討論では, 岡山理科大の大西先生が「ソフトウェア危機は嘘だったのか?」との問題提起をしたり,「インターネットは日本のような階層社会に矛盾しないのか?」や「地域として広島は遅れているといわれているが, そこで仕事をしている技術者はどうしていけばいいのか?」などの問題が議論されました. また, 広島大学医学部の佐々木先生から,「もっと異業種間交流をしよう!」との提言もあり, 大変盛り上がりました.

---

5. 第 2 回 IWSED (11/7 ～ 9)

昨年秋の京阪プラザに続く第 2 回目の "ソフトウェア工学データに関する国際ワークショップ" でした. オーガナイザとして御苦労された大場先生のレポートを, やはり SEA 広島ニュースから.

去る 11 月 7 日から 9 日の 3 日間, 広島市内厚生年金会館と県情報プラザを会場に, 第 2 回目の "ソフトウェア工学データに関する国際ワークショップ" が開催され, 内外から計 35 名の研究者が参加しました (SEA 広島からは大場). SEA 関係者としては, 事務局の岸田氏 (SEA), 幹事の鳥居教授 (奈良先端科学技術大学院大学), 久保氏 (富士通), 坂本氏 (オムロン), 高橋氏 (電力中央研究所) ほか, その他の日本からの参加者は, 主として首都圏にある企業 (日立, 富士通, NEC, オムロン, 東芝, SRA, シャープ, 電力中央研究所, 日本 IBM など) のソフトウェア技術者者と, 主催者である広島市立大学と (株)「けいはんな」からの代表者でした.

このワークショップは, 米国メリーランド大学のバシリ教授と奈良先端科学技術大学院大学の鳥居教授の呼びかけで, 昨年から日本で開催されているものです. ソフトウェアを開発·保守している多くの組織 (情報シ

ステム関係組織) では，計画や管理のためにさまざまなデータを収集し，分析しています．しかし，どのようなデータを収集し，どう分析するかは，個々の組織のノウハウとなっており，経験が共有されていないのが現状です．

そのような現状を改善し，他の組織で経験されたことを多くの組織で共有するための枠組みを構築しようというのが，このワークショップ開催の趣旨です．そのためには，単なる数字としてのデータの交換だけでなく，その定義や意味，どのような状況で，どのようにして収集されたか，対象となった組織の大きさや特徴，プロジェクトを取り巻く状況や構成員の特徴などに関する文脈データの収集が重要になります．

たとえば，一言で「100行当たり2個のエラーだった」といっても，その意味することは，何人ぐらいのプロジェクトで，だれが，どの段階で，どのようにしてエラーを発見し，どのような手順で記録し，1行や1個のエラーとは何を指しているかによって，変わってきます．そのような解釈上の混乱を避けるため，計測データとともにどのような文脈データを記録すれば，計測データにもとづく経験の有効な共有と蓄積が可能になるかを模索したのが，昨年のワークショップでした．

昨年は，日本の企業からの25名の参加者を中心に，米国，ドイツ，オーストラリアからの12名の研究者や管理者が会議に参加し，現在どのようなデータが収集され，どのように活用されているかについて発表したのち，4つの分科会に別れて，各国，各企業間の共通点や違いについて突っ込んだ議論をたたわせました．その結果，文脈データを記述し，記録する手段としての質問項目が整理され，調査表の形式でまとめられました．

本年度は，7日の9時から広島市立大学の田中隆荘学長による歓迎のあいさつのがあったあと，実際に調査表を用いて，実験的に日本，米国，オーストラリア，イタリアの各国のソフトウェア開発組織の実態について，各国を代表した企業の代表者による発表がありました．8日の午前中は，イタリア，日本，米国に関する国別の実態調査報告が，バリ大学ビサージオ教授，広島市立大学大場教授，メリーランド大学バシリ教授の3名によって行われました．2日目の午後からは各国のソフトウェア·プロセスの実態，収集されているデータと収集方法，そして3日目に収集したデータを何のためにどう活用しているかの実態について，各国におけるプラクティスの違いを議論しました．数値データの比較だけではわからなかった事実が，この議論を通じて明確になりました．

特に議論が紛糾したのは，日本ではソフトウェアの開発でも協力会社をよく利用している点です．日本では，大体の仕様がかたまると，ソフトウェアの実現とテストは，多くの場合協力会社に発注されます．このとき，その大部分は資本関係のある子会社に発注されるのが一般的です．外国では，特定の会社間の関係が強くなす過ぎないように，法的に規制されている場合もあり，一般に，このようなことは行なわれていません．そのため，外国の企業の研究者や大学の研究者にとっては，日本のソフトウェア開発の実態がわかりにくいようです．いわゆる『子会社』とか『下請け』とかの概念になじみがないのでしょう．

逆に，日本からの参加者にとってわかりにくかったのは，米国では一般的な，ソフトウェア開発組織のプロフィット·センタとしての運営です．日本ではほとんどの場合，ソフトウェア開発組織はコスト·センタとして位置づけられており，与えられた予算内で運営され，厳密な意味での採算の責任はもっていません．このことは，一部のアメリカ人を驚かせたようですが，もっと驚いたのは日本人でした．どんな組織もできるだけプロフィット·センタとして位置づけ，独立採算で運営しようとするアメリカ的な経営の競争力の源泉を見せつけられたような気がしました．いかに開発コストを下げて国際競争力を維持するかの執念のようなものを感じました．一言でいえば，狭い意味での品質を追及する日本と，不断のコスト低減によって国際競争力を維持しようとする米国との対立だといえましょう．

---

6. 全国縦断 Forum in 京都 (11/10-11)

その週前半に広島で開かれた第2回 IWSED に来日された V.Basili 先生 (Maryland 大学) 他を招いて "プロセスとメトリクス"をテーマに，オムロン本社で開催．参加者 56 人．

7. KICS'94 (11/16-19)

Domain Engineering をテーマに雲南省·昆明で開催．今回から，SEA 単独主催でなく，国連大学·国際ソフトウェア技術研究所 (UNU/IIST) との共催になり，参加者も，中国 (40 人)，日本 (20 人) の他に，マカオ，インドネシア，アメリカ各2人，韓国·ベトナム各1人と，国際色が強くなってきました．

Domain Engineering の背景と現状を手際よくまとめられた Lloyd Williams さんの基調講演資料をこの号に収録しました (pp.24 ～ 40). 編集部で日本語に訳す予定だったのですが, なかなか時間がとれないので, とりあえず原文をのせます. 日本語版はいずれ 3 月 Forum (後述) のレポートと一緒に (!?).

もう 1 つの基調講演は国連大学 (UNU/IIST) の Bjorner 先生. SEAMAIL Vol.8, No.10 で紹介した PRaCoSy プロジェクト (中国の鉄道制御システムを Formal Spec で整理しなおすという壮大な計画) のその後の進展状況の報告.

また, 会議初日のランチョン･スピーチには, 作家の張承志さんをお招きし, 御自身のモンゴルでの生活体験にもとづく興味深いお話を伺いました."こういう話が聞けるのがこのシンポジウムのいいところだ!" というのが, Bjorner 先生の御感想. 張承志さんの主要な作品はほとんど日本語に訳されています. また, 御自分の文革体験をまとめた著書ー紅衛兵の時代 (岩波新書), モンゴル草原の遊牧生活 (朝日選書), 回教から見た中国 (中公新書) ーもおもしろい.

Paper Session は以下の通りでした (一部 Double Track)：

- OO and Domain (Paper 4 篇)
- CASE (Paper 4 篇)
- User Interface (Paper 4 篇)
- OO and Concurrency (Paper 4 篇)
- Artificial Intelligence (Paper 4 篇)
- Managing New Technology (Paper 3 篇)
- Domain Analysis (Paper 4 篇)
- MIS and DSS (Paper 4 篇)
- Process (Paper 4 篇)
- Environment Architecture (Paper 4 篇)
- Formal Metod (Paper 4 篇)
- CLosing Panel (DE をどう活用するか?)

次回は, 今年の 10 月末, 今度は故･毛沢東主席の生まれ故郷である湖南省･長沙での開催を予定しています (CICS'95). 今年は, 新たにフィンランドおよびオーストラリアからも論文応募がありました. プログラム委員会は, ちょうどこの Seamail が印刷･製本されている 8 月上旬開催. 9 月には, Advance Program の発表および一般参加募集のご案内ができるでしょう.

8. 第 15 回ソフトウェア信頼性シンポジウム (11/15 ～ 16)

今回は, 千里ライフサイエンスセンターで開催. 不況の影響か, いつもより少なかったが, それでも参加者 98 人という盛況でした. オープニングの基調講演は中野先生の "インターネットとセキュリティ". チュトリアルは "ソフトウェア開発におけるヒューマンファクタ". 論文発表は 4 セッション (プロジェクト管理, モデル, 信頼性評価技術, テスト/レビュー), 他に電子情報通信学会 FTS 研究会の論文発表 2 セッション.

論文集 (頒価 2000 円) は, まだ若干在庫が残っていますので, 御希望の方は事務局まで.

9. 12 月の Forum (12/20)

若手の会の PC メンバーを中心に,"CSCW, グループウェア, そして ....?"をテーマに, いつもの後楽園会館で開催. 参加者 30 人.

司会をされた熊谷さん (PFU) 曰く：「たいへんおもしろかった. "CSCW とは何か?" (単なるキャッチフレーズではなく, 技術的に何があるのか?" についての議論が中途半端に終ったので, いずれ機会を見て, もう一度このテーマを取り上げたい. また, 今回の議論については SEAMAIL にレポートしたい」

しかし, 例によって, 原稿はまだ届いていません. たまたま同じテーマに関連して, コロラドの林香さん (SRA, Boulder) から, セミナー･レポートが送られてきましたので, pp.41 ～ 53 に載せました.

10. 阪神･淡路大震災 (1/17)

まだ記憶に新しいところです. 中野先生が朝日新聞に寄稿された記事の元原稿を再録します (pp.54 ～ 55).

なお, この文章の中で触れられているロス地震の時に作られたニュースグループ "current-events.la-earthquake" のすべての交信記録, および今回の震災時の "fj.misc" での交信記録は, IPA 技術センターの鈴木裕信さんがすべてアーカイブしてくれています.

もしこのデータに対して何らかの分析を試みたいという方がおられたら, hironobu@stc.ipa.go.jp 宛に mail でお問い合わせください.

11. 1 月の Forum (1/23)

例年通り, 教育分科会 (SIGEDU) メンバを中心のForum . テーマは,"マルチメディアビジネスと教育を考える". 会場はいつもの後楽園会館. 参加者は 31 人.

参加者の 1 人である杉田義明さん (NCD) の感想は,「時流に乗った話題であり, それなりに人が集まり, 議論も白熱したが, ビジネスのイメージはそれほどはっきりとは見えなかった!」ということでした.

12. 2 月の Forum (2/20)

すでに何回かやっているインターネット最新状況についての Forum. 今回のテーマは,"インターネットのソフトウェア開発への利用". 会場は後楽園会館. 参加者は予想通り多く 56 人 (しかし, 申込んでいて当日不参加も 10 人!?).

インターネットの利用がしだいに現実のものになってきたせいか, セキュリティまわりの質問が多かった. しかし, 深瀬弘恭さん (IIJ) にいわせれば,「インターネットの活用が一番遅れているのは, 悲しいことに, ソフトウェア業界!」だそうです.

13. 3 月の Forum (3/22)

別件で来日された去年の KICS'94 の基調講演者 Lloyd Williams さんを迎えての 1 日の Seminar & Forum. 会場は後楽園会館. 参加者 29 人. 一応記録はテープにとってあるので, そのうちにレポートできるでしょう.

ところで, ご存知のように, この Forum の 2 日前の 3/20 には, 地下鉄サリン事件発生. もしあれが Forum 当日だったら, スピーカ, 参加者のうち何人かは巻き込まれていただろうと考えると恐ろしい! 3/21 が休日 (春分の日) だったので, 事務局は事件当日の夜8 時ごろ会場へ配布資料を預けにいったのですが, その帰途, 地下鉄後楽園駅に向かって歩きながらふと上を見ると, 崖のうえの地下鉄車両基地に, あかあかと電気を灯し窓もドアも開け放した車両が停まっていて, ガスマスクをつけた兵士が大勢, モップで残留物を洗い流しているところ. まるで SF 映画の 1 場面のように不気味な光景でした. こちらがわの歩道橋の下には, ホームレスのおじさんが 2 ～ 3 人, 新聞紙の上に座り込んで, ワンカップ片手に見物していたのも, 実にアンバランス! 現代の不条理を絵に描いたようでした.

14. 第 7 回 テクニカルマネジメント WS(4/6 ～ 8)

今年は, 申込みをしたにもかかわらず, 公私さまざまな理由で参加できなかった人が何人か出て, 結局 9 人という小規模なワークショップになりました. ただし, 参加した人たちによれば, 開催場所 (五島列島·福江島) は, "これまでで最高!" だったそうです.

世話役の田中一夫さんが, 手回しよく, 来年の分も予約をしてきたそうなので, テクニカル·マネジメントに関心をお持ちの方々, 仕事のスケジュールをやりくりして, 来年 4 月中旬には時間を空けておきましょう!

15. ICSE 併設ワークショップ (4/23)

"Seattle はアジアからアクセスしやすいから, ICSE に引っかけて何かイベントを企画しないか?" という誘いが実行委員会からあり, "それでは!" と片肌をぬいで "Asian Approaches to Software Engineering" と題した 1 日のワークショップを企画しました. 参加者は 30 人 (日本: 20, アメリカ: 5, 中国: 2, 香港: 2, オーストラリア: 1).

プログラムは以下の通り. ハードな 1 日でした:

08:00 Continental Breakfast

09:00 Keynote Presentations

- L.A.Belady(MERL, USA): "Software Research with a Japanese Touch"
- L.Duvall(Kaman Science, USA): "A Sociological Look at Software Management in USA and Japan"
- T.Matsubara(Office Peopleware, Japan): "Software Process from Japanese Perspective"
- N.Nakakoji(Colorado Univ, USA): "The Need and Support for a Cross Cultural Perspective of Software Engineering"
- A.Kumagai(PFU, Japan): "Conceptual Models and Their Cultural Backgrounds"
- M.Noro(Nanzan Univ, Japan): "Is OO Software Development Is a Japanese Way of Doing It?"
- K.Reed(La Trobe Univ, Australia): "Government Policy Setting for Software Engineering Research for Emerging Economics in the Pacific Rim"
- J.Verner(City Univ of HongKong, HongKong): "Status of Software Engineering Practice in HongKong"

12:00 Lunch Break

13:00 Working Group Discussion

- Group-1: Technology Issues
- Group-2: Management Issues

18:00 Wrap-Up Session with Dinner

20:00 Closing

なお, 次回の ICSE は来年 3 月末, ドイツ (ベルリン) での開催が予定されていますが, 引続き同じようなワークショップを企画しないかという誘いが来ています. いずれ幹事会で検討することになるでしょう.

16. 原爆博物館 on WWW

年初めに入院·大手術をされた大場先生 (広島市大) です

が，退院後さっそくアクティブに活動を開始されました．その1つが，WWW上での原爆ミュージアムの開設です．いずれ詳しい報告は御本人からあると思いますが，とりあえず，プロジェクト開始にいたる経緯を説明したSEA広島ニュース1995年5月25日付(第7号)の記事を転載しておきます(pp.56 ～ 57)．

17. Software Symposium 95 (6/13 ～ 16)

SEAとしては年間最大ののビッグ･イベントであるソフトウェア･シンポジウム95(@ ラフォーレ琵琶湖)は，無事に終了しました．参加者は160人(併設チュトリアルは6講座合わせて延べ80人強)，当初目標の200人には到達しませんでしたが，採算上はまずまずの数字になりました．160人のうち，SEA正会員は52人，賛助会員は47人でした．

シンポジウムのプロシーディングスはすでにお手元にとどいているはずです．琵琶湖に行かれた方々には重複しますが，どなたか知人/友人に差し上げて，SEAのPRに使ってください．

来年のSS96は広島です．会場は平和公園の国際会議場．プロシーディングスの巻末にCall for Paperがついています(若干誤植がありますが:-)．最優秀論文賞(賞金20万円)を目指して，いまから論文応募の準備をはじめてください．ちなみに，今年の授賞論文は，青山幹雄さんの"Life with Process: プロセス支援環境PRIMEの実践から"でした．

また，来年からは，これまで東京で開いていた年次総会を，SSの会場で一緒にやるようになると思います．論文を出さない(書くひまがない)方も，ぜひ広島行きをいまから計画に入れておいて下さい．

18. SEA Classic Seminar

コンピュータが世界に登場してから半世紀近い時間が経過し，専門分野としてのコンピュータ･サイエンスやソフトウェア･エンジニアリングも，しだいに確立され整備されつつあります．しかしながら，そこでの研究開発の成果の多くは，まだ開発や保守の現場で充分に活用されるにはいたっていません．

この春以降，SEA幹事会では，そうしたソフトウェア分野における"古典的な"論文や書物を若い世代にきちんと紹介するための新しいセミナー企画について，mailing-list上でのやりとりも含めて，討論を繰り返して来ました．そして，新たに代表幹事になられた山崎利治さんの提案により，とりあえず最初の試みとして，すでに30年近くの歴史を持つACM/Turing賞の授賞者(授賞論文)を題材として，幹事会メンバーの何人かが，それぞれ自分好みの人物とその業績について語るという形式の連続セミナー･シリーズを，9月からスタートすることが決まりました．

つまり，それぞれの授賞論文がソフトウェア技術の発展にどのような影響を与えたか，そしてまた，これからのソフトウェア開発にとってどのような意味を持つかを，それぞれの講師が解説して行くわけですが，それは，同時にまた，話し手自身のソフトウェア技術論を語ることにつながります．

最初のシリーズとしては，フォーマル･アプローチおよびアルゴリズム理論の分野から6人のコンピュータ･サイエンティストを取り上げて論じようということになりました．

開催日	講師	対象(授賞年度)
9月8日(金)	山崎 利治 (Free)	E.W.Dijkstra (1972)
10月13日(金)	中野 秀男 (大阪市大)	R.M.Karp (1985)
11月10日(金)	玉井 哲雄 (東京大)	D.E.Knuth (1974)
12月12日(火)	野呂 昌満 (南山大)	N.Wirth (1984)
1月12日(金)	二木 厚吉 (JAIST)	C.A.R.Hoare (1980)
2月8日(木)	青山 幹雄 (新潟工大)	R.Milner (1991)

会場は，日本青年会議所会館(東京･千代田区平河町)，時間は毎回14:00 ～ 17:00．セミナー終了後，講師を囲んでの夜の懇談会も予定されています．

定員30人限定で，6回通しの予約受講者を募集したところ，申し込み〆切期限より2週間も早く，満員札止めとなってしまいました．

柳の下の2匹目のドジョウを狙って，類似企画の第2弾を計画するかどうか，いま幹事会で討議中です．また，関西他のいくつかの地方支部でも，開催を検討中です．

オブジェクト指向とComponentWare

酒匂 寛
sakoh@sra.co.jp
(株) SRA 技術部システム技術グループ

概要

この小論の目的はオブジェクト指向の概念を復習しつつ、新しいソフトウェア開発環境であるComponentWare について解説を行うことである。ComponentWare はインスタンスベースのオブジェクト指向開発環境の総称であり、現在主流のクラスベースのオブジェクト指向開発環境を補完するものである。解説にあたっては、オブジェクト指向そのものを様々な角度から眺めながら、従来の手法と ComponentWare の位置付けの違いを順次述べて行く事にする。なお解説のための実例としてはSmalltalk 上に構築された PARTSWorkbench という 製品を中心に用いる。

□ オブジェクト指向小史

ソフトウェア構築技術を導いてきた動機の一つに「モジュール化」と呼ばれる概念がある。モジュール化の原理は極めて単純で明快である。人間が一度に考えることのできるシステムを一つの大きな塊として捉えるのではなく、システムをより小さなサブシステム群に、それぞれのサブシステムは更に小さなサブサブシステム群に、という具合に分割を繰り返し行い、それぞれのサブシステムを他との境界がはっきりした「モジュール」として定義する。こうすることによって、人間が一度に考えなければならない部分の大きさが小さくなり、その部分を作成するにも、保守するにも有利になる。チームで作業する場合にはモジュール化は必須である。利用者の要求が高度になり、成長する複雑なシステム構築が人間の手に余るようになるにつれ、自然と高度なモジュール化の研究と実践も盛んになった。

概念的には単純なモジュール化だが、その実現となると何に着目して分割を行うかによってずいぶんと方式が異なっている。モジュールへ分割を行おうと考えるとき、大きくわけて二つの視点が存在している。それは「データからの視点」と「機能からの視点」である。計算機はその生い立ちから、「計算するもの」と思われていたので、システムの分割も機能的な側面から進んだ。この努力はサブルーチンとしてほとんどの言語に見ることができる。こうしていわゆる手続き型言語は「順次実行される固定計算木」として表現されるようになった。

しかしながら、計算機の用途は単なる計算だけではない。「情報」が取り扱いの対象になるにつれ、より複雑なデータ構造を計算機は扱わなければならなくなった。こうして言語が最初から提供する整数や、文字といったデータ型を組み合わせて、より抽象的なデータ構造の定義を許す言語が登場した。Pascal の record 型、C の struct 型などがこの例である。

複雑な情報を、抽象的なデータ型で表せることの利点は今更述べるまでもないが、このように手続き型言語が進化してもプログラムの構造の方は相変わらず「順次実行される固定計算木」のままであった。情報はより柔軟な構造を表現できるようになったのに、それが狭苦しい階層型手続きのなかに閉じこめられてきたのである。このような構造はプログラムがある時点から開始して、決められた手順で仕事を執り行い、一定の時間内に終了するという世界では有効だった。しかし情報そのものが取り扱いの対象になり、それにどのような種類の計算や操作が加えられるかをより柔軟に決めたいという世界では必ずしも最適ではない。

情報そのものを取り出して、それに必要な時に必要な順序でさまざまな操作を加えようと思った場合、そのデータ構造に加えることのできる操作はそのデータ構造自身と共に存在していなければならないし（必要な操作手続きを探してそのデータ構造を扱うサブルーチンの森をさまようというのは現実的な解ではない。どのような文脈で用いられるかがはっきりしていないのだから）、「操作の意味」もはっきりとしていなければ安心して使うことはできない。このことからデータをその操作手続き群とともに定義して、きちんと決められた界面からのアクセスのみを許そうとするやりかたが考え出された。手続きとデータ両者の視点からのモジュール化を実現したもの、それがオブジェクト指向である。

シミュレーション記述言語 Simula-67 にその端を発する「オブジェクト指向」は 30 年近くの歳月を経て成長を続けてきた大変成熟した技術である。それにもかかわらず広く世界に広まらなかったのは、

高価であったこと
実行が遅かったこと
学習が難しかったこと

という理由に集約されるであろう。確かに最初の二点に関して言えば、長い間真であり続けた事柄であるが、近年の安価で高速な計算機環境では大きな問題ではなくなって来た。また最後の点は必ずしも正しくはない。多くの比較事例が示すことだが、オブジェクト指向言語の習得は他の言語の習得に比べて決して難しいわけではない。ただ上でも述べたように、今までもっぱら「計算」に目を向けてきた言語に慣れ親しんだ技術者が、突如「情報」自身の目で周囲を見よと言われて当惑しているというのが、オブジェクト指向が困難という評判に拍車をかけているように思える。

オブジェクト指向を一気に有名にしたのが米 Xerox の研究所で開発された Smalltalk である。その Smalltalk-80 システムは単なる言語ではなく、開発環境であり、優れたプログラムの宝庫でもあった。

□ ComponentWareの誕生 □

オブジェクト指向は開発者に対して鋭い挑戦を突きつけてきた。その挑戦の最たるものは、そのシステム構築アプローチにおける本質的な抽象性である（ここでいう「抽象性」とは時々世間で使われる「曖昧」という意味ではもちろんない）。オブジェクト指向言語を直接用いてシステムを開発しようとする者は、「情報のありかた」に関する深い洞察が必要である。こうして考察された情報は「クラス」という構造として定義される。「クラス」はデータの構造とその操作を定義する場所であって、データ（＋手続き）そのものではない。実際の実行に際しては「クラス」の定義にしたがって「インスタンス」と呼ばれる実際のデータ＝オブジェクトが生成され、このオブジェクトに対してクラスであらかじめ定義された操作の実行を依頼するというのが普通のスタイルである。

インスタンスの配置は実行時に行われるが、クラスの設計時にはインスタンス同士の具体的な結びつきは未定の場合が多く（それ故によく抽象されたクラスはより柔軟な拡張性を提供できるのであるけれども）、アプリケーションの開発者はインスタンスの実際の配置を既存のクラスの定義を通して考えることになる。普通のオブジェクト指向言語を使った開発では、こうしてアプリケーション開発者が考えたインスタンスの組み合わせ構造も「クラス」のレベルで記述しなければならない。たとえば特定アプリケーションのためだけに存在するサブクラスや初期化とインスタンス統合のための「アプリケーションクラス」などがその例である。しかし純粋なアプリケーション開発者の目から見た場合、このようなクラスを通しての抽象性はいつでも必要なわけではない。アプリケーション開発者の関心はより高いところにある。

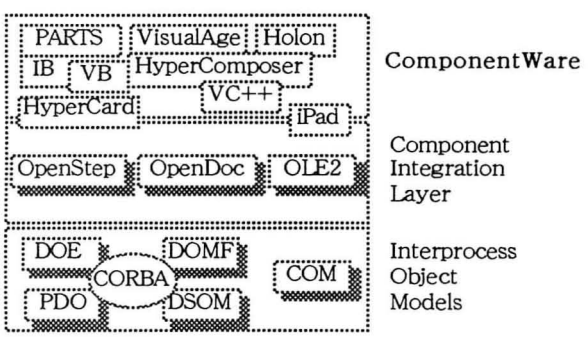
アプリケーション開発者が相手をしている顧客は、通常計算機の専門家ではない。顧客が求めるビジネスのモデルを上手に抽出してやり、情報構造を決定し、必要なインターフェイスを提供してやるのがアプリケーション開発者の仕事である。もしこのときに吟味されたアプリケーションのための部品が多数存在し、それらをインスタンスとして実際に配置する仕掛けがあれば、より容易にアプリケーションを構築することができる。

こうした流れを受けて、既に用意されている部品を組み合わせて素早いアプリケーション構築を許すシステムが最近の流行である。Visual Basic がその代表選手のように扱われている場合が多いが、こ

れはもちろんオブジェクト指向を否定する方向ではない。少し観察してみれば、このシステムがオブジェクト指向の概念を敷衍したものであることが容易に理解される筈だ。コントロールと呼ばれる部品群はオブジェクトそのものである。この世界ではインスタンスを組み合わせてアプリケーションを作成する。このようにクラスではなく、インスタンスを組み合わせ、アプリケーション構築を許すシステムはComponentWareと呼ばれる。

従来のアプリケーションレベルより基本的な層でのコンポーネント化も進んでいる。代表選手がOpenDocとOLE2である。これらは分散環境上に存在するオブジェクト（コンポーネント）を統合・相互運用するための仕掛けである。有用性の高いコンポーネントが多く存在すれば、利用者は自らのドキュメント上にそれらの部品を単に配置するだけで、かなり高度なEUC (End User Computing) 環境を実現できるようになる。こういった層をここではComponent Integration Layerと呼ぶ。

この文書で説明している ComponentWareとは、こうしたシステムレベルの部品統合環境よりも更に上位に存在し、単純な部品の組み合わせ以上のアプリケーションを実現するための開発環境である。次の図はComponentWareとそれを支える階層を図式化したものである。



ここで、更にオブジェクト指向におけるインスタンスの役どころを考えよう。ここでは現実の世界でなされる様々な仕事がインスタンス群の協調によって実現されている。例えば図書館を考えてみよう、ある日の図書館の様子を観察すると、次のような出来事が観察されたとする。

・広尾図書館に佐藤さんがやってきて「若草物語」を借りた
・広尾図書館に鈴木さんがやってきて「おぢいさんのランプ」を返却した
・相模原図書館から広尾図書館に問い合わせがやってきた
（以下同様）

ここに登場する広尾、相模原図書館や、佐藤さん鈴木さん、そして「若草物語」や「おぢいさんのランプ」などは皆立派なインスタンス候補である。もし図書館のシステムが作られた場合には、これらに対応するインスタンスが生み出され、実際に求められている操作が行われることになる。

ところであるシステムを考えたときに、あらかじめ全てのインスタンスを書き出しておくことができるだろうか？考えなくてもこれが全く現実味を書いた試みだということは直ぐにわかる。上の図書館の例を考えても全てのインスタンスを逐一書き出して定義していたら、このシステムが完成する見込みは全くない。

そこで私達は適当な抽象化を行うことになる。例えば上に述べた図書館の出来事をもう少し一般化してみると、次のようなものになるだろう。

・<x> 図書館に <y> さんがやってきて <z> を借りた

ただし <x>, <y>, <z > にはそれぞれ図書館、利用者、本の型を持つインスタンスの一つが入る。

このような記述を「シナリオ」と呼ぶ。シナリオは考えているシステムが提供するサービスを書き並べたものだと考えることができる。シナリオを繰り返し分析することにより、そこに必要とされるインスタンスの持つべき性質が次々と浮かび上がり、そして同時に不要な部分がそぎ落とされて行くことになるだろう（蛇足：「作品が完成したと思えるのは、手を加えるものが何もなくなったときではなく、取り除くものが何もなくなったときである（サン＝テグジュペリ）」）。

ともあれ、こうして抽出された無名のインスタンスの性質が「クラス」の定義となる。必要十分なクラスの定義が集められたときに、オブジェクト指

向分析或いは設計はひとまず終了することになる。

厳密に記述されたシナリオは分析や設計のモデルの意味を与え、分析・設計モデルの検証を助けることができる。シナリオが実現されるために必要な文脈（舞台）の条件を規定し、登場するインスタンス（役者）に制約を課すことになる。ここで「シナリオ」という言葉をもう少し詳しく考えてみよう。映画や舞台のシナリオを考えると、出来事を逐一書き記したものであるという印象がある。ではシナリオは「手順書」なのだろうか？直観に反して、少なくともソフトウェアの「シナリオ」は何が起きるべきか（仕様）を記述したものに過ぎず、それをどのように実現すべきか（設計）についてはあまり述べてはいないのである。ひるがえって映画や、演劇のシナリオを考えてみてもそれは脚本家による「仕様」に過ぎず、設計は別の役割の人物（監督等）の仕事であると考えることがでるだろう。

さてシナリオを表現するための補助手段として用いられるのが、オブジェクト図 あるいはインスタンス図である（前者は Booch の、後者は Rumbaugh の用語）。ここに登場する役者はクラスそのものでもなく個々のインスタンスそのものでもない。いわば役割を与えられた「無名」のインスタンス達である。再び図書館の例を考えると「ある図書館」「ある利用者」「ある本」といった匿名のインスタンスを用いながらそれらの関係だけに着目してオブジェクト図を描き示すことができる。ここでは「役割を与えられたインスタンス」の関係が表現されることになるのである。

ここでやっと ComponentWare に話が戻る。先に「ComponentWare とはインスタンス（コンポーネント）を直接関係付けていくことで、アプリケーションを作成していくことのできる環境」と述べたが、ここまでの解説を経て、次のように言い直すことがことがでる。即ち、ComponentWare とは「有名・無名のインスタンス群を用いてシナリオを直接記述させ、アプリケーションを構築するための環境」なのである。

□ 例題：PARTS Workbench 環境

PARTS Workbench （パーツワークベンチ）環境は伝統的なクラスベースのプログラミング環境と、新しいインスタンスベースのプログラミング環境を併せ持つ、強力なアプリケーション開発システムである。インスタンスベースの作業環境の上で利用者はあたかもプラモデルを組み立てるかのようにアプリケーションを構築していくことができる。背後に Smalltalk （米国 DIGITALK 社製）の開発環境を持っているため、必要ならクラスレベルの開発も行える。現在 Windows 3.x, Windows NT, OS/2 版が利用できるが、ここでの例は Windows 3.x を用いて行う。

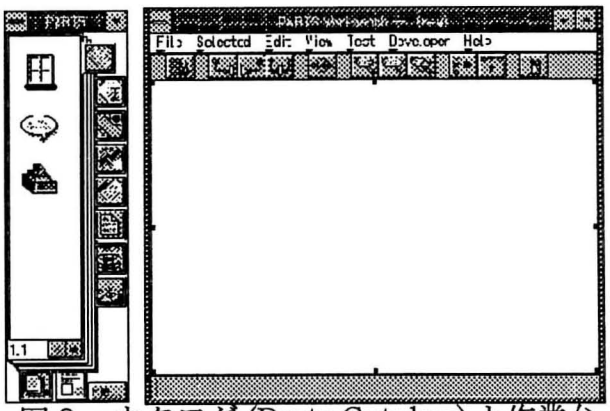


図２：カタログ (Parts Catalog) と作業台 (Workbench)

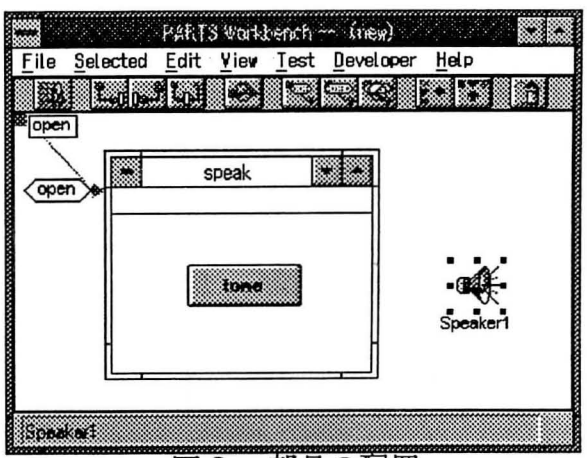


図３：部品の配置

図１に示したものが PARTS Workbench の典型的な作業環境である。画面の左側にあるのが「カタログ」と呼ばれる部品倉庫、右側が「作業台」と呼

ばれる実際のアプリケーション構築場である。開発者はカタログから部品を作業台へドラッグして必要な部品を配置しつつ、それらの部品間を <イベント、メッセージ> の組を指定しながらリンクと呼ばれる線を引いて行く。標準のカタログには普通のGUI 部品（ウィンドウ、ボタン、リストボックス等々）の他に、オブジェクトの集合を表現する部品や、条件判断により新たなイベントを生成する部品、ファイルアクセス部品、DDE や DLL へのアクセス部品、クリップボードの操作部品、メニュー構築部品、様々なダイアログ部品等、様々な「プログラミング」上すぐに役に立つ部品が登録されている。

簡単なアプリケーションの例を示そう。アプリケーションとしては殆ど意味がないが、ボタンを押すとコンピュータのスピーカーが鳴るというプログラムを考える。まず図２で示したように必要な部品を画面上に配置する。このやりかたはドラッグを用いた直感的な方法である。

図３はボタン部品とスピーカー部品の間にリンクと呼ばれる関係を設定しようとしているところである。マウスの右ボタンを用いてボタン部品からスピーカー部品にドラッグをおこない右ボタンを離すと、図のようなダイアログウィンドウが表示される。左側に並んでいるのがボタン部品から発生する「イベント」の一覧で、右に並んでいるのがスピーカー部品が受け取ることのできる「メッセージ」の一覧である。ここではclickedイベントとtoneメッセージを選んで OK ボタンを押す。

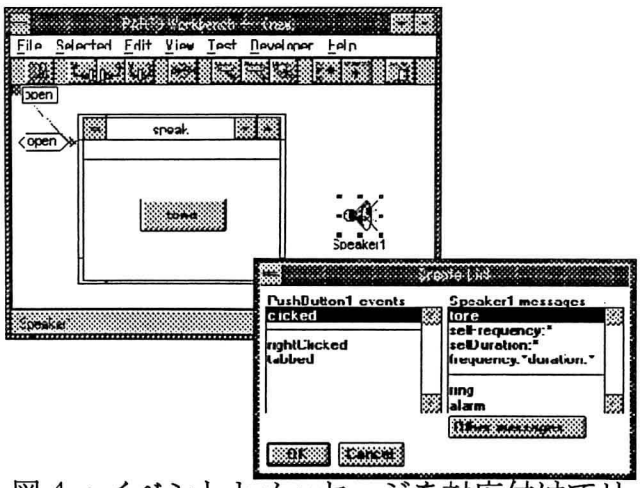


図４：イベントとメッセージを対応付けてリンクを生成

図４が完成画面である。ウィンドウ部品に対して open メッセージが送られているが、これはアプリケーション起動時に送られる外部からの open イベントを内側にリレーしているリンクである。ボタン部品からは利用者が選んだ通り clicked イベントが起きると、tone メッセージが呼び出されるというリンクが張られている。この時点で作業台左上の起動ボタンを押せば、直ぐに実行が始まる[1]。この時点で利用者はまったくソースコードを書いていないことに注目して貰いたい。またこの例ではメッセージに引数を渡していないが、もちろん処理に必要な引数を渡すこともできる。

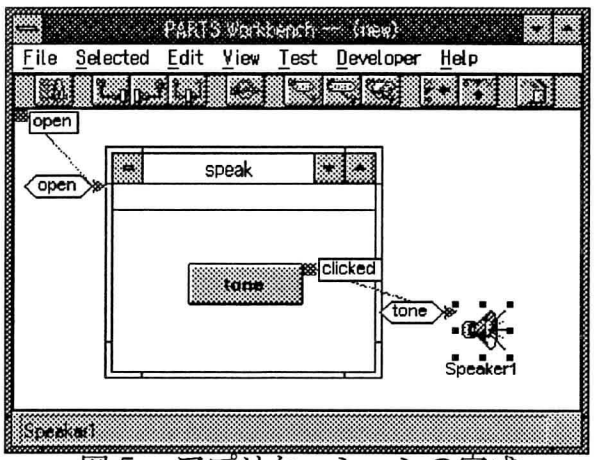


図５：アプリケーションの完成

先にも述べたとおり PARTS Workbench は米国 DIGITALK 社が開発した Smalltalk 環境の上に実現されている。原則としてすべての部品には対応する Smalltalk のソースコードが存在している。これは高度なカスタマイズがクラスレベルで Smalltalk を用いることによって行えることを意味している。それだけでは既存のオブジェクト指向言語環境と同じだが、PARTS Workbench は更にインスタンスベースでのカスタマイズも許している。

インスタンスベースで新しい部品を作るのには二通りの方法がサポートされている。（１）上の例のように部品同士をリンクで組み合わせた部品群を、改めて大きな部品として利用する方法と、（２）個々のインスタンス部品にスクリプト言語を用いて独自の機能を定義する方法である。どちらの方法を採用しても利用者が定義した新しいインスタンス部品は

そのままカタログに登録して再利用することが可能である。カタログは複数持つことができ、複数のカタログを同時に並べて仕事をすることもできるので、組織のカタログ、プロジェクトのカタログ、個人のカタログといった具合に目的別に部品を整理することができる。なお（2）で用いられるスクリプト言語はPARTStalkと呼ばれるものだが、実際にはSmalltalkそのものと考えても良い。

従来の手続き型言語を用いた世界における「部品（モジュール）」は、先にも述べたように「計算木」を表すサブルーチンと、「構造化されたデータ」を表すレコード型等の様々な形で表現されていて、それらの組み合わせ方は個々のケースにより様々であった。これに対してComponentWareの世界では部品＝インスタンスという構造が基本である。特にPARTS WorkbenchのようにSmalltalkベースのComponentWareでは、クラスもインスタンスも最終的には全てがオブジェクトの世界に収斂するので、非常に高度な統合環境を手に入れやすい。

□ ComponentWareによるシステム開発

ネットワーク環境におけるアプリケーションの構造を考えてみよう。図5はその構造を簡単な図式で表したものである。この図において「インターフェイス」は利用者のためのインターフェイス（ウィンドウなど）を表し、「データモデル」はアプリケーションの本質的なデータ構造（オブジェクト構造）を表している。「アプリケーション」はそれぞれの「インターフェース」と「データモデル」の協調のための下働きである。「データモデル」は、どのような構造でも取り得るが、その物理的なデータは例えばネットワーク上の「サーバー」に保存されている。このようなアプリケーションモデルに、GUI Builder, Client/Server Builder, ComponentWareと呼ばれる開発ツール・環境がそれぞれどのようにアプローチしているかを以下に見てみよう。

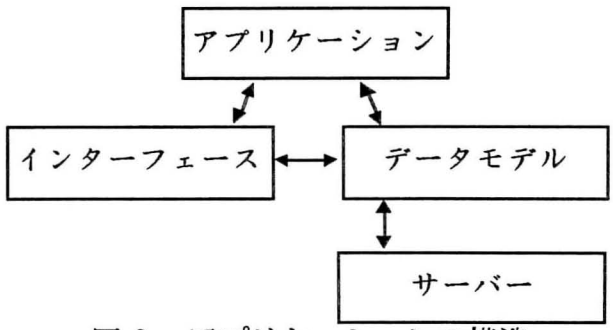


図6：アプリケーションの構造

GUI Builderを用いたアプリケーション開発では、単にGUIの部分が手早く作成できるだけであり、それ以外の全ての部分を利用者が作成しなければならない。図5では「インターフェース」の部分だけが作成できる。ウィンドウシステムが流行り始めた頃は隆盛を誇ったものの今やこのカテゴリのツールは急速に勢いを失いつつある。

GUI Builderに更にデータベースアクセスのための機能を加え、一つのアプリケーションの完成まで行わせてくれるのがC/S (Client/Server) Builderである。これはデータベースの提供するデータモデル（関係モデルが圧倒的である）の上にインターフェースを対応付ける手助けをしてくれるツールである。より複雑なデータ構造を作成しようとする場合付属のスクリプト言語で行うことになるが、多くの場合言語自身の能力はSmalltalk言語の提供するものに及ばないようである。現在流行の波に乗っているツール群である。

ComponentWareは「アプリケーション」「インターフェース」「データモデル」の全てを部品化して対等に扱い、開発者の作成したいアプリケーションに柔軟に対応する。一つのデータモデルに複数のインターフェースを与えることも、あるいは一つのインターフェースが複数のデータモデルに対応することもできる。データモデルを部品化して扱うことができるということは、類似のアプリケーションで共有されるロジックも部品化しやすいということである。画面のデザインや、特殊なデータ処理等々部品化され得る候補は多い。こういった資産はカタログの中に分類して登録され、再利用されるのを待つだけとなる。

「ビジュアル」を売り物にする開発ツールは多

いが、広い意味でのComponentWareは単にインターフェース部分の構築をビジュアルに行わせるだけのものではない。すべての部品を対等に、しかも視覚的に扱うことを許すものがComponentWareである。

Smalltalk環境はかつてはその閉鎖性も何かと批判の的だったが、例えばPARTS Workbench環境では、外部に対する様々なアクセス手段が提供されている。先に述べたDDE, DLLはもちろんOS/2版ではSOMも「部品」としてアクセスできる。C/S Builderが提供するデータベースアクセスのための部品も提供されていて、オブジェクト指向の世界からSQLの世界への橋渡しを行ってくれる。一般に様々な外部リソースをオブジェクトの皮を被せて環境の一部として取り込んでいくことができるのである。

すでにおわかりのように、単にインスタンスの配置が行えるだけでは、その有効性を発揮させることはできない。ComponentWare環境が満たすべき特徴としては、とりわけ重要なものだけを考えても、下のようなものになるだろう。なお重要さは順不同である。

（1）シナリオを表現する場所（＝部品）を提供すること
（2）クラスレベルの定義ができること
（3）インスタンスレベルの配置ができること
（4）作成した部品を組込部品と区別無く用いることができること
（5）多相性（Polymorphism）が提供されること

（1）が必要な理由は先に述べたとおりで、オブジェクト指向分析・設計の一翼を担うシナリオの表現能力がシステムを試行錯誤で構築していくときの有効な道具となり得る。（2）は抽象化という意味で重要である。単に型紙を定義できるというだけでなく、継承を用いて安定してかつ拡張性を残したシステム作りを狙うことが可能になるからだ。（3）は当然である。供給される実際に「動く」部品を吟味しつつ、あらたなアプリケーションを作成できることがComponentWareの醍醐味なのだから。（4）と（5）は自由度と柔軟性を高める役に立つ。

残念なことにVisualBasicはこのうちの（3）と多少の（5）しか持って居ない。そのためVisualBasicはその商業的成功にもかかわらず、ComponentWareとしては辺境のシステムであると述べざるを得ない（筆者はVisualBasicを否定しない、純粋に技術的な側面を述べている）。これに対して例えば背後にObjective-C環境を持つNEXTSTEPや、Smalltalk環境を持つVisualSmalltalk (PARTS Workbench)、VisualAgeでは、まがりなりにも（1）～（5）の特徴が提供されている。筆者の私見では、たとえ背後にオブジェクト指向環境を抱えていなくても最低限（3）～（5）は提供されるべきだと考える。

（4）が提供されることによって、プロジェクトの中で部品開発が行いやすくなる。これは技術者の職能を考えたときにも非常に大切な事である。かつてはそれなりに大きなシステムでも要求定義から実装までを同一人物が関わりながら開発を行うことが可能であった。しかし高度な基盤が提供されるにつれ、ソフトウェア技術者の世界でも専門職による分業化が進んで来ている。もはや誰も全体を見通して開発を行うことはできない。ソフトウェアを様々なレベル（要求に近いレベルから、実際のプログラミングに近いレベルまで）の部品として記述し、融通し合えるような器が必要とされているのである。ComponentWareはこのような要求に応えるために生まれてきたという側面もあると言えるだろう。

図書館の例の所でも述べたが、すべてのインスタンスを記述することは不可能である。もし（3）しかできないコンポーネントプログラム環境があったとすると、それはオブジェクト指向で開発しよう考える技術者者に「全てのインスタンスを書き下ろせ」と迫るようなものである。現実問題としてそれは不可能なわけであるから結局より静的なインスタンス配置を持つシステムしか記述できないことになる。

もっとも昔ながらの事務アプリケーションを考えると、データは全てDBに格納されており、機能を介したその間のデータフローを考えれば良いものが多いので、そういった目的には一応利用することはでる。ただし、ここで注意しなければならないことがある。先にも同様の事を別の形（図6）で述べたが、現在いわゆるクライアント＝サーバ・アプリケーションの世界では「2層構造から3層構造へ」というスローガンが唱えられ始めている。これは大

量のデータサーバと小回りの利く入出力端末という2層構造から、間にもっと高度なサービスを提供するレイヤを挟み込んで3層構造とし、より高度なサービスを提供しようという構想である。なんだか話をそのまま聞いているだけだとBell研究所の「Plan9 Project」を彷彿させるスローガンである。ここで大事なことは高度なサービスのためには高度なデータ（オブジェクト）構造が必要であるということで、これはとりもなおさず強力なデータモデリング能力をもったシステムが必要とされているということに他ならない。少なくともこういったデータモデリングにBASICが全く不向きであることは論を待たないであろう。

□ 部品の発見

ComponentWare環境では、どのように部品を発見し、作成していけば良いのだろうか。またその作業とオブジェクト指向分析・設計との関係はどうなっているのだろうか。

この問題を考えるためにWindowsの環境を観察してみよう。ここに登場するGUIのための部品はコントロールと呼ばれているが、これらの部品はどのように決定されて来たのだろうか。このインターフェースはもちろんMicrosoft一社が恣意的に決めた物ではない。周知のようにウィンドウを用いる環境としてはWindowsに先立つ、いくつかのお手本があった。Xeroxのワークステーションや、AppleのMacintosh等である。これらのシステムは、試行錯誤を繰り返しつつ、ウィンドウ指向プログラムのためのフレームワークを提供してきた。

こうした流れの中で、ウィンドウ、ボタン、トグル、スクロールバー、テキスト入力フィールド、メニュー等の「共通項」が洗い出されてきたのである。この「共通項」は「ウィンドウ指向プログラミング」を遂行する上で頻繁に再利用されやすい概念であると考えられる。ここで部品抽出の対象になったのは「ウィンドウ指向プログラミング」の世界であるが、このように解かれるべき問題の世界を一般に「問題領域（Problem Domain）」と呼ぶ。ウィンドウ指向プログラミングの世界は、それ自身「視覚的」な問題領域である故に、多くの「視覚的開発支援環境」が生み出された。GUI Builderはこの部分だけを支援しているわけである。

こうして「問題領域」に関する共通的特徴の抽出と、それに対応する部品を用意することによって「ウィンドウ指向プログラミング」の少なくともインターフェースの部分は容易なものとなった。ではプログラムの残りの部分はどうなのだろうか。

オブジェクト指向との関係で考えると、GUIの各部品は「視覚化」されたオブジェクトそのものと考えることができる。オブジェクト指向では、これをGUIの世界のみならず、アプリケーション全体まで広げて考えようとしているのである。ここで問題領域という目で図5を眺め直してみよう。すると、ここに登場する箱はそれぞれ「インターフェース」「データモデル」「データサーバー」「アプリケーション」といった問題領域に関する部品の集合体であると考えることができる。「インターフェース」に関しては上の説明で良いだろうが、それ以外の部分はどうだろう。

例えば「データモデル」のための部品開発は、開発しようとするアプリケーションが対象にしている問題領域を詳細に分析することから始まる。ここで、そこから真っ直ぐ開発を行うだけなら、通常の開発とあまり変わりがないし、オブジェクト指向あるいはComponentWareのメリットを享受することはあまりないだろう。ところが実際には同じ問題領域に対する開発は繰り返し行われているので、ここに「ウィンドウ指向プログラミング」で行ったような分析を行うことができそうである。

もちろんこの目標は、考え方としては決して新しいものではない。これまでも特定の問題領域に対する解を提供する部品がサブルーチンの形等で提供されてきた。このようなアプローチは、局所的な解を素早く得るためには有効であり、現に科学技術計算の分野では様々なアルゴリズムがルーチンの形で用意されている。プログラムが定型入力を逐次的に処理して、出力を得るだけならこれでも十分である。しかしながら一般的な近代的アプリケーションは計算だけではなく、顧客からの様々な要求に応える形で作成されなければならない。業務知識から導かれるデータフローとデータの制約の分析と設計、それを実現するための画面設計、データ設計、帳票設計、アルゴリズム設計、プロセス設計、プロトコル設計、ネットワーク設計等々を行いしかもそれらを実現に向かってうまく統合して行かなくてはならない。これらはそれぞれ相異なる「問題領域」であり高度な専門知識が要求される。それぞれの問題領域の内部では日々革新が起きる可能性があり、その改良の影

響がいちいち外部に大きな影響を与えるようでは問題である。このように考えると、個々の問題領域に関する分析を行いその領域に特化した部品の用意をする仕事と、それらの各問題領域に対応した解を集めて、実際のアプリケーションを組み上げる仕事の二種類が存在することに気付くだろう。

これまでのソフトウェア開発は、この二種類の仕事がはっきりとわかれていない場合が多かった。しかし開発対象のソフトウェアシステムが複雑になるにつれ、一人あるいは小さなグループがすべて行うのは不可能になっている。現に今やいちいちWindows に相当するシステムを作りながら、同時にアプリケーションを開発したりはしない。

オブジェクト指向が提供する実現技術はこの二種類の仕事の領域で行われた仕事を統合するための便利な仕掛けでもある。単なるサブルーチン集だけではなかなか成し得なかった、様々な問題領域のための部品統合がオブジェクト指向を採用することで行いやすくなったのである。ComponentWare はそれをインスタンスレベルで操作させることによって、更に使いやすくしたものとして捉えることができる。

ComponentWare （オブジェクト指向も同様）のための部品をどのように得るのか、という問題は、結局繰り返されるアプリケーション開発を通して「問題領域」として切り分け可能な界面を順次発見し、それらをうまく抽象しながら「クラス」や「インスタンス」の形に順次昇華していくという手段で解決される。これは従来の開発手法によっても行われていた手続きの部品化に思想的には似通っているが、実際に生み出された部品の組み合わせ可能性という意味ではるかに勝っているため、「ソフトウェア資産形成」のための強力な動機付けとなる。

こうした問題領域分析 (Domain Analysis)は現在オブジェクト指向分析設計をよりよく行うために盛んに研究されている手法である。問題領域分析自身はまだ発展途上の技術であるが、無理を承知で簡単に定義付けると、その活動は以下の２点に集約される。

（１）問題領域に関する辞書の作成（共通語彙の抽出と意味の定義、制約条件）

（２）問題領域記述言語の定義（構文、文法）

問題領域を観察して得られる共通概念（辞書の項目）がオブジェクト指向の「クラス」の候補となり、インスタンスとして ComponentWare 環境で利用される。制約条件は「クラス」自身の制約であったり、インスタンスの制約であったり、あるいはオブジェクト間の制約等に反映される。問題領域記述言語とは、その問題を表現するのに特に適した言語であり、理想的には問題領域毎に用意されるものであるが、構文は既存の言語のものを借りても構わない。

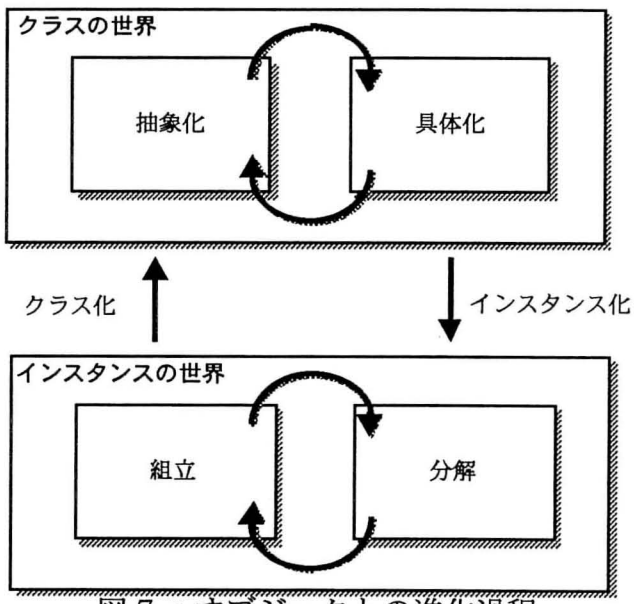


図７：オブジェクトの進化過程

クラスの概念を持たない ComponentWare は真のプロフェッショナル技術者の目には中途半端なものと映るかも知れない。確かにインスタンスを直接操作するやりかたは、非常に直感的であるし効果も直接的である。しかしながらインスタンスだけに頼っていては用意される部品の管理がすぐに破綻してしまうだろう。開発のサイクル中に随時クラスで概念の整理を行い、そこから生み出されたインスタンス群を組み合わせながら、新たなクラスへの昇華を行うというサイクルが ComponentWare の世界では理想であろう。

部品のライフサイクルはどのようなものであろうか。図７は青木淳氏の著書（「オブジェクト指向システム分析設計入門」ソフト・リサーチ・センター刊）に述べられたクラスライブラリの進化過程を元に翻案したものである。オブジェクト指向開発過程

で起こるタスクを示している。

オブジェクト指向システムは「クラスの世界」と「インスタンスの世界」で構成されている。開発者はクラスをインスタンス化し、アプリケーションを組み立てたり、作成したインスタンス部品からクラス抽出のアイデアを得ている。

「クラスの世界」の「抽象化」と「具体化」はそれぞれ「スーパークラスへの抽象化」「サブクラス作成による具体化」のサブタスクを指している。また「インスタンスの世界」の「組立」と「分解」はそれぞれ「部品の配置と連係」「部品の分解と再配置」のサブタスクを示す。

通常のオブジェクト指向言語環境は「クラスの世界」に対する操作を提供していて、「インスタンスの世界」に関する支援はクラスの世界を介した間接的なものである、実際には開発者が頭の中で行っている。これに対してComponentWareは図6の「インスタンスの世界」の作業に直接の支援を与えるための枠組みである。

図8が示すものはComponentWareと通常のオブジェクト指向言語環境の協調関係であり、このことからもComponentWareとクラスベースの支援環境の並存が望ましいことが窺える。

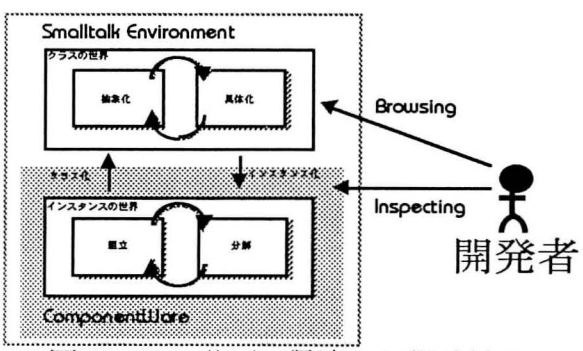


図8：Smalltalk環境への埋め込み

Smalltalk等の動的言語上に構築されたComponentWareは、Smalltalk自身が備える強力な開発環境自身に加えて、インスタンスの構造や定義を同じSmalltalkという土俵の上で扱うことができる。すわなち図7に示した過程を、自分自身の環境の中で直接観察できるということである（図8を参照）。

システムが一つ考案されると、様々な側面から記述されたモデル群が残される。これらのモデルはいずれもシステムを構成する問題領域の特徴を表現していると考えることができる。こういったモデルをうまく抽象化して残しておけば、次に似たようなシステムを企画するときの参考になると考えられる。ごく自然な考え方である。オブジェクト指向では抽象クラス（Abstract Class）の存在がそういった上位の概念の再利用を助けている。

実際のソフトウェア開発に利用するために、様々な問題領域に対応したモデルを体系的に構築する手法を問題領域分析と呼ぶ。問題領域分析自身は手法が確定しているわけではないが、問題領域記述言語を定義したり、開発プロダクトや開発プロセスをモデル化したりといった作業が含まれることは概ね合意されているようである。

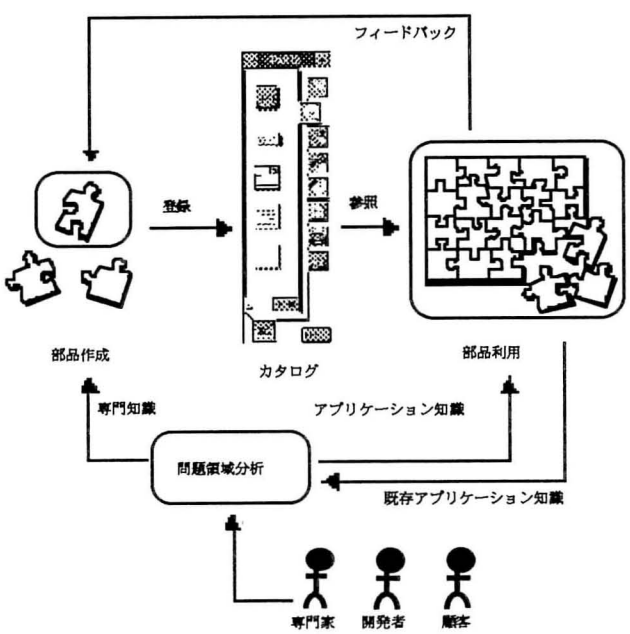
ComponentWareも問題領域分析に寄与することができる。クラスレベルのモデリングの寄与は勿論だが、インスタンスレベルで行われる配置は先にも述べたようにシナリオのより直接的表現であり、仕様のヒントあるいはより下位レベルに適用されれば設計を表現した部品として考えることも可能になる。逆に言えば、問題領域分析を通して良質の部品を作成できるか否かがComponentWareが活用され得るか否かの勝負の分かれ目といえるだろう。

□ とりあえずのまとめ

ComponentWareを「簡易」アプリケーション開発ツールとだけ認識するのは誤りである。それは確かにアプリケーションの構築を「簡易」にするが、作成されるアプリケーションは必ずしも単純なものである必要はない。今まで登場してきた多くの「簡易アプリケーション開発ツール」は、与えられた枠組みの中での開発は簡易だったが、作成されるアプリケーション自身も「簡易」なものになり、それを拡張しようとすると極端に難しいかあるいは不可能であることが多かった。これに対してPARTS Workbenchを始めとする拡張性に優れたComponentWareツール群は、アプリケーション開発を簡素化しつつ、複雑な内容の処理も定義できるという特性を持っている。

また既存のオブジェクト指向分析・設計手法との相性の良さにも注目したい。よく使われているOMT (Object Modeling Technique) 法にしろ、Booch 法にしろ、イベントを交換し合うオブジェクト群といったモデル化を行うが、このモデルは例えばPARTS Workbench における <イベント、メッセージ> という形式にそのままマップしやすい。故に構造化分析設計 (SA/SD) の時代に必要だった、DFD (Data Flow Diagram) から構造化チャート (Structured Chart) への変換といった「設計から実装への変換操作」もより容易である。このことは開発者が分析設計をより力をこめて行う動機となることだろう。

最後に問題領域分析を視野に入れたComponentWare の利用形態を図示しておこう。



好むと好まざるとに関わらず、オブジェクト指向は既にソフトウェア開発の現場を席巻しつつある。それなしにはもはや複雑性を開発者が制御できないからである。そしてそういった状況の下であらゆるレベルの開発者（エンドユーザから上級 SE 迄）に対して門戸を開く ComponentWare の重要性もますます増大して行くに違いない。

□ 参考文献について

[1] "ComponentWare", BYTE magazine, May 1994

ComponentWare という言葉を初めて前面に押し出して取り上げた記事。冒頭にいきなり「オブジェクト指向プログラミングは失敗した」とあるので、何の事かと驚く。

[2] "VisualProgramming", IEEE Computer, Mar 1995

視覚的プログラミングシステムとしてのComponentWare ついて述べたもの。同じ号に特集されている "Emersing Standards for Component Software" と "Taligent's CommonPoint: The Promise of Objects" も必読

[3] "Inter-Operable Objects", Dr. Dobb's Journal, Oct 1994

分散オブジェクトによるアプリケーション構築の将来像に関して述べた記事群。

[4] G. Booch, "Object-Oriented analysis and design with applications", The Benjamin/ Cummings Publishing Co. 1994

[5] J.Rumbaugh, "Modeling and Design", articles in JOOP Jan., Feb., Mar. 1995

上記の２つを読み合わせて、最近の「オブジェクト指向方法論」につけ加えられたものを学習するとComponentWare の占めるべき位置が少々見えて来る。

[6] Ruben Prieto-Diaz and Guillermo Arango ed., "Domain Analysis and Software Systems Modeling", IEEE Computer Society Press, 1991

今となっては少々古いが、問題領域分析とモデリングに関するまとまった論文集である

□ 付録：ツール比較ひとくちメモ □

お断り：以下の記述は筆者の極めて個人的な見解に基づく評価であり、特定の団体や会社に対する不当な利益や損害を狙ったものではない。すべてのツールは何らかの手段で筆者自身が実際にさわって評価している。いくつかのツールは純粋にユーザーとして結構本格的なプログラミングにも使用している。ご意見ご感想は sakoh@sra.co.jp まで。

PARTS Workbench (VisualSmalltalk)

米 DIGITALK 社の Smalltalk ベースの ComponentWare 環境。インスタンスベースとクラスベースのプログラミングの両者を可能にしている。部品 (Component) はカタログに登録されており、それを作業場所に配置し、間をリンクで結ぶことによってアプリケーションが完成する。自作の部品をカタログに再登録することも可能。

VisualAge

IBM が販売する PARTS Workbench とほぼ同じ ComponentWare 製品である。下が Smalltalk でできているのも同じ。ほとんど表面的な仕様は変わらないが、PARTS Workbench の方が個々のインスタンスを別個に扱いたい場合に少しだけ自由度が高い。ただしこれはトレードオフの問題である。

最近 VisualAge を C++ で実装し直した版が発表された。これからスクリプト言語はどうなるのだろうか？

VisualWorks

Smalltalk 本家 PPS (ParcPlace Systems) が販売する最新の Smalltalk 環境。実行エンジン、コアのクラスの仕上がりは流石と呼べる品質である。様々なプラットフォーム (Unix, Windows, OS/2, Macintosh) 間で文字通りの「バイナリコンパチビリティ」を実現するこだわりも見事である。最近の流行に乗って GUI ビルダとデータベースアクセスツールがバンドルされているが、PARTS Workbench のようなインスタンスベースの開発環境は持っていない。視覚的なアプリケーション構築もできない。筆者の個人的な希望では VisualWorks の Smalltalk 環境の上に PARTS Workbench を載せて欲しいのだが。

... とぼやいていた筆者の祈りが天に届いたのか、DIGITALK 社と ParcPlace 社の合併が発表された（１９９５年５月）。長生きはするものである。Smalltalk 界の自社連合だという輩もいるものの、これで可能性としては PARTS Workbench on VisualWorks という線も出てきたわけである。しかし過度の期待は失望の元。

IntelligentPad

北大の田中譲教授の指導のもと作成された、純国産 ComponentWare。徹底的な簡素化により LEGO ブロックのような気軽さで部品の組み直しと、アプリケーション作成が可能になった。部品はすべて Pad とよばれ、それらを貼ったり剥がしたりすることで、動的に部品の組み替えも可能である。デスクトップメタファならぬ Pad メタファの提唱としても面白い。プリミティブな部品は C 等で書かなければならないことと、ブラウジング機能が弱いのが難点。IntelligentPad は標準化団体 IntelligentPad Consortium によって研究されていて、研究バージョンは Unix 上のフリーウェアとして公開されている。製品版は Consortium のメンバである富士通と日立によって販売されている。

VisualBasic

米国の BYTE ('94 May) でもとりあげられ、ComponentWare の代表例のように記述された Microsoft の VisualBasic は、日本でも Windows 向けのお手軽な開発環境として注目を集めている。「簡易」プログラミングツールとしての

VisualBasic の手軽さには見るべきものがある。また既に流通が始まっている VBX (Visual Basic eXtention) と呼ばれる部品を購入して、新しい機能を追加することができるのも魅力である。米国では既に相当数の VBX 部品が流通しており、プログラミングコストの削減に一役買っている。日本ではまだそれほど多くの部品は出回っていない。PARTS Workbench の部品流通戦略は DIGITALK 社の行う PARTner プログラムによって推進されているが、登録部品はまだ少ない。

基本的に GUI からコールバックを用いて部品を呼び出すだけのアプリケーションの構築には便利な VisualBasic 環境であるが、自分自身で作成した部品を、更に自分自身の部品として取り込むことはできない。これに対して PARTS Workbench や IntelligentPad は、自分自身で作成した部品を、カタログに登録して再利用できる。

VisualBasic は画面の構築は容易であるが、それ以降コールバックの内容はすべて Basic で記述しなければならない。また本来独立である筈のインターフェースとデータモデルが同じ構造(Form)の上に同居しているために、インターフェースとデータオブジェクトの独立性が低くなり、両者の組み合わせの変更も手間がかかる。PARTS Workbench を用いた場合には、アプリケーションロジックもリンクを用いて視覚的に記述して行くことが可能であり、またオブジェクトの殻で守られたインターフェースとデータオブジェクトの独立性は高い。これは新しいインターフェースの追加、データモデルの変更等をやりやすくする。

Tclbombur

Tclbombur は Tcl/Tkベースの EUC (End User Computing) 環境である。製品としての特性は VisualBasic と似通っている。利用者は GUI を素早く構築し、Tcl 言語で記述されたアプリケーションロジックにアクセスする。丁度 VisualBasic の Basic が Tcl に置き換わったようなものである。VisualBasic と違い市販の部品はそれほど存在しないが、Tcl/Tk のコミュニティが生み出した夥しい数のフリーな部品が存在しているので、開発者はそれらを自由に利用できる。

Tclbombur は UNIX と Windows 環境で全く同じソースコード (Tcl) を利用できるので、Windows, X-Window の両方のプラットフォームに同様のアプリケーションを作成しなければならない時にもメリットを享受できる。なおこれはまだ予定のレベルであるが Tcl/Tk の Macintosh や OS/2 などへの移植も開発スケジュールに入っている。これらが公式にサポートされた場合、Tclbombur もまたそれらのプラットフォーム上に容易に展開可能であろう。

NeXTStep

NeXTStep が提供する InterfaceBuilder はその名に「反して」インターフェース構築以上の仕事をこなせる環境である。その設計思想は PARTS Workbench に非常に似通っている。大きな違いは PARTS Workbench が Smalltalk の上に構築されたものであるのに対して NeXTStep は Objective-C の上に構築された環境であるということである。速度、信頼性、審美性どれをとっても優れた NeXTStep であるが、大きな弱点はプラットフォームとしてマイナーであるということである。これはすなわち市販のツールがそれほど存在しないということである。もし将来的にも自家製のツールを主に使い続ける顧客で、さらに速度や信頼性が重要な因子ならNeXTStep を選ぶのは賢い選択である。

Visual C++

Microsoft のVisualBasic に続く Visual シリーズ第二弾。C++ のための環境である。VisualBasic の手軽さを求めて購入するとあまりの落差に驚く

（筆者の個人的感想）。開発環境としても使いにくく Microsoft の「公式開発環境」であること以上の積極的な意味は現在のところ見出せない。にもかかわらず Win32, Win32c, Win32s の違いを意識したくなかったり、最新の 裏技API にアクセスしようと考えると避けて通れない環境である。つい最近 Macintosh へのクロス開発環境の出荷も始まったが（ホストは Windows NT3.5）、このクロス環境は実際に Word や Excel のクロス開発に使われているものらしい。おかげでこのクロス開発環境を用いてワードプロセッサもしくはスプレッドシートを開発しようとする者は、Microsoft と何らかの特別アグリーメントが必要とのことである（Dr. Dobb's Journal Feb. '95, p158)。プロダクト開発はさておき、普通のアプリケーション開発プロジェクトへの導入に際しては熟慮を要する。

MADO-OS++ for Windows

流通業者の Sofmap がソフトウェア開発に進出して発表した ComponentWare である。部品を組み上げる際には、基本的にプログラミングを行わず、せいぜい複数のアクションをシーケンシャルに起動するだけである。その他には部品間に座標を元にした制約関係が存在し、それをうまく利用してやると Geometric なアプリケーションをある程度簡単に作成できる。現在は組込部品点数も少なく、標準的なスクリプト言語も存在しないので（言語が無いのが売りのようではあるが）汎用のアプリケーション構築環境としては、まだまだ非力であると感じる。同じような部品を次々に張り合わせてより大きなアプリケーションを作成しようという姿勢は IntelligentPad に通じるものがあるが、きちんとした差別化を図らないと、最終的に IntelligentPad のサブセットで終わるのではないかという危惧も少々ある。

最後に。SEA は 95 年度の全国縦断 Forum のテーマとして ComponentWare をとりあげ、製品を提供しているベンダ各位のご協力を得て、デモ等も積極的に行っている。ちなみにご協力いただいている会社とその製品は以下の通りである（順不同）。

IBM ... VisualAge

Apple .. OpenDoc

NEC ... HOLON

PFU ... HyperComposer

富士通、日立SK ... IntelligentPad

SRA ... PARTSWorkbench (VisualSmalltalk)

Domain Engineering

Lloyd G. Williams
Software Engineering Research
Boulder, Colorado USA

Abstract

Systems analysis has traditionally focused on individual systems in isolation. Each new problem has spawned its own project in which the cycle of analysis, specification design, and implementation is repeated. In many cases, however, the problems being solved are similar or even substantially the same. As a result, there is considerable duplication of effort.

The traditional approach focuses on the differences between individual systems from a given problem area or application domain and ignores the commonalty that exists among them. It also ignores the potential for reuse of software products across different, but related, applications from within the domain. This paper discusses an approach to software development that explicitly considers families of systems within a given application domain and exploits the commonalty among them.

1 Introduction

Application systems are designed and developed to solve a particular problem. The particular problem may be the management of customer data for a corporation or the navigation of a spacecraft to Mars. In each case, the individual application can be viewed either as a unique problem or as a member of a set of applications from within its domain.

Here, the term *domain* refers to a particular problem area, such as data management or spacecraft navigation. A domain is characterized both by its subject matter, or content, and by the types of problems that are addressed within that subject matter. The domains of airline flight reservations and air traffic control, for example, address similar subject matter (e.g., aircraft and flights) but deal with very different kinds of problems (e.g., seat reservations versus movement of aircraft through an airspace). Thus, they are different domains. From the point of view of software development, a domain is defined by a collection of (actual and potential) software products that share common characteristics and address a common set of problems.

The meaning of a domain-oriented approach can, perhaps, best be appreciated by contrasting it with the traditional application-oriented approach. In the application-oriented approach, each individual application is approached in isolation. This occurs even though the organization may build several related applications or provide

software for a "product line," a series of essentially similar applications of increasing sophistication, such as a line of microwave ovens or copiers.

A product line typically includes several different models of increasing functionality. More sophisticated models typically provide the capabilities of their less fancy siblings and add new capabilities of their own. There is often significant commonalty across the line. This commonalty is usually exploited by using common hardware parts in all of the models. Unfortunately, this does not extend to the software used to control the parts.

Currently, in most organizations, the construction of each application is organized as a separate project. Each group performs its own development process with little or no coordination with the others. As a result, there is likely to be significant duplication of effort and little opportunity to reuse software products across projects. A domain-oriented approach explicitly considers the collection of potential applications first and particular individual applications second. The work of different groups is coordinated and duplication of effort is avoided. By identifying common components and using a common software architecture, opportunities for reuse are increased.

The idea of using a domain-oriented approach instead of an application-oriented approach is not new. As early as 1976, Parnas advocated the design and development of "program families:"

> "We consider a set of programs to constitute a *family*, whenever it is worthwhile to study programs from the set by *first* studying the common properties of the set and *then* determining the special properties of the individual family members." [Parnas, 1976].

In 1984, Neighbors coined the phrase "domain analysis" to refer to the identification of the objects and operations that represent information common to a class of similar systems within a problem domain [Neighbors, 1984]. Prieto-Diaz [Prieto-Diaz, 1987] and Arango [Arango, 1988] extended this notion and further defined domain analysis.

While these early efforts were aimed at increasing software reuse, there are a number of other potential advantages to domain analysis. The following section discusses several of these advantages. Section 3 describes the activities that contribute to domain engineering. Section 4 discusses domain separation. Section 5 discusses domain-specific architectures. Section 6 presents the notion of a domain-specific environment and describes a software process based on such an environment. Finally, Section 8 discusses the characteristics that help determine whether a given domain is a likely candidate for domain engineering.

2 Why Use a Domain-Oriented Approach?

Using a domain-oriented approach requires more effort than the traditional application-oriented approach. If this effort is to be justified, it must provide corresponding benefits. This section describes the major benefits of using a domain-oriented approach.

2.1 Overcoming the "Thin Spread of Application Domain Knowledge"

In their landmark study of the software design process for large systems, Curtis, Krasner, and Iscoe [Curtis, et al., 1988] identified the "thin spread of application domain knowledge" as one of the most significant problems facing today's software development organizations. Accurate problem domain knowledge is critical to the success of a software project. This knowledge, however, is very difficult to obtain. Typically, software professionals must become experts in the application domain or application domain experts must learn to develop software.

The problem is compounded by the fact that what domain knowledge does exist is spread thinly and unevenly across project personnel. While individual developers may understand certain aspects of the system, the deep domain-specific knowledge required to integrate these different aspects of the problem is often possessed by only one or a few individuals. This knowledge, however, is often the key to the success of a project.

The use of a domain-oriented approach can help overcome the "thin spread of application domain knowledge" and reduce the dependence on key individuals for project success. A domain model captures knowledge about both the components of a problem and their relationships. Documenting this knowledge makes it available to anyone involved with the project. Moreover, documenting the knowledge in the form of a domain model captures this crucial information and preserves it in the event that a key individual is transferred to another project or leaves the organization.

2.2 Establishing a Common Understanding

One of the common frustrations with software development methods, whether of the structured analysis or object-oriented variety, is the lack of uniformity among system models produced by different developers. For a given problem, a team of six developers is likely to produce six (or more!) different models. This variability is often perceived as a lack of definition or rigor in the software development method being used.

The source of this variability, however, is not necessarily in the method. It may come about because of the different conceptual models of the problem held by the different individuals. In any software project, there are a number of stakeholders, including users, marketing, developers, and management. Each of these stakeholders has his or her own view of the proposed system. In addition, as noted above, knowledge about the application domain is spread unevenly among these stakeholders.

The problem is further compounded by the fact that a great deal of knowledge is implicit – "everyone knows about that." Unfortunately, everyone knows something different about that. The fact that this information is implicit and, therefore, never explicitly represented means that the differences in understanding are often not uncovered until some problem occurs in the software, typically during integration testing or later.

Construction of a domain model, an important component of a domain-oriented approach, can help to address these problems. A domain model provides a way of capturing and representing knowledge about the domain. The model can also be used to externalize (make explicit) the implicit knowledge held by various domain experts. Constructing the model makes it possible to identify differences in understanding among the various stakeholders. Once identified, these differences can be resolved before work on the software even begins.

2.3 Economies of Scope

In manufacturing, mass production makes it possible to realize *economies of scale*. When producing large numbers of identical items, such as automobiles, it is possible to justify tailoring the manufacturing process and equipment to those items. Workers can also specialize on individual tasks. Tailoring the process and equipment and specializing job functions improves both productivity and quality. Finally, raw materials can be purchased in large quantities, adding further savings.

Traditional notions of mass production and economies of scale are not readily applicable to software development, however. Software is not manufactured in the traditional sense. Each software product is unique and every new project requires at least some custom design. The closest thing to economies of scale in the software industry is the mass distribution of end-user "shrink-wrap" software packages. The duplication of this software is amenable to the techniques applied to other mass-production enterprises. It is, however, a trivial process when compared with the development of the original software.

A large project may be able to take advantage of some economies of scale. If the project is large enough, it may, for example, be possible to justify buying or building specialized tools. It may also be easier to justify the cost of training for larger projects. However, these benefits are only possible for large-scale efforts and rarely extend beyond the individual project.

The most important economies in software development, therefore, are not economies of scale but *economies of scope*. Economies of scope are

> "... cost reductions or productivity gains that come from developing a series of products within one firm (or facility) more efficiently than building each product from scratch in a separate project." ([Cusumano, 1991], p. 8)

Focusing on a series of products, rather than a single product at a time, makes it possible to realize savings by sharing resources, including tools, methods, people, and software products (e.g., designs and code), across several projects. Traditionally, this has happened only accidentally in software development organizations.

The greatest leverage comes from the production of a number of related systems within the same application domain. Although software tools and methods are supposed to be

domain independent, they rarely are. The tools and techniques used to develop database and real-time software systems are, of necessity, very different. People can rarely become expert in several different domains. Finally, software products from one domain are rarely reusable in another. When they are, they are likely to be so general (e.g., abstract data types) that the payoff for their development and reuse is small.

When a group of related systems is produced, it is also possible to tailor the software development process so that it provides optimal support for their production. While several "generic" software process models exist, experience has shown that they must be adapted to meet the circumstances of individual projects.

The principles of domain engineering form the basis for the success of Japan's software factories. Although they were founded before the term "domain analysis" became popular, the software factories have exploited the idea. Software factories concentrate on developing software within a particular application domain, such as telecommunications or systems software. They concentrate on developing expertise within that domain, designing software components that can be reused across systems within the domain, and optimizing the software process for the types of systems that they produce.

2.4 Evolution

Software systems typically undergo evolution. Evolution may include enhancements in functionality or changes in target platform. Historically, evolution in software systems has been easy to talk about but difficult to achieve. While software is, at least in principle, more malleable than hardware, it is not infinitely so. In particular, making a piece of software easy to change in one dimension inevitably means that it will be less easy to change in others. The capability for evolution, therefore, must be designed into software and the types of change that are likely to be required must be anticipated.

A domain-oriented perspective can help to address this problem. Understanding the domain and the applications that are likely to be constructed within it helps in anticipating the kinds of changes that are likely to be required as the software evolves. The software architecture than then be crafted to make the anticipated changes as easy as possible. Of course, other types of changes are likely to be more difficult. However, a good understanding of the domain can help to ensure that such changes are unlikely to be needed.

2.5 Reuse

One of the earliest motivations for performing domain analysis was reuse. The greatest leverage in software reuse comes from components that are application domain specific [Williams, 1991]. It has long been recognized that there is a trade-off between power and generality in the reuse of software components [Biggerstaff and Richter, 1987]. Generally useful components may be (re)usable in a wide variety of applications. However, these general components are also likely to be low-level and their reuse is

thus unlikely to produce significant productivity gains. The "Booch components" are an example of such generally reusable components [Booch, 1987].

Domain-specific components tend to be larger, providing greater productivity gains for their reuse. The disadvantage is that these components are likely to be reusable only within the application domain for which they were developed. Finally, domain-specific components provide abstractions that are tailored to the needs of a particular application area and thus are easier to understand and (re)use as well as more stable with respect to changes in functional requirements.

3 Domain Engineering

Domain engineering includes several related activities. The particular details of these activities will depend on the approach to software development (methodology) being followed. Thus, these activities are presented as a set of related tasks to be performed, rather than as a coherent process or methodology.

3.1 Defining the Domain

Defining the domain requires establishing both the subject matter to be addressed and the types of problems to be solved. This is, perhaps, the most difficult part of domain analysis. It amounts to answering the question "What business are we in?" The answer defines the types of applications that fall within the domain and, either directly or indirectly, those that do not.

3.2 Domain Analysis

A domain analysis is an investigation of a particular problem area or application domain whose purpose is to discover and exploit the common characteristics of a family of related applications within that domain.

It is possible to identify two very different approaches to domain analysis. Prieto-Diaz has labeled them *synthetic* and *evidentiary* domain analysis [Prieto-Diaz, 1987]. Synthetic domain analysis parallels the traditional systems analysis process. It starts with some theory about the domain and/or a description of one or more desired applications from within the domain. These are then used as a basis for constructing a general model of the domain. Evidentiary domain analysis is based on existing applications. These applications are "reverse engineered" to uncover common data, functions, or components. Ultimately, these components may be refined and placed in a component library.

The following activities contribute to the process of domain analysis:

- *Selecting representative applications:* Applications that are representative of those to be constructed within the domain are selected. These applications may be

planned projects or existing applications. They may also be based on some body of theory about the domain.

- *Modeling the domain:* Once representative applications have been selected, it is possible to construct a model of the domain. This model is broader in scope than a typical requirements model. A domain model captures knowledge about: the application domain (such as monitored and controlled variables or control laws), the environment(s) that applications from the domain must operate within (such as government regulations or arrival rates of events in a real-time system), and any constraints that apply to the applications (such as safety constraints).

  A domain model is also at a higher level of abstraction than a typical requirements model. As with any requirements model, the domain model should not introduce any bias that will unnecessarily constrain the implementation of an application. In addition, it should not unduly constrain the applications that can be specified from the model.

  The particular form of model used will depend both on the type of domain knowledge to be represented and the software development methodology being used. Examples include: object-oriented models [Ward and Williams, 1990], formal (i.e., mathematical) descriptions [Garlan and Delisle, 1990], and AI knowledge representations [Fischer, et al., 1991].

3.3 Development and Validation of Reusable Components

Once a domain model has been constructed, it is possible to identify abstractions that are common to several of the systems in the family. These abstractions then provide a basis for designing reusable components.

These components must then be validated. Components are validated through use. An application is constructed to determine whether the components are as reusable as anticipated. It is not unusual to redesign components several times before an optimum set has been obtained. Experience with object-oriented systems indicates that it can take as many as three iterations to get the "right" design [Love, 1993].

4 Domain Separation

Any completed application inevitably includes components from more than one domain. While any usable banking application requires a user interface, for example, that component is not an intrinsic part of the domain of banking. A user interface is required for any interactive application. Similarly, an aircraft flight management system requires interfaces for devices such as radar altimeters or gyro compasses. These device interfaces represent ways of obtaining needed data but are not, in themselves, part of the domain of aircraft flight management. The data might, for

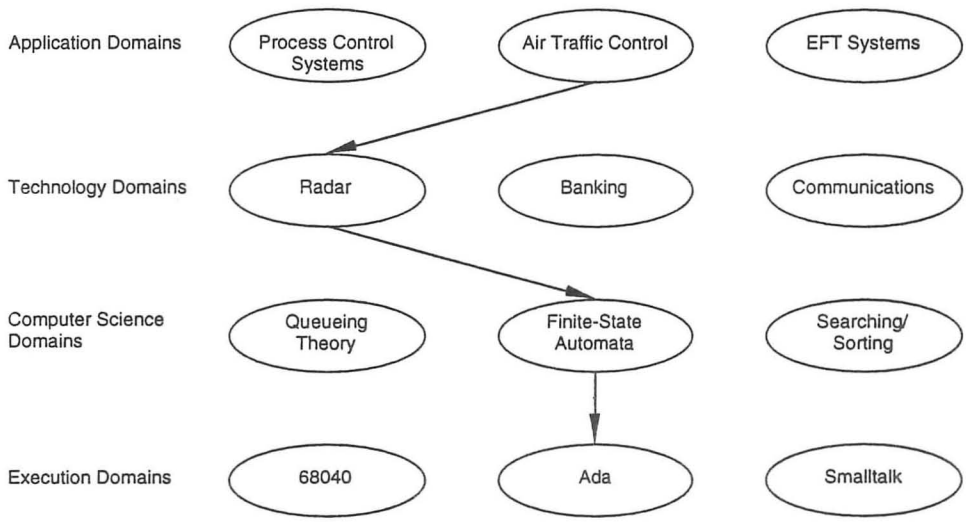


Figure 1. Domain Network

example, be obtained from other devices or via a user interface. This observation leads us to recognize the existence of multiple domains and the relationships among them.

Arango first identified the existence of domain networks [Arango, 1988]. A domain network organizes different domains into a network in which some domains are subsidiary to others. The network indicates that concepts from one domain are implemented using those from another, subsidiary domain. This division into subsidiary domains continues until all of the concepts needed to implement the application domain concepts are described in terms of "implementable domains."

Ward and Williams described an approach that is similar to domain networks [Ward and Williams, 1990] [Williams, 1991]. This approach recognizes that an application includes components from several different domains. Some of the components are application-specific; they belong to the primary or *dominant domain*. Others are application-independent; they belong to *supporting domains*. Supporting domains can be valid application domains in their own right. The distinction is that they can also be used in constructing systems from several different dominant domains. As with domain networks, these components form an hierarchy in which components from higher domains can be implemented using components from lower ones.

The top of the hierarchy is the dominant domain. Those domains that directly support the construction of components from the dominant domain are at the next level and so on. The relationship between the domains is a "uses" relationship. The "uses" relationship in an inter-domain relationship; it relates a component in one domain to its realization in another. The meaning of this relationship is the same as the "uses" relationship described by Parnas [Parnas, 1979]: domain A "uses" domain B if the correct operation of a component from A requires one or more (also) correct components from B. The "uses" relationship defines a partial ordering on the domains.

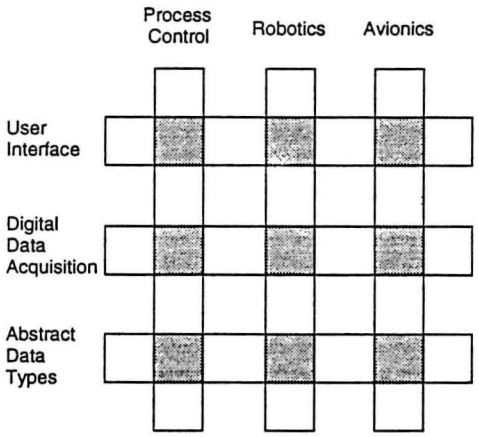


Figure 2. Horizontal and Vertical Domains

Figure 1 illustrates a domain network. Here, the "air traffic control" application domain is implemented in terms of the "radar" domain. Air traffic control is the dominant domain and radar is a supporting domain. The radar domain is, in turn, supported by the "finite-state automata" domain. Finally, the "finite-state automata" domain is implemented in terms of the Ada execution domain. These lower domains are all supporting domains.

Domains may also be classified into horizontal and vertical domains [McDonnell Douglas Missile Systems Company, 1990]. *Horizontal domains* are application-independent groupings of components. The domain of user interfaces is a horizontal one. Window-based user interfaces, for example, span application domains from banking applications to real-time systems. *Vertical domains* are application-dependent groupings of components. Vertical domains correspond to the problem domain of the primary application. Examples include: banking, industrial process control, and robotics. A vertical domain represents a product line or, at least, a family of closely related applications. Figure 2 illustrates horizontal and vertical domains.

Shlaer and Mellor [Shlaer and Mellor, 1990] describe four types of horizontal domains:

- *Application:* The application domain is the domain that includes the primary problem at hand.

- *Service:* Service domains provide general capabilities needed throughout the application. Examples of service domains include the user interface and

- *Architecture:* The architecture domain provides support for the organization of data and control. It provides a higher level of "virtual machine" than a programming language or operating system. As described by Shlaer and Mellor, an architecture domain appears to be a combination of a computational model (e.g., object-message, finite-state machines) and basic services (e.g., external device interfaces, error checking).

- *Implementation:* The implementation domain is defined by the operating system and programming language used to implement the application.

These domains form an hierarchy of abstraction with the application domain at the top and the implementation domain at the bottom. As with other approaches to domain separation, design proceeds by mapping entities in a higher domain to those in a lower one.

5 Domain-Specific Architectures

Most software developers think of reuse in terms of code-level components, such as common procedures, functions, and subroutines that are shared within an application program or even across applications. Although reuse of code-level components can be valuable, reuse of products at higher levels of abstraction (such as designs or specifications) can be even more valuable. This is because a code-level component is a concrete realization of a particular abstraction for a given target environment. Arriving at that realization requires a number of decisions that must typically be changed in order to reuse the abstraction in a new context. Thus, the potential for reuse of code-level components across a wide range of applications is limited.

Designs are at a higher level of abstraction and thus, potentially reusable in more situations. Reusable designs are based on *domain-specific architectures*. A software architecture describes the components that make up the system and the ways in which they interact. A domain-specific architecture describes the components and interactions for a family of related systems within a given application domain.

In an object-oriented design, a domain-specific architecture is known as a *framework*. A framework is a collection of classes which interact in a particular way [Deutsch, 1989] [Johnson and Foote, 1988]. The classes that make up the framework may be concrete or abstract. A concrete class directly implements some abstraction from the application domain. An abstract class provides a partial specification and implementation but defers some decisions to its subclasses. Individual applications are implemented by creating concrete subclasses for any abstract classes. The use of abstract classes thus makes it possible to specify common properties and constraints across all applications from the domain and also allows for variation among individual applications.

Perhaps the best known framework is Smalltalk's model-view-controller (MVC) framework. This framework is specific to the domain of interactive applications. The MVC framework consists of three components:

- The *model*, which stores the data and performs the behaviors of the underlying application.

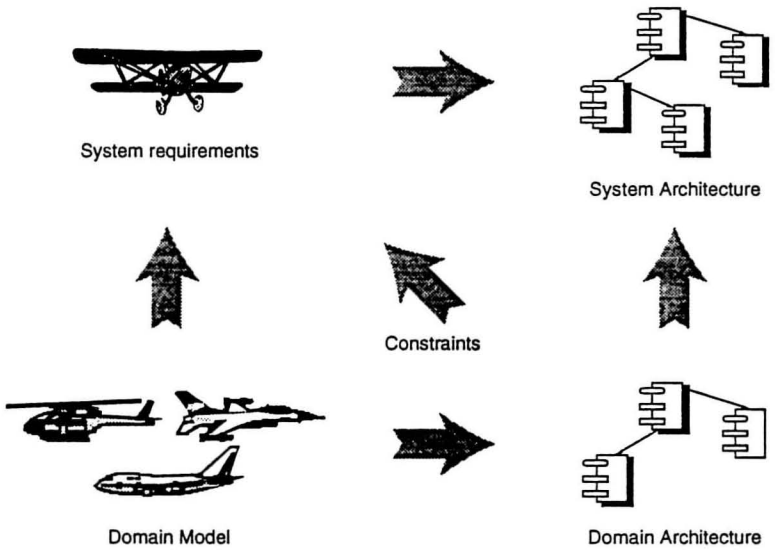


Figure 3. Domain-Specific Architectures

- The *view*, which determines how various aspects of the model are to be presented to the user in visual form.
- The *controller*, which interprets user initiated events (such as keystrokes or mouse actions), sends appropriate messages to the model, and provides communication between the model and its view.

As shown in Figure 3, a domain-specific architecture defines the common components and interactions that are to be used in implementing a set of applications within the domain. The architecture also constrains the types of new applications that can be considered. Any new system must fit into the structure defined by the architecture if it is to be cost effective [Garlan, 1994].

6 Domain-Specific Software Environments

The availability of reusable components and a domain-specific architecture suggests the possibility of constructing a *domain-specific software environment*. A domain-specific environment would provide support especially geared toward developing applications within a particular problem domain. Such an environment would provide a repository for reusable components together with support for locating, comprehending, and modifying those components to produce finished applications. Tools within the environment would be focused on composing applications from reusable components and creating new specialized components within that application domain. Figure 4 illustrates the concept of a domain-specific software environment.

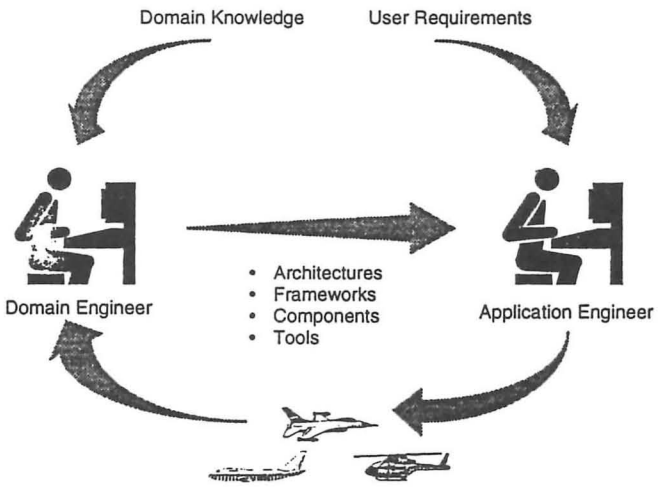


Figure 4. Domain-Specific Environments

It may also be possible to use knowledge about the domain to provide even higher levels of support. Gerhard Fischer and his research group at the University of Colorado have constructed several prototype domain-specific design environments. These prototype environments embed knowledge about the target domain. This knowledge is used to provide feedback to designers on the quality of their designs. For example, the JANUS environment [Fischer, et al., 1991] contains knowledge about design principles for kitchens that are based on building codes, safety standards, and functional preferences. When a design principle is violated, the environment identifies the problem for possible user action. While this work has, to date, not been applied to designing software artifacts, the potential for doing so is clear.

Figure 4 also suggests a software process model to complement a domain-specific software environment. In this model, a domain engineer uses application domain knowledge to create a domain-specific architecture and reusable components. These components would be used to populate a domain specific software environment. This environment would then be used by an application engineer to create individual applications to meet user requirements. There is also a feedback loop in the model. As applications are developed, experience will lead to the introduction of new reusable components and modification of existing ones to make them more useful. This will, in turn, lead to modification of the domain-specific architecture.

The model also suggests two distinct roles for developers: *domain engineer* and *application engineer*. A domain engineer designs the domain-specific architecture and creates reusable components. The individual(s) who are assigned to this role are responsible for developing the domain model, identifying and constructing reusable components, soliciting potentially reusable components from application engineers, and assisting the application engineer in reusing those components. A domain engineer should be familiar with the application domain or able to work closely and productively with domain experts. He or she should also be familiar with reuse technology. Perhaps

the most important characteristic of a domain engineer, however, the ability to think abstractly. Much of the work involved in creating reusable components and a supporting architecture is aimed at creating general, reusable abstractions from specific instances.

An application engineer creates end-user applications from components in the domain-specific environment. This individual is responsible for developing individual applications, adapting and modifying reusable components, and proposing new, potentially reusable components to the domain engineer. An application engineer should be familiar with the applications to be developed and able to work closely with customers or users to understand their requirements. An application engineer should also be able to work at a concrete level and have a proven ability to deliver a product.

7 Evaluating a Domain

Domain analysis can be performed for any application domain. In some cases, however, the return may not justify the investment of time and effort. The goal of domain analysis is to discover and exploit commonalty among applications within a domain. If there is not sufficient commonalty among the applications or the number of potential applications is too small, then a domain analysis may not be justified.

When evaluating a domain to determine its suitability for domain analysis, it is necessary to consider a number of different factors. These include:

- *breadth*: Is the set of potential applications that define the domain large enough? For systems that are truly unique or "one off" there is not sufficient payoff in terms of reusability or economies of scope to justify a domain analysis. Similarly domains that have only a few applications may also not be good candidates for domain analysis.

- *commonalty*: Do the various applications in the domain have sufficient commonalty? Even if there is a large number of potential applications within the domain, these applications must share a substantial set of objects and operations. If not, a domain analysis may not be economical.

- *maturity*: Is the domain mature? In immature domains, such as those encountered in research applications, the solutions to problems are not yet well-understood. As a result, it is difficult to identify the appropriate abstractions for constructing a domain model. In addition, objects and operations are likely to change rapidly as the domain matures.

- *stability*: Is the domain stable, or is the set of applications likely to change? The set of potential applications both defines the domain and forms the basis for the domain model. If this set is not stable, the model is likely to undergo change as the definition of the domain evolves.

- *theoretical foundation*: Is there a solid theoretical foundation for solving problems within the domain? The presence of a solid theoretical foundation

8 Summary

This paper has discussed several aspects of domain engineering. This approach is based on capturing knowledge about a set of related applications from within a particular problem domain. This knowledge is then used to exploit the commonalty among those applications to enhance reuse of software components. We have also discussed related topics such as domain separation, domain-specific architectures, and domain-specific software environments.

Domain analysis is currently a labor-intensive activity. The relevant information is held by "domain experts" who may not be skilled in constructing the necessary models. In addition, as we have noted, much of the necessary information is implicitly held and must be made explicit as part of the domain analysis process. Currently, identification of the relevant abstractions from within a given domain is a process that relies heavily on human intelligence. There are, however, indications that some aspects of the process can be automated. Goldin and Berry [Goldin and Berry, 1994] have described AbstFinder, a tool for finding abstractions in natural language text.

In addition, the construction and use of domain-specific software environments can help to ease some of the work associated with a domain-oriented approach. Although a domain-specific environment such as that envisioned in this article has not yet been constructed, the technology is available. The construction and evaluation of such an environment is a topic for future research.

9 References

[Arango, 1988]
G. F. Arango, "Domain Engineering for Software Reuse," Technical Report No. 88-27, Department of Information and Computer Science, University of California, Irvine, CA, 1988.

[Biggerstaff and Richter, 1987]
T. Biggerstaff and C. Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, vol. 4, no. 2, pp. 41-49, 1987.

[Booch, 1987]
G. Booch, *Software Components with Ada*, Menlo Park, CA, Benjamin/Cummings, 1987.

[Curtis, et al., 1988]
B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268-1287, 1988.

[Cusumano, 1991]
M. A. Cusumano, *Japan's Software Factories: A Challenge to U. S. Management*, New York, Oxford University Press, 1991.

[Deutsch, 1989]
L. P. Deutsch, "Design Reuse and Frameworks in the Smalltalk-80 System," in *Software Reusability, Volume II: Applications and Experience*, T. J. Biggerstaff and A. J. Perlis, ed., Reading, MA, ACM Press, 1989, pp. 57-71.

[Fischer, et al., 1991]
G. Fischer, A. C. Lemke, R. McCall, and A. Morch, "Making Argumentation Serve Design," *Human Computer Interaction*, vol. 6, no. 3-4, pp. 393-419, 1991.

[Garlan, 1994]
D. Garlan, "The Role of Software Architecture in Requirements Engineering," *Proceedings of the First International Conference on Requirements Engineering*, Colorado Springs, CO, April, 1994, pp. 240.

[Garlan, 1990]
D. Garlan, "The Role of Formal Reusable Frameworks," *Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development*, M. Moriconi, ed., Napa, CA, May, published as *Software Engineering Notes*, vol. 15, no. 4, 1990, pp. 42-44.

[Garlan and Delisle, 1990]
D. Garlan and N. Delisle, "Formal Specifications as Reusable Frameworks," in *VDM '90: VDM and Z – Formal Methods in Software Development*, D. Bjørner, C. A. R. Hoare and H. Langmaack, ed., Berlin, Springer-Verlag, 1990, pp. 150-163.

[Goldin and Berry, 1994]
L. Goldin and D. M. Berry, "AbstFinder, A Prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology, and Evaluation," *Proceedings of the First International Conference on Requirements Engineering*, Colorado Springs, CO, April, 1994, pp. 84-93.

[Johnson and Foote, 1988]
R. E. Johnson and B. Foote, "Designing Reusable Classes," *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22-35, 1988.

[Love, 1993]
T. Love, *Object Lessons*, New York, SIGS Books, 1993.

[McDonnell Douglas Missile Systems Company, 1990]
McDonnell Douglas Missile Systems Company, "Developing and Using Ada Parts in Real-Time Embedded Applications," McDonnell Douglas Missile Systems Company, 1990.

[Neighbors, 1984]
J. Neighbors, "The Draco Approach to Constructing Software from Reusable Components," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 9, pp. 564-573, 1984.

[Parnas, 1979]
D. L. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 2, pp. 128-138, 1979.

[Parnas, 1976]
D. L. Parnas, "On the Design and Development of Program Families," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 1, pp. 1-9, 1976.

[Prieto-Diaz, 1987]
R. Prieto-Diaz, "Domain Analysis for Reusability," *Proceedings of COMPSAC-87*, Tokyo, 1987, pp. 23-29.

[Shlaer and Mellor, 1990]
S. Shlaer and S. J. Mellor, "Recursive Design," *Computer Language*, vol. 7, no. 3, pp. 53-65, 1990.

[Ward and Williams, 1990]
P. T. Ward and L. G. Williams, "Domain Modeling," Technical Report No. SERM-012-90, Software Engineering Research, Boulder, CO, February, 1990.

[Ward and Williams, 1990]
P. T. Ward and L. G. Williams, "Domain Analysis: An Example," Technical Report No. SERM-013-90, Software Engineering Research, Boulder, CO, May, 1990.

[Williams, 1991]
L. G. Williams, "Software Reuse: Definition and Overview," in *Aerospace Software Engineering*, Progress in Astronautics and Aeronautics, vol. 136, C. Anderson and M. Dorfman, ed., Washington, DC, American Institute of Aeronautics and Astronautics, Inc., 1991, pp. 395-413.

[Williams, 1991]
L. G. Williams, "Modeling Multiple Domains," *Workshop on Domain Modeling for Software Engineering, Participant Proceedings, Thirteenth International Conference on Software Engineering*, Austin, TX, May, 1991.

聞き書き "グループウェア技術"

― Skip Ellis 教授の講義レポート ―

林 香
(SRA, Boulder)

0. はじめに

昨年の夏，われわれの会社が企画したワークショップの一環として，何人かの日本人技術者の方々と一緒に，Colorado 大学で，Clarence (Skip) Ellis 教授の講義を聞く機会がありました．その内容は，グループウェアの入門編といった感じのもので，専門家にとっては当たり前のことばかりだろうと思われますが，随所に興味深いエピソードが盛り込まれており，私自身にとっては大変楽しい講義だったので，簡単に SEA のみなさんにも報告しようと思い立ちました．

Clarence (Skip) Ellis 博士は，グループウェアや CSCW，およびワークフローの分野で 20 年以上研究を続けて来られた方であり，いろいろな概念やモデル，製品のアイデアを提唱したパイオニア的存在です．Colorado に来られる以前は，IBM, Bull, MCC, Xerox PARC, AT&T (Bell Labs), Los Alamos Scientific Labs などの企業で働く一方で，MIT, Texas 大学, Stanford 大学, そして中国の上海交通大学でも教鞭をとられたこともあるというきわめて多彩な経験を持った人物です．そのせいか，講義は豊富な実例をまじえた大変興味深いものでした．余談ですが，Skip というのは彼の仇名で，グループウェアの世界では Skip Ellis の名で通っているようです．

では，本題に入りましょう，講義の大筋は以下のようなものでした．

1. 用語の定義
2. 歴史的背景
3. 例題，ケーススタディ
4. グループウェアの設計とアーキテクチャ
5. 今後の展望

以下，この筋にそって，Skip 教授の一人称表現で書いていきます．講演を録音していたわけではありません．このレポートは，私の記憶とスライドのコピーだけを頼りに書き下ろしたものです．細かな事実関係に間違いがあるのではないかと恐れますが，あえて Seamail に投稿したのは，講演内容の興味深さを少しでもみなさんにお伝えしたいと思ったからです．

それでは，Skip 先生，どうぞ．

1. 用語の定義

最初に，コトバの定義をします．まず，次の２つのコトバの違いをはっきりさせておきましょう．厳密に区別せずに同じようなものだとして使っている人もいますが，私には明確な違いがあるように思われます．

２つのコトバとは,"グループウェア" と "CSCW" です．"グループウェア" とは，共通の任務やゴールを達成するためにグループとして働いている人びとを支援するための，科学技術に裏打ちされた，グループ全体で共有できる環境を提供する「システム」です．一方,"CSCW" とは，グループを構成するメンバーがどのように互いに影響し合うかを考察し，科学技術 (特にコンピュータ技術) がグループの活動をいかに支援できるかを追及する研究分野です．それは，社会心理学，認知科学などの多分野にまたがった学際的な「研究」です．

私は，自らをグループウェアの研究者だと思っています．グループ活動を支援するシステムやツールの研究開発が私の主なテーマだからです．

用語としては，これらの他にも，Organizational Computing，共同作用理論，グループ意志決定システム，Workgroup Computing など，いろいろな buzzword が発明されていますが，ここでは個々の詳細には触れません．

それよりもグループウェアの定義をもう少し深く考察してみたいと思います．まず，共通任務遂行の支援とは何でしょうか？ 複数の人間が同時にコンピュータを利用できるようになっているタイムシェアリング･システムは，共通任務遂行を支援するグループウェアなのでしょうか？ また，観点をまったく変えて，もしここに，複数の人間がソフトウェアの仕様や論文をレビューできるシステムがあったすれば，それがグループウェアなのでしょうか？

共通任務の支援という点に関していえば，この２つの例は，両極端に位置しているといえます．すなわち，タイムシェアリング･システムは，非常にレベルの低い「共通任務遂行支援」を行なうものであり，ソフトウェア･レビュー･システムは，大変レベルの高い「共通任務遂行支援」を実現しているといえます．そして，両者の間には，白黒では割り切れないいくつもの細かな段階が存在するのです．

このようにグループウェアとは，きわめて幅の広い概念です．共有環境という概念も同様に幅が広いものです．参加メンバーがそれぞれの役割を達成するには，相互に連絡しあう必要がありますが，そのための手段はさまざまです．レベルの低い方法としては電子メイルがあります．その対極には電子教室，電子会議のようなものが上げられます．

電子会議のイメージは，参加メンバーそれぞれに必要なツールが用意され，たとえば議長には議事進行に必要な道具が，秘書には議事録作成の道具がそれぞれ用意されており，しかも１つの黒板を前に議論しているにもかかわらず，全員が同じ場所にいる必要がないという環境を，電子的に実現したものです．これと電子メイルとの間にもいくつもの細かな段階が考えられます．

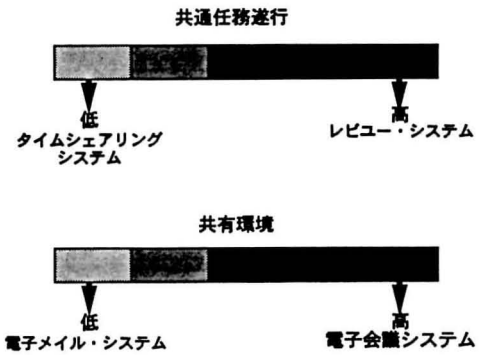


図 1. グループウェアにおける２つの次元

このように考えを進めていくと，みなさんは次の点に気がつかれると思います．つまり，グループウェアは空間と時間のマトリックスで分類できるということです．

(1) 対面型　　(同じ時間，同じ場所)
(2) 非同期型　　(異なる時間，同じ場所)
(3) 同期分散型　　(同じ時間，異なる場所)
(4) 非同期分散型　(異なる時間，異なる場所)

それぞれについての細かい説明は不要だと思いますので，簡単な例を上げておきます．

(1) 対面型 (同じ時間，同じ場所)
　電子会議支援システムが整備された会議室．
(2) 非同期型 (異なる時間，同じ場所)
　たとえば編集履歴を参照しながら，他のメンバーが設計しコンピュータ上に残していった図面を変更する．
(3) 同期分散型 (同じ時間，異なる場所)
　コンピュータ･ネットワークを通じた電子会議．
(4) 非同期分散型 (異なる時間，異なる場所)
　論文の審査を，世界中に散在している委員が他の委員の結果も参考にしながら，時間を見つけては審査し，結果を全員に連絡することができるような論文審査システム．全員の審査結果がでると，委員長に連絡が行き，これを何度か繰り返すこともできる．

	同時	異時
同場	対面型	非同期型
異場	同期分散型	非同期分散型

図 2. グループウェアの時空間マトリックス

ここで，ちょっと視点を変えて，グループウェアや CSCW の歴史的な背景に触れ，いくつかの例題をあげて，さらに深くこの分類を理解したいと思います．

2. 歴史的背景

大雑把にいって 1965 から 1975 年までの間，コンピュータは組織活動を支援するために利用されていました．すなわち，メインフレーム･コンピュータ上に情報管理システムが構築され，組織全体の管理のために使われていたのです．その次の 10 年間の主流は個人の活動の支援です．マイクロもしくはミニコンピュータ上に個人活動を支援するシステムが開発され，コンピュータと人間との相互作用が研究されました．そして，コンピュータ･ネットワークの発達につれて，1985 年以降グループ活動の支援が注目されはじめ，CSCW やグループウェアといったコトバが生まれてきたのです．

以上が大まかな歴史的事実ですが，では，なぜ組織活動の支援が最初に行なわれ，グループ活動の支援が最後になったのでしょうか？　１つの大きな理由としては，メインフレーム･コンピュータがきわめて高価であった (いまでも高価です) ということが上げられます．つまり，組織全体を支援することで，やっとその高価なコストに見合う利益を得ることが可能だったというわけです．グループ支援が最後になったのは，ワークステーションやネットワークなどの基幹技術がまだ十分成熟していなかったというのが最大の理由ですが，グループという概念が，非常にあいまいで定義しにくい集合単位であることも，個人支援が優先された大きな理由として上げられます．

さて，その後，基幹技術の成熟と歩調をあわせて，グループ活動支援の研究が盛んになってきました．それは，単に

機が熟したというばかりではなく，実は，ソフトウェア開発者自身がグループ活動支援の必要性を強く感じていたからでもあります．Tom DeMarco が指摘したように，ソフトウェア設計者の時間の 70% は，会議·相談·質問·レビューなど，他人との共同作業に費やされています．プログラマたちはどうかといえば，かれらも決してプログラミングやテストに集中しているわけではありません．私の調査では，プログラマもその作業時間の大部分を，チームメンバーとのコミュニケーションに費やしているという結果が出ています．

ワークステーションやネットワークなどの基幹技術が成熟し，ソフトウェア技術者が自分たちのためにグループ活動の支援環境が必要であると気づいたからこそ，CSCW やグループウェアの研究が始まったのです．そのオフィシャルな期待効果としては，チームワーク力の向上，情報伝達の迅速化，管理負担の軽減などが上げられていますが，本音は，みんながそれぞれ自由な時間が欲しかったからなのです．

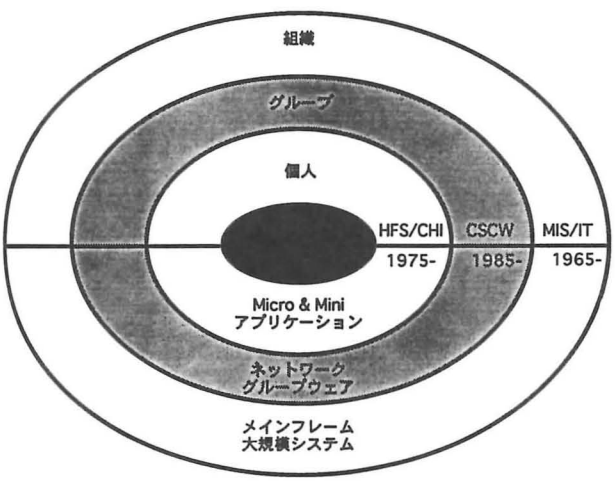


図 3. グループウェア/CSCW の位置づけ

3. 例題とケーススタディ

3.1 Lotus Notes

では，グループウェアの実例を見ていきましょう．その中で，もっとも実用的であり，商品化されているという点で，いま最も注目され話題を集めているのは Lotus Notes でしょう．機能的には，何も新しいものはありませんが，電子メイル，電子会議，表計算，固定フォームのエディタ，その他のグループ活動支援に必要なツールがうまく統合され，共有環境として提供されています．さまざまな書類をオンライン化し，ペーパレス·オフィス環境を実現することができるわけです．

ビジネスとしても大変成功しているのですが，その最大の秘訣は販売戦略にあります．発売当初，Lotus Notes は個別売りや小数での販売がなされず，組織全体にたいして大量の導入をする形でなければ購入できませんでした．その理由は，グループウェアは全員が使わなければ効果を発揮できないからです．たった一人のメンバーだけが電子メイルを使っている状況を想像してください．自分に対するメモをメイルする以外の効果はありませんので，グループ活動を支援しているとはいえません．それほど極端でなくとも，グループ内の半数しか電子メイルを利用しない場合には，全員への連絡はメイルだけでなく，他の手段を講じなければならないので，無駄な労力が必要になり，効果は大幅に減少します．Lotus Notes は，この点を重視して，大量導入かゼロかという戦略で販売され，成功をおさめました．

実際のところ，Lotus Notes よりすぐれた機能を部分的に備えた競合商品はいくつかあるのですが，総合的な機能のバランスのよさと販売戦略によって優位をかち得ています．また，この販売戦略だと，顧客数が絞られるので，顧客からの不満や改良要求が出れば即座に対応でき，サービスもよくなります．事実，Lotus Notes の初期のバージョンにはカレンダ機能がありませんでしたが，大口顧客からの要求により，2 週間後には機能拡張がなされ，次期バージョンがリリースされました．

Lotus Notes はこのようにたしかに成功しているのですが，問題がないわけではありません．すべての要素ツールを自前で作成しているため，個々のツールを見るとどれもが未成熟です．とくにメイル·ツールには不満が多く出ています．受信メイルの整理機能などが非常に劣っているからです．個々のツールの機能を如何に向上するかが，これからの課題だといえましょう．

また，ペーパレス·オフィスを実現するには，米国ではサイン，日本では印鑑をコンピュータ上で取り扱う必要があるわけですが，この点は今のところ研究段階です．いくつかの面白い研究を紹介しましょう．1 つはオフィシャルなサインを指紋と一緒に登録し，キーボードの横に親指をいれる穴をつけて，指紋を押しつけながら名前をタイプすると，名前と指紋がチェックされ，OK であれば電子的なサインが記録されるというものです．他には，特別な椅子を用意し，腰の形で個人を特定するという研究もあります (笑い)．

Lotus Notes に関して，もう一点，私の経験をお話ししたいと思います．論文審査のための異時間異場所グループ活動支援共有環境として Lotus Notes を利用した例です．簡単にいってしまえば，ワードプロセッサとメイルを合体させ，査読結果の管理機能が追加されたものとして利用したわけです．この例で私が強調したいのは，このような機能の統合面ではなく，グループウェアにとって不可欠な特性

についてです．その特性とは，融通がきくこと，しかも，ダイナミックに利用者の都合を聞き入れることです．

世界中に散らばっている査読者は，論文審査を依頼されるほどの能力のある人ですから，当然忙しいことが予想されます．突然，出張してしまうこともあるでしょう．これに対して，査読用グループウェアが，査読者全員の評価が出るまでは委員長に結果を知らせないというように，堅く作られていたらどうなるでしょう．査読のプロセスは行き詰まってしまいます．

私たちは，Lotus Notes を CSCW'94 のプログラム委員会での論文審査に利用したのですが，実際にこの問題に遭遇しました．ある委員が出張のため長期不在になり，かれからの査読結果が届かなかったのです．査読の管理を司っているサーバマシンには，全員の査読結果が揃ったらそれを全委員に配付するように指定がなされていたため，初めのうちはなんの連絡もありませんでした．そのうちに，この事実に気がついたサーバマシンの管理者が，システムに対して，不在の委員の査読結果は第一ラウンドではスキップし，第２ラウンドから取り入れるようにダイナミックに指示を変更したおかげで，以後ことはスムーズに進みました．

Lotus Notes がこのような指示を受け付けるように作られていたので，ことなきを得たわけです．Lotus Notes の競合商品に，WordPerfect と EditPerfect を統合して，同様の機能を発揮するものがありますが，商業的には Lotus Notes ほど成功していません．なぜなら，Notes のよう融通がきかないからです．ダイナミックに利用者の指示を変更できないのです．グループ活動を支援する以上，数多くのさまざまな人を相手にするのですから，いつ何が起こるかわかりません．したがって，システムは常に融通がきくように考慮して作成する必要があるのです．

3.2 GroupSketch

GroupSketch とは，最小限の機能を持ったマルチユーザ向けのスケッチパッドです．複数人の参加者が同時に同じウインドウに対して，描画，作文，削除，ポインティング，ロード，セーブ，クリアなどを行なうことができます．WYSIWIS (What You See Is What I See) を実現し，その効果を計るためにプロトタイプされたもので，Canada の Calgary 大学の Greenberg 等の研究成果です．「同じ時間異なる場所」型のグループウェアの例だといえます．

離れた場所にいる人びとが，同じ画面を見ながら相談しつつ何かを設計をするように考えられたものですのから，各人にはマイクとスピーカも用意され，音声による会話ができるようになっています．プロトタイプは，意図的に，何の拘束もないように作成されました．したがって，最初のうちは，利用者全員が混乱し，欲求不満になりました．何かをタイプするそばから削除されたり，タイプ中に他の人に画面をスクロールされて，自分が何をしているのかわからなくなったりしたからです．しかし，次第に利用者自らがルールを決め，混乱を解消するようになりました．

この現象は，私自身が手掛けた GROVE でも起こっていますので，詳細については GROVE の話の時にふれます．ここでは，簡単に結論づけておくだけにとどめますが，重要なのは，たとえ当初はどんなに混乱が生じようとも，利用者の自由を奪うべきではないということです．グループウェアにおける利用者サイドの混乱を解消するさいの原則は，システムが利用者を縛るのではなく，利用参加者自身がルールを見つけだせるように，可能なかぎりの自由を与えておくべきだということです．何が起きているのかは全員が見ているのですから，自分たちでルールを作り出すのはたやすいことであり，それが一番効率的なのです．

3.3 Cruiser

アメリカの都会で生まれ育った人間は，「クルーズする」といわれると，それだけで何のことかピンときます．私もシカゴ生まれなので，クルーズといえば，週末の真夜中，町なかで何が起きているのか車であちこち見て回ることだとわかります．Cruiser は AT&T ベル研で開発された電子版のクルーズです．利用者は，車の代わりに，カメラとマイクとディスプレイ･モニタを使います．仕事が終わった夕方，あなたは自分の事務所のあるビルや，遠隔地の (もしかしたら海外の) 事務所のあちこちをクルーズし，情報交換や相談あるいは雑談をすることができます．グループウェアの思想では，無駄話も重要な情報交換です (この点については，面白いエピソードを後でお話ししましょう)．

あなたのお相手が不在であれば，画面上には閉まったブラインドが現われます．その時には，場所を移しましょう．半開きのブラインドは，相手がいるにはいるが，邪魔して欲しくないというサインです．ブラインドが開いていれば，向こうも話相手を求めているという意味です．参加者は何人でも構いません．興味ある話題であれば参加しましょう．

このようなアプローチに効果があるという主張は，次のような経験に裏打ちされています．それは，無駄話も重要な情報交換手段であるとの主張の裏付けにもなっています．その経験とは，スニーカ･ネットワークにコンピュータ･ネットワークが及ばなかったということです．詳細をお話ししましょう．

ある科学薬品会社はミネソタ州に事務所を持っています．事務所は４つのビルに分れていて，書類の転送が必要な場合には，秘書が歩いて運んでいました．ミネソタ州の冬は大変寒く，秘書がかわいそうだというので，書類をオンライン化しビル間およびビル内にネットワークを引きました．

実際のところ，秘書がかわいそうだというよりは，事務の効率化が管理者の第一の目的だったわけです．結果はどうだったのでしょうか．事務能率は低下してしまいました．なぜでしょうか？

実は，４つのビルはそれぞれ，営業，製造，研究，事務の各部門に分れて使われていました．各部門間での書類の配送を秘書が行なっていたわけです．営業部長の秘書は，研究部門のビルに行くと20分，また，事務部門では１時間以上も姿を消してしまっていました．研究部門では友だちの研究部長秘書と世間話をし，事務部門ではボーイフレンドと会っていたのです．その際，彼女は何気なく，重要な情報交換を行なっていたのです．

ふだん，研究部長と営業部長はあまり情報交換をしませんし，また，事務部門への適時適切な連絡が営業から行なわれていたわけではなかったのですが，かの女が知らず知らずのうちに，それをカバーしていたのです．研究部長秘書との世間話が営業戦略に重要なヒントを与え，また，事務部門にいる彼女のボーイフレンドは原材料仕入れの担当だったので，近々納入が必要な原材料について，デートのついでに念おしをしていたというわけです．

秘書たちが歩き回って情報を伝達する，つまり，スニーカを使ったネットワークが，ハイテクを使ったコンピュータ･ネットワークより勝っていたという，ちょっとでき過ぎたお話でした．この逸話からわれわれが学ぶべきことは，グループウェアを開発導入する際には，現状分析を慎重に行なわなければならないということ，そして，一見無駄に見えることが非常に役に立っている場合が多く，これを見逃してはならないということです．Cruiserは，こうした経験にもとづいて開発されたのです．

3.4 ClearBoard

ClearBoardは，NTT研究所の石井裕さんが提唱/研究されたものですから，ご存じの方も多いと思います．簡単にいえば，その基本的なアイデアは，透明なボードを間にはさんで向かい合い，ボードに描画しながら会話すると，相互理解が深まるというものです．CSCWとしての研究ですので，画面の背景に対話相手の顔が映しだされ，その上での描画，しかも各人がそれぞれ別のレイヤに描画できる環境がプロトタイプされています．

それでは，なぜ，透明なボードをはさんで描画･対話を行なうと，相互理解が深まるのでしょうか？　石井さんの主張は，対話相手の視線を認識しつつ対話すると理解が深まるというものです．人間は，画面上の図形と対話相手の視線という２つのものを同時に認識できるという実験結果が発表されています．大変ユニークで興味深い研究だと思います．CSCWの研究対象範囲の広さを示すよい例でもあります．

3.5 Information Lens

次は，発想は単純ですが，大変実用的で役に立っているツールの例です．一言でいえば，受け取った電子メイルを自動的に分類整理してくれるツールです．メイルの属性，たとえばFrom，subject，topicなどの内容により，事前にスクリプトで指定しておいた通りに，分類格納したり，再配送してくるというものです．スクリプトの例は以下のような単純なものです．

```
IF
Subject:LENS
Topic:BUGS
.....
THEN
Move to:LENS-BUGS
```

この例をお見せしただけで，みなさんには，そのアイデアがきわめて単純なものだということがおわかりいただけると思います．しかし，情報の共有や協調作業のために非常に役に立つものです．詳細について，ここでお話しする時間的余裕がないので，参考文献を１つお知らせしておきます．

The Information Lens: An intelligent system for information sharing and coordination (Malone et al., Technological Support for Work Group Collaboration, pp65-88, edited by M.H.Olson, 1989, Hillsdale, NJ: Lawrence Erlbaum)

3.6 電子会議室

次の例は，これまでに私が関係した仕事の中で最もエキサイティングなものでした．会議室のテーブルに，会議参加者各人のためのマッキントシュを埋め込み，ネットワークでつないで，顔突き合わせ型の会議を支援しようとするものです．もちろん，壁に埋め込まれた大型のスクリーンがネットワークにつながり，会議参加者が入力した内容を映しだせるようになっていることは，いうまでもありません．

このシステムが開発された動機は，従来の会議室の設備では，秘書が会議の内容を完璧に記録することは不可能であり，いくつもの重要なアイデアが失われていること，また，会議の終了時にすぐ議事録ができないために，後でそれ配付されたときに，誤解や異論がいくつも見つかることなどを未然に防止しようというものです．つまり，会議終了時に，参加者全員の合意を得た議事録を作れるようにしたいわけです．もちろん，そればかりではなく，会議そのもの質を向上する，よい結論を早く，的確にだせるようにすることも期待されました．

このシステムは，共有スクリーン，共有ウインドウを使っ

た共有環境の例になるわけですが，私たちは，ウインドウの属性をパブリック，プライベート，共有の３種類に分けました．たとえば，個人の意見をプライベート･ウインドウで作成し，パブリックに切り替えると壁のスクリーンに映しだされるというような形です．共有ウインドウという名前は，みなさんを混乱させるのでないかと心配なのですが，限られた二人の間のみで共有できるウインドウを指します．共有環境という場合の’共有’よりは狭い意味で使っています．

電子会議室の効果を計るために，私たちは会議室の横にモニタ室を作り，会議参加者各人のディスプレイの内容を映しだすモニタを設置し，いくつもの実験を行ないました．その結果，この２人だけの共有ウインドウが非常に役に立つことがわかりました．ある人がパブリック･モニタを使って説明をしている間，多くの人が他の参加者に内緒の確認･相談をし，それが，議事進行に役だっていたのです．実に興味深い発見でした．

3.7 GROVE (The GRoup Outline Viewing Editor)

GROVE はリアルタイム･グループウエアの一例といえます．いわゆるアウトライン･エディタを，複数人で同時に操作/編集できるようにしたものであり，MCC において，実験的に研究試作されました．入力されたテキストの内容，マウスのポインタ，カーソルやスクロールのの位置などが，まったく同じアウトライン･エディタのウインドウとして，参加者全員のディスプレイに表示され，だれでもが，自由に入力･編集･スクロールなどが行なえるようになっている WYSIWIS (What You See Is What I See) 型ツールの１例です．

このシステムの開発には，私自身深くかかわっていましたので，いくつかの有意義な発見についてお話しできると思います．さて，GROVE は実験的な試作品であると申し上げましたが，いったい何を目的としていたのかといえば，この種のグループ･ウエアがチーム･メンバーの密な結びつきを促進し，それによって，試行錯誤を繰り返しながら設計を行なう課程を支援できるのではないかという仮説を証明するためでした．

システムの試作にあたってもっとも重視されたのは，利用者の自由を束縛しないということでした．便利さや機能とのトレイドオフを理由に，利用者の自由を妨げることがないよう，優先的な配慮がなされました．たとえば，GROVE の要求定義には，高度な対話性や並列性，同時更新，データの複写，コントロールの分散など，同期分散型のグループウエアに必須の特性に加えて，ロックのメカニズムは "導入しない" という点が付け加えられていました．ロックによって利用者の自由を奪うことを恐れたわけです．利用者に多少の混乱を与える可能性があるとしても，ロック･メカニズムのせいで欲求不満を与えることのないように考慮がなされました．

さて，実験の結果はどうだったのでしょうか？

GROVE の試作品が完成したのち，私たちは自らの手で評価を行ないました．その結果，得られた結論は以下の通りです．

(1) 参加者全員が概念的には１つのウインドウを自由に操作できるわけですから，当然，混乱が生じました．入力途中で突然画面がスクロールされてしまい，何を入力していたのか分からなくなってしまうといったことが多発しました．最初のうちは，まさに混沌状態だったのです．しかし，Group Sketch の場合と同様に，利用者自らがルールをきめ混乱を解消するようになり，事態は収拾されました．どのような取り決めがなされたかについては，あとでお話しします．

(2) 利用者間での衝突は驚くほど少ないものでした．セッションの参加者全員が，他の参加者の活動を潜在意識下で常に気にかけていたというのが，その理由です．ある参加者は次のようにいっています．

「いわゆる，ブレーン･ストーミングなどに参加すると，私はできるだけ早く自分の考えをまとめようと努力し，他の参加者の言動には余り注意を払いませんでした．しかし，今回 GROVE を使った経験では，まず第一に，参加者がタイプする内容に重複がほとんどなかったので，私は他人がタイプした内容を無意識のうちに読んでいました．また，同時に同じ項目について編集作業をした場合でも，作業者間での編集の衝突はほとんどありませんでした．お互いが，他者の作業空間を侵略しないように常に心がけていたからです」

(3) グループ編集の機能は，作業を分担し同時並行でアイデアをまとめるのに，大変役立ちました．作業者間での協調という意味で，非常に興味深いのは，ツールを使うために，全員で合意した約束を作り上げたことです．GROVE はグループ作業ができるアウトライン･エディタであると申し上げましたが，アウトライン･エディタということは，いろいろなアイデアを，階層構造で記述/整理できるということです．ところが，GROVE には，階層の中のサブ･トリーを簡単に移動する機能がありませんでした．そこで，あるグループでは，既存の文章を移動したいと思った場合には，まず空の行を移動先に作り，移動内容を全員で確認したのち，各人が責任をもって必要なカット & ペーストを同時並行的に行なうという方法をあみだしました．このグループ活動は，１人の人間にサブ･トリーの移動を任せるよりも，はるかに効率的でした．

(4) 実験に参加したすべてのグループが，意義深いアウト

ラインを作成することができました．重要な点は，アウトラインがセッションの参加者全員の協力による成果物であり，それがGROVEでのセッションの終了時点ですでにできあがっていたということです．また，アウトライン作成プロセスも，以下のようにバラエティに富んだものでした．

- ·独立入力： すでに入力されている内容や，議論の内容に各人が少しだけ注意を払いながら個別に情報を入力する．
- ·反映入力： 主に他者の入力内容にたいして，各人が個別にコメントしたり，内容の追加や変更を行なう．
- ·合意入力： 議論の結果の総意を入力，変更する．
- ·分割入力： 各メンバーがそれぞれの編集部分を分担し，内容の洗練，再構成を行なう．
- ·記録入力： 口頭での議論の内容の意訳を入力し，確認する．

このようなルール決めにしたがった結果，議事録担当を決めて入力を任せる場合に比べて，はるかに情報の喪失が少なくなり，セッションの終了時点で，意味深く分かりやすいアウトラインを作成することができました．実験参加者はその結果に満足し，このツールがもたらす高い生産性を認めました．

(5) 新規利用者が自然とツールの機能／利用法を学ぶことができました．いわゆるWYSIWIS で，セッションの参加者全員が同じ画面を見ているわけですから，自然に他人のツールの利用法を学習でき，知らず知らずのうちにツールをマスターできてしまうわけです．

試作前に仮定していた，チーム·ワークの向上や，試行錯誤的設計の支援という点でも，参加者全員から，有効かつ効率的であったという評価を得ました．予期せぬ結果としては，GROVEの利用者が，もはや，シングル·ユーザのツールでは満足できなくなってしまったということがあげられます．一度，グループウエア·ツールを経験してしまうと，他のすべてのツールにも同様の機能が欲しくなってしまうようです．

GROVEの試用実験のあとで，あるグループは，共同で発表用のOHPスライドを作成しました．スライド作成のために，かれらは，あるドキュメント編集システムを利用していたのですが，そのシステムは共有「デスクトップ」の機能を備えていたにもかかわらず，シングル·ユーザ用のツールでしかありませんでした．各人が適時スライドを編集できるわけでも，同時にスライドを複数人で見ることができるわけでもなかったのです．

グループのメンバーは，いつもファイルをセーブしたり，クローズやオープンを繰り返したりしなければなりません．これでは，複数人での文書作成を支援しているとすらいえません．システムが記憶しているのは，だれが最後にファイルをセーブしたのかということだけだったからです．このシステムは，豊富なグラフィックスやフォーマッティング機能を備えているにもかかわらず，かれらにとっては不十分であり，全員が，GROVEのような協調編集機能がスライド作成ツールにも必要だと訴えました．

4. グループウェアの設計とアーキテクチャ

さて，次に，グループウェア，特に小·中規模のグループを対象にしたリアルタイム·グループウェアの設計上の課題についてお話しします．現在，グループウェアの設計者が直面している主要な課題は，グループ·インタフェース，グループ·プロセス，そして，同時性管理です．それらのすべてについてお話しする時間はありませんので，各テーマごとに，いくつかの重要と思われることからを選んでお話しします．

4.1 グループ·インタフェース

グループ·インターフェースの特徴は，いうまでもなく，単一のユーザではなく，複数のユーザによってコントロールされるという点にあります．GROVEのグループ·ウインドウや，リアルタイムのコンピュータ会議システム，複数人で行なうゲームなどが，例としてあげられるでしょう．

このようなグループ·インターフェースに関して，システムの設計上問題になるのは，複雑性の管理です．シングル·ユーザにくらべるとセッションはより活発になり，同時性もはるかに高くなりますので，これをきちんとサポートできなければなりません．

また，次のような疑問もあります．シングル·ユーザ·インタフェースの技術や概念はグループ·インタフェースの構築に役立つのだろうか？ たとえば，複数人のユーザが，同時にスクロールバーを操作できることに意味があるのだろうか？ それとも，それは単に混乱を招くだけなのだろうか？ といった種類の疑問です．

4.1.1 WYSIWIS

グループ·インタフェース構築のための１つのアプローチに，WYSIWIS（What You See Is What I See）があります．その名が示すとおり，すべての参加者に同じものを見せることで，コンテキストの共有を保証しようとするものです．共有コンテキスト (Shared Context) とは，複数人のユーザに対して操作が可視化されているオブジェクトの集合だと，定義できます．共著支援システムで編集対象となっているドキュメントや，電子教室における仮想黒板などがその例です．WYSIWISは，たしかかにコンテキストの共有には大いに役立ち，具体化も比較的簡単なのですが，参加者の

自在性/融通性を阻害するという欠点があります.

私の経験によれば, たとえば, ウインドウの位置や大きさをユーザ別にコントロールしたいとか, ウインドウ内の情報をカスタマイズして表示したいなどの要求は, グループウェアの実験的な利用者から, 必ず出てきます. これを許すと, 厳密な意味でのWYSIWISとはいえなくなってしまいますが, 実際のユーザにとっては大変不便なことですので, 厳密な(strict) WYSIWISに対して, ゆるやかな(relaxed) WYSIWISという概念が, Stefik等によって提唱されています. 詳しいことは彼らの論文(Stefik M. et al - WYISWIS revised: Early experiences with multiuser interfaces. ACM Trans. off. Inf. Syst. 5,2 Apr. 1987, 147-186)を読んでいただきたいのですが, 簡単に説明すると, かれらは, 次の4つの点で, WYSIWISをゆるやかにすることができるのと提唱しています.

それは, 表示空間(WYISWISが適用されている表示対象), 表示タイミング(表示内容を同期させるタイミング), サブ·グループ化(相互に関連の深い参加者だけでサブ·グループを構成する), そしてビューの一致(表示情報の視覚的な同一性)の4つです.

4.1.2 グループ的視野と混乱

よいグループ·インタフェースは, グループの活動全体を表現できると同時に, 無用な混乱を回避することができなければなりません. あるユーザがグループ·ウインドウを勝手に開いたり, 閉じたり, スクロールしたり, はては, 他の利用者が作業をしているオブジェクトを変更したりすれば, 全体を混乱させるだけです.

これは, 明らかに, シングル·ユーザ·インタフェースとは異なる点です. シングル·ユーザ·インタフェースでは, いかなる画面上の変化も自分の操作によってのみ引き起こされるという, 暗黙の了解があります. なにかのはずみに, 編集中の文章が, 突然消えてなくなってしまっても, ユーザは, 自分が何か誤操作をしたのではないかと疑えばよいだけです. これに対して, グループ·インタフェースでは, 表示内容が突然思いがけない変化をしたとき, ユーザは, 他人が何かしたのだというふうにはすぐには思い至らずに, 混乱してしまうのです. これまでの作業習慣のせいで, 他の利用者を常に意識下に置いておくことが, 簡単にはできないからです. 同じ理由から, 他人が作業中のコンテキストを間違って操作してしまうことも起こりえます.

この種の問題を解決するよい方法はないのでしょうか? 単純な答えとしては, 他者に影響がありそうな作業を行なう前には, 必ずその旨を全員にアナウンスするという方法が考えられますが, 頻繁になると煩わしいだけです. もうすこし有効な手段としては, 利用者全員の作業の様子を, 逐一表示するリアルタイム·アニメーションが考えられます. たとえば, 何人もの利用者が同時に文章の入力や変更を行なっている場合に, それぞれの文章の1文字づつの変化をハイライトさせて, 全員の画面に逐一その変化を表示するというようなことです.

しかし, これはこれで, 新たな問題を産み出します. まず第一に, 実現のための計算コストが非常に高くつき, 特別なワークステーションが必要になります. また, この種の同時並行操作を的確に表示するための有効なメタファーがまだ見つかっていないことも問題です.

さらには, いかにしてこの問題を解決するにせよ, 処理速度と連続性という両方の要求を満たさなければならない点も課題となります. つまり, システムの即応性を維持しつつ, 他の利用者への適時かつ適切な途切れのない状況報告も, 同時に実現しなければならないということです.

4.1.3 グループ力学

グループ·インタフェースは, 利用者全体のシステム利用形態に適応しなければなりません. シングル·ユーザ用のテキスト·エディタならば, 文字の挿入や削除につれて, 文字を表示したり, 消したりするという単純なインタフェースで十分なのですが, マルチ·ユーザ用のテキスト·エディタの場合には, われわれがGROVEで経験したように, 独立入力, 反映入力, 合意入力, 分割入力, 記録入力などの様々な利用パターンに対応できなければなりません.

「どしゃ降り(cloudburst)」モデルと呼ばれるマルチ·ユーザ·テキスト編集方法があり, それは, グループ·インタフェースに必要な2つの技術を採用しています. その1つは, 文章が「歳をとっていく」というものです. つまり, たとえば, 入力されて間もない文章は, 明るい青色で表示され, 時間の経過と共に, 徐々に, 黒くなって行くといったものです. 第2の技術は, だれかの手で文章に何らかの変更が加えられると, すぐさま, 変更の様子が, その文章の原作者の画面に表示され, その他の利用者の画面には, もとの文章の上に「雲」が現われて, 原文に何か変更が加えられていることを示すというものです. 「雲」の位置と大きさで, 変更箇所と変更の度合いを現すようにしています.

文章の変更がしばし中断すると, 「雲」が消え, 変更後の文章が明るい青色で画面に表示され, 徐々に黒くなっていきます. 自分の役目があり, それに集中しているユーザは, 他者が行なっている変更をいちいち気にしてはいられないという研究結果が, このインタフェース·モデルの前提となっていますので, わずらわしくない程度に, 簡潔な形で, 他者の動きを知らせるにとどめているわけです.

4.1.4 画面空間の管理

シングル·ユーザのアプリケーションにとっても, スク

リーンの大きさに制限があることは問題ですが，グループ・インタフェースになると，どのユーザも自由に他のユーザの画面にウインドウを開けるのですから，さらに深刻な問題が生じ，ウインドウ多発を管理する技術が必要となります．

1例を上げれば，特定のタスクに関連するウインドウを一つの「部屋」に押し込めておく方法が考えられます．共同作業の参加者は，「部屋」から「部屋」へと渡り歩いたり，ときには，他の参加者の手によってテレポートさせられたりします．「部屋」の中に入ると，その部屋に集められているすべてのウインドウが一斉に開かれるというわけです．

ある特定のユーザに，ウインドウの管理を任せてしまうという方法もあります．他の参加者は，その管理のもとに，各自の作業をすることになります．この方法は，経験の浅いユーザにとっては有効だと思われます．

4.2 グループ・プロセス

たとえばコード・ウォークスルーのように，複数人が参加し，かつ比較的内容が明確に定義されている作業のことを，グループ・プロセスと呼んでいます．グループ・プロセスは，グループの協調作業や並行作業の度合いを強めてくれますが，メンバー間の調整のための負荷が大きすぎるために，その効果が認められにくいというのが現状です．グループウェアの技術はこの負荷を最小に押さえ，かつ効果を最大にする努力をしなければなりません．

4.2.1 グループ・プロトコル

プロトコルとは双方が納得した相互作用に関する取り決めです．プロトコルには，ハードウェアやソフトウェアに組み込まれている技術的プロトコルと，社会的プロトコルと呼ばれ，参加者の手にコントロールが残されているものとがあります．

技術的プロトコルの例としては，コンファレンスの運営システムなどで用いられている，部屋割りの方法があげられます．一度に1人のユーザの入力しか受け付けないようにして，参加者に申込の順番を強制するというプロセスです．

一方，社会的なプロトコルは，システムによって強制されるのではなく，メンバー相互の理解と同意のもとに，社会的なエチケットにしたがってグループのプロセスをコントロールしようとするものです．礼儀正しく順番を待つとか，挙手によって部屋割りを決めるという方法です．GROVE の実験的な利用では，共有ウインドウを円滑に利用するための社会的なプロトコルが生まれました．だれでも，いつでも共有ウインドウをスクロールすることができるわけですが，おたがいの了解なしに，各自が勝手にスクロールをしていたのでは，混乱を生じるだけだということに，利用者が気づいたからです．

両方のアプローチには，それぞれに，長所と短所があります．

社会的なプロトコルの余地を多く残しておくことは，参加者の相互協調を促進することになります．参加者が自らの手で自分達のプロトコルを開発しなければならないからです．また，グループ独自のプロトコルに対応できるように，グループウェアには適応性が求められます．しかし，社会的プロトコル (特にその場しのぎのプロトコル) は，往々にして不公平であったり，混乱を招いたり，役に立たなかったりする場合があります．

これに比べて，ソフトウェアに組み込まれている技術的なプロトコルでは，利用者がプロセスを忠実に守ることが保証されていますし，グループの活動をより組織的に運営でき，経験不足のユーザのサポートが可能です．しかしながら，技術的なプロトコルは拘束がきつすぎる傾向があります．グループに固有な作業形態はサポートできませんし，作業ごとに固有のプロセスを強いられることにもなります．

4.2.2 グループ作業

個人でやるよりも，グループで作業したほうが正確でスピーディな場合は，よくあります．バスケットボールや消防士のチームがよい例でしょう．そのような作業のやり方を合成的作業とかグループ作業と呼びます．

グループ作業は同期，非同期の両局面で起こりえます．各種の事務処理の手続きは，非同期型のグループ作業を記述したものといえますので，オフィス内の情報処理システムとして幅広く研究されています．研究課題としては，組織的知識，例外処理，調整機構，非構造的な活動などが上げられます．

組織の構造やその目的，歴史に関する知識は，たとえ，その内容が変わりやすく，表記しにくいものだとしても，事務手続きを履むうえで役に立ちます．例外処理は，事務処理が「開かれたシステム」である以上必ず起こりえます．しかも，日々の活動に関する情報の中に不完全なものや，部分的にしか記述されていないものがある場合には，事務手続き上起こりうるすべての局面を明確に定義することは不可能ですから，この点をいかに解決するかが課題です．

また，事務処理の中には，並列に進行する非同期の仕事があり，一時的にだけ相互にその進行を束縛するという場合があります．したがって，事務の遂行にあたって，担当者が履むべきいくつかの手順と個々の手順の間の掛かり合いを，担当者に知らしめる調整機構が必要となります．非構造的な活動が研究課題となっている理由は，事務処理がまったくのルーチン・ワークにはなりえないからです．企画や問題解決といった，構造化されていない活動は事務処理

の至るところで起こりえますから，これらをどのように取り扱うのかが，課題となるわけです．

同期型のグループ作業の支援はグループウェアの特質の一つといえます．非同期型のグループ作業の時にお話しした問題は，すべて同期型のグループ作業にも当てはまります．この点について，小グループ向けの架空の「投票ツール」を想定して，説明しましょう．ツールの機能は以下のようなものになるでしょう．

ユーザがツールを起動すると，「投票開始」と「投票終了」のボタン，テキスト入力用の空間がついたウインドウが開かれます．ユーザが，投票の主旨を入力し「投票開始」ボタンを押すと，投票参加者すべての画面にグループ･ウインドウが現われます．グループ･ウインドウの中には，「賛成」「反対」「白紙」「棄権」の４個のボタンがあり，その下に投票の結果を示す棒グラフが表示されています．

以下に，同期型のグループ作業支援のための問題として提起されている事項を，この例にそって列挙します．

組織的／社会的要因： 上記の機能を備えたツールを作成することは，比較的容易です．設計上の問題は，ツールをいかにしてさまざまな状況に対応できるようにするかという点にあります．参加者が投票内容を変更できる，投票途中の結果を表示する，参加者の誰でもが票決を開始できる，もし互いの操作を隠すことができるのであれば匿名での投票を許す，等々のことがらを考慮しなければなりません．

ツールが，たとえば仲間内の投票にも，階層構造をもった多少非民主主義的なグループでの投票にも，対応できるようにするにはどうすればよいのかが問題なのです．少なくとも，組織的な知識のみでなく，グループの知識までをも考慮する必要があることは明白です．

例外処理および調整機構： 典型的な例外処理は，非協力的なユーザが自分の役目をサボった場合や，予期せぬアクシデントにより投票の最中にユーザの１人がいなくなってしまうなどして，グループの構成が変わってしまった場合に発生します．グループ作業を行なうには，参加者の各人が果たすべき責務を持たなければなりませんし，また，各人の応答時間には当然ばらつきがありますので，必然的にそれを調整する必要がでてきます．

単純な解決法としては，音声などによる予備の意志疎通手段を用意しておいて，不測の事態に対処するということが考えられますが，システムは，少なくとも，問題の発生を感知/通知でき，利用者が動的に作業環境のパラメータを変更できるようになっていなければならないと思われます．たとえば，システムは投票の進捗をモニターし，いつまでも投票しない人がいれば，その旨を投票の仕掛け人に知らせる．そして，極端な例ですが，仕掛け人は投票者のリストからサボっている人を外すことができるといった具合です．

グループ活動支援の統合化： 同期型と非同期型のグループ作業は，相互に補足する形で，より高次元の仕事の達成を可能にしています．たとえば，システム設計プロジェクトでは，要求分析という非同期型作業を行なう一方で，対面会議という同期型作業も行なわれています．会議全体の進行は，それほど組織だった形式では運営されませんが，会議の最中には，投票やブレーン･ストーミングのような組織化された同期型作業も発生します．この事実は，グループウェアの設計者が考慮すべき点として，組織的/非組織的作業の統合，および同期/非同期型作業の統合，さらには両者の統合支援の必要性があることを示唆しています．

投票ツールを例にとっていえば，他のツールや作業で利用できるように，少なくとも投票結果をセーブする機能くらいは持っていなければならないというような点です．グループ･プロセス支援ツールの設計者は，グループにだけ目を向けるのではなく，グループ全体の目的，さらには組織や社会の中におけるグループの位置づけまでを見通していなければならないともいえます．

4.3 同時性管理

複数の利用者が同時に行なった操作によって矛盾が発生しないようにするために，グループウェア･システムでは同時性の管理が必要になります．たとえば，GROVE のようなグループ･エディタでは，あるユーザが文章を変更している際中に，他のユーザがその文章を抹消してしまうというようなことが起こりえます．この種のユニークな同時性管理の問題が，グループウェアの研究ではいくつか指摘されているのですが，排他制御やトランザクション処理のようなデータベースでの同時性管理で用いられている手法では，役に立たないばかりか，グループ内の密接なチームワークを妨げてしまう恐れさえもあります．

以下に，グループウェアの設計者が直面している，同時性管理に関するいくつかの問題を上げておきます．

応答性： ブレーン･ストーミングや集団での意思決定を行なうと，即時応答型のやり取りからよい結果がもたらされる場合が多々あります．リアルタイム･グループウェアは，この種のグループのリズムを乱すことなく，作業を支援しなければなりません．そのためには，ユーザの操作に対する即応性と，その操作の内容を他の利用者に敏速に通知する機能が求められます．

グループ･インタフェース： WYSIWIS やグループ･ウインドウといった技術を基礎にしたグループ･インタフェースも，同時性管理とかかわっています．グループ･インタ

フェースの実現には，まったく同じか，ほぼ同様な内容を同時に画面に表示する必要があります．もし同時性管理の仕組みが，あるユーザの活動をすぐさま他の利用者の画面に反映できないものであるならば，グループ力学の観点からの考察が必要になります．考察の結果，壊滅的な事態を引き起こすことが予測されるようならば，そのような仕組みは採用されるべきではありません．利用者各人が，それぞれ少しずつバージョンの異なった内容を見ながら共同作業を行なうのは，単に各人の密着性を損なうだけだからです．

広域分散： リアルタイム･グループウェアによってもたらされる第一義の恩恵は，物理的にたいへん離れた者同士でも，同時に，一緒に働ける点です．現在の通信技術では，ローカル･エリア･ネットワークに比べると，広域ネットワークの通信速度は格段に遅いため，広域分散を支援する場合には，実際にどの程度の即応性を確保できるのかを，検討しておく必要があります．さらに，通信回線のトラブル発生率の高さを考慮すると，弾力に富んだ，柔軟な同時性管理アルゴリズムが必要になります．

データの多重保存： リアルタイム･グループウェア･システムの即応性を確保するためには，データの複製を各ユーザ･サイトに保持する必要があります．なぜなら，各サイトで頻繁に，非常に重い処理が行なわれる可能性があるからです．たとえば，ロスアンゼルスのユーザとニューヨークのユーザとが共同で文書の作成をしている場合を考えてみましょう．当然，両者はグループ･ウインドウを使って文章を共有しています．もし，データの複製が両サイトになければ，単なるウインドウのスクロールが行なわれただけでも，スクロールに必要な文章を手にいれるためだけで，両サイトの間に多量な通信が発生し，その応答時間は破滅的に低下することが，容易に想像していただけると思います．

頑丈さ： ここでいう「頑丈さ」とは，部品の故障や不測のユーザ操作などの異常事態からの復旧のことを指しています．どこかのサイトのクラッシュや通信網のダウンからの復旧といった，分散システムではおなじみの問題が，やはり，グループウェアでも重要です．また，ユーザの予期せぬ行動への対処も考慮されなければなりません．たとえば，データベース･システムにおいて，新しい利用者を追加することが問題を引き起こすケースはあまり考えられませんが，グループウェアのセッション中に参加者を追加しようとすることは，システムの構成を再編成することを意味します．同時性管理のアルゴリズムは，新規ユーザのセッションへの不意の参入や，突然の退出といった不測の事態から，容易に回復できるようになっていなければなりません．

同時性管理にはいくつかの方法がありますが，リアルタイム･グループウェアの問題を，一般的なリアルタイム･システムの問題にまで広げてしまうと，話が大げさになってしまうので，ここでは，リアルタイム･グループウェアの設計にとって有効な技術だけに的を絞ってお話しします．

4.3.1 単純なロック機構

至極単純な同時性管理の方法としては，データを更新する前に必ずロックするという方法がありますが，これには，いくつかの問題があります．

まず第１は，ロックを掛けるそのこと自体，つまり，ロックを依頼して実際にロックされるまでに，オーバーヘッドがかかるということです．すでにロックされているデータにアクセスするための待ち時間まで考慮すると，応答時間が極端に低下することは，おわかりいただけると思います．

第２の問題は，ロックの対象範囲の問題です．たとえば，テキスト編集において，ユーザがカーソルをある行の中央に移動し，１文字入力したとしましょう．この時，いったいロックされるべきは，１文字でしょうか，１単語もしくは１文，それとも１節でしょうか？ 利用者にとってみれば，できるだけロックの範囲は狭いほうがストレスを感じないと思われますが，きめの細かいロックはシステムにオーバーヘッドをかけてしまいます．

第３番目は，ロック要求および解除のタイミングの問題です．テキスト編集においてロックはいつ行なわれるべきでしょうか？ カーソルの移動中でしょうか？ それとも，キーをタイプしている最中でしょうか？ システムは，この種の意志決定をユーザに委ねて，ユーザを煩わせてはいけません．しかし，テキスト･エディタのコマンドの中にロック･メカニズムを埋め込むことが非常に難しいのも事実です．

たとえば，カーソルが移動するとロックを解除するという方法を採用した場合を考えてみましょう．あなたが入力中のテキストは，カーソルが固定されているのでロックされ，他の利用者は変更することができない状態にあります．ところが，あなたが，どこかからテキストをコピーしようとして，カーソルを移動しただけで，ロックが解除され，あなたが入力中のテキストを他の利用者が変更あるいは削除可能になってしまいます．しかも，他の利用者が変更を始めた時点で，あなたが入力していたテキストはかれによってロックされてしまうので，あなたはテキストをいじれなくなってしまいます．これでは，システムが，スムーズなグループ活動の流れを阻害してしまうことになりますので，よくありません．

より柔軟なロック機構の研究としては，一定時間のアイドル状態が続くと，ロックを解除する"Tickle locks"方式や，

優先的なコマンドによってロックを強制的に解除する"soft locks "方式などが報告されています．

4.3.2 トランザクション･メカニズム

トランザクション･メカニズムは，非リアルタイム･グループウェア･システムの同時性管理では成功していますが，リアルタイム･グループウェアではいくつかの問題があります．まず，トランザクション処理をもとにした分散型の同時性管理アルゴリズムは，実現が非常に難しく，かつユーザへの即応性能が低すぎます．また，ロック機構を使ってトランザクションを管理する方法は，先程お話ししたような問題を引き起こしますし，タイム･スタンプなどの方法を使うと，システムが勝手にユーザの活動を中断してしまう可能性が出てきます．

トランザクションの長さも問題です．一般的に，長いトランザクションは対話型の利用にはふさわしくないといえます．トランザクションの処理が終了するまで，その結果として引き起こされる事態が，他の利用者には見えないからです．かといって，たとえばキーストロークごとのような短いトランザクションでは，負荷がかかり過ぎます．

これらの問題は，データベースとグループウェアの思想的な差異を，如実に表わしています．これまでのシステムでは，個々の利用者に対して，まるで一人だけでシステムを利用しているかのような錯覚を与えることに，力が注がれてきましたが，グループウェアの場合には，個々の利用者の活動が他の利用者によく見えるようにするための努力が必要になります．個々の利用者のトランザクション処理の途中経過が，他の利用者の目に触れないようにすることは，グループウェアの目的を間接的に妨げることになるからです．

4.3.3 中央管理法

同時性管理のために，中央管理プロセスを導入する方法もあります．データの複製がすべてのユーザのワークステーションに保持されている場合を想定して，お話しします．中央管理プロセスは，個々のユーザの要求を受け取ると，すべてのユーザのワークステーションにその要求をブロードキャストします．すべてのユーザに対して，同じ操作が同じ順番で行なわれることになりますので，すべての複製の中身が同じになります．しかし，この方法では，ボトルネックなどの，中央管理機構にそもそも内在している問題が当然起こりえますし，ユーザから要求がなされた瞬間に操作が行なわれるのではなく，いったん要求は中央管理プロセスに送られ，再編成されてから戻ってきますので，即応性に欠けることになります

4.3.4 従属関係の検出

マルチ･ユーザ･システムの同時性管理の方法には，トランザクションの従属関係を検出するモデルもあります．操作の衝突を検出するために，それぞれの操作にタイム･スタンプを打ち，人手によって衝突を回避しようとする方法です．この方法の最大の利点は，なんら同期をとる必要がないことです．衝突が起こらない操作は，要求されるとすぐに実行されますので，即応性は非常に高くなります．

しかしながら，一般的にいって，グループウェアには，ユーザの協力を求める型のメカニズムが有効なのですが，ユーザの間違いによってデータの整合性が破壊されやすいという弱点があり，この点が問題だといえます．

4.3.5 逆実行

操作を逆実行することによって同時性を管理する方法もあります．ユーザの要求はすぐさま実行されるのですが，必要に応じて，一度実行した操作を元に戻せるように，情報を保持しておくやり方です．この分野では，多くの有望視された同時性管理機構が消えていきました．この方法を取るかぎり，ユーザによって行なわれたすべての操作に対して，グローバルな順序づけをしておく必要があるからです．相互に干渉し合う複数の操作が同時に行なわれ，そのいくつかを元に戻して再実行しなければならなくなった場合，一連の操作に正しい順序付けがなされていなければならないので，どうしてもグローバルな順序付けが必要なのです．また，全参加者の操作の再実行/取消を行なわねばならないので，実際には，再実行と取消だけが繰り返されてしまうという結果になってしまいます．

4.3.6 操作の変換

本日お話しするグループウェア同時性管理の最後の方法は，操作の変換です．この技術はGROVEで使われているもので，先程お話しした「従属関係の検出法」に操作衝突の自動回避機能を付け加えて，即応性の高い操作トランザクションを可能にしています．

この方法では，個々のユーザが，それぞれGROVEエディタの複製を保有します．そして，あるユーザから何らかの操作が要求されると，それは，操作を要求したユーザに対してのみ，局所的に，すぐさま実行されます．その後，操作は，「状態ベクトル」と呼ばれる，他のワークステーションから要求された操作の数を示す情報と一緒に，参加者全員のワークステーションにブロードキャストされます．個々のエディタの複製はそれぞれ独自の状態ベクトルを保持しており，他のワークステーションから送られてきた状態ベクトルと比較されます．両方の状態ベクトルが等しければ，他から要求された操作は，その通りに実行されますが，違いがある場合には，実行の前に，操作要求が変換されます．変換は挿入や消去といった操作の型によって異なります．また，すでに，実行された操作ログの内容によっ

ても変わってきます.

変換の詳細について述べている時間はありませんので,興味のある方は次の論文をお読みください.

Ellis,C.A., Gibbs,S.J., 'Concurrency control in groupware systems' In Proceedings of the ACM SIGMOD'89 Conference on the Management of Data (Seattle WA, May 2-4) ACM, New York, 1989.

5. おわりに

本日予定していたお話は,これでほぼ終わりです.残り時間もありませんので,あまり多くを話すことはできませんが,ごく簡単に,これからのグループウェアの方向性についてお話しします.キーワードはワークフローです.

現在,ワークフローを基礎にしたグループウェア製品が徐々に出回りつつあります.米国でのワークフローの普及度は次のようなものです.

- ·導入済み 11.9%
- ·商品選定中 2.6%
- ·必要性の調査中 48.4%
- ·未考慮 37.1%

このようにワークフローが注目されるようになってきた背景には,1980年代の生産性に関する次のような調査結果が起因しています.すなわち,「1980年代,製造業の生産性は40%上昇したが,事務の生産性は2%減少した」というものです.しかも,事務の合理化や自動化のために1兆ドルものお金が投入されたにもかかわらずです.この事実は何を物語っているのでしょうか? 製造業の生産性が飛躍的に向上した最大の理由として,コンピュータの導入によって,合理化がしやすいように,製造作業のプロセスが明確になっていた点が上げられます.ところが,事務処理では,事務作業のプロセスが明確に定義されていないので,コンピュータの導入によってもその生産性が向上しなかった,というのが専門家の見解なのです.

では,事務作業のプロセスを明確に定義するにはどうすればよいのか? その1つの答えがワークフロー技術の導入ということです.ワークフローとは,何らかの手続きを詳細に記述したのもです.手続き遂行のための活動と情報との関連,手続きとユーザとの関連,また作業の実行順序などが記述されます.さらには,作業の実行や,実行に伴うユーザ活動のモニタリング,調整機構といった機能も含まれます.従って,手続き/活動/仕事/役目/演者/スクリプトなどが,概念的な構成要素となります.

ワークフロー実現のアプローチとしては

- ·情報の構造を重視したデータ·モデリング法
- ·活動と制御フローに着目した活動指向法
- ·相互作用と役割分担を主にしたプロセス指向法
- ·目標,計画とその相互束縛を軸にした目標指向法
- ·人間,社会環境,組織構造を基礎にした人本主義的な方法

などが上げられます.

残念ながら,終了の時間になってしまいましたので,今日は,これらの詳細についてのお話しはできません.最後に,私の話を聞いて,グループウェアやCSCWに興味を持っていただいた方のために,よい入門書を1冊紹介して,講義を終わります.

Ronald M. Baecker, "Readings in Groupware and Computer-Supported Cooperative Work", Morgan Kaufmann Publishers, 1993

インターネット: もう１つの震災時のメディア

中野 秀男
(当時は大阪大学，現在は大阪市立大学)

平成７年１月17日の早朝，午前３時に寝たばかりの私は，激しい揺れで浅い眠りを起こされた．それからは，すでに報道のとおりの惨状である．

この小文ではわれわれが日ごろから研究や教育ばかりでなく，それこそ電話代わりに使っている通信ネットワーク(インターネット)が，この大震災の場面でどのように使われたかを報告する．話は，私の周辺で起こったことと，インターネット上を流れた情報で確認したことを，そのまま書くことにしたい．

私の住まいは，吹田市(昭和45年に開催された万博会場)の近くである．ここでも揺れは激しく，マンションでは，特に上の階に行くほど被害が出た．停電も30秒ほどあり，一瞬,"研究室のマシンのハード･ディスクは大丈夫だろうか?"と心配した．電気はすぐについたので，急いでテレビをつけたが，大きな地震と報道されるだけであった．子供を学校に送り出し，私も１つ用事をこなして，車で研究室に向った．すでにテレビやラジオは，災害の模様を点情報として１つ１つ紹介していた．まだまだ被害の全体の状況が見えないながら，徐々に大変なことになったなと思った．

研究室に到着して最初に見たのは部屋の状況で、次にマシンたちが動いているかどうか、そしてネットワークが外部につながっているかどうか，であった。時間は確認していないが，私が外部宛に初めてe-mailを書いたのが，午前10時７分と記録されているので、研究室到着は，10時前であったろう．すでに関西以外の数人の方から私の安否を尋ねるe-mailがとどいていた．これらのうちの２つはメイリング･リストと呼ばれる形のものあり，同時に複数の登録された人たちに送られる．１つめのリストは，主に関東の人びとのグループのものであり，そこで，私が，私自身の無事と大阪大学吹田キャンパスの状況を報告すれば，少なくともそこに登録された60名以上の方が，その状況を知ることができる．もう１つは，関西を中心にして一部関東の方が入っているリストである．それ以外にも，個人的な安否確認の電子メールも２～３入っている．

昼から大阪市内で予定されている会合の開催確認のためのmailも，京都から入る．研究室でも自宅でも，電話がまったく通じないので，外部への電話連絡はディジタルの携帯電話を使う．幸い，東京にはふつうに通じたので，大阪に来ているはずの担当者から電話があれば携帯電話にかけて貰うように伝言する．しばらくして自宅に戻った時に，携帯電話に連絡が入り，大阪市内の会合はキャンセルとなる．関東方面のメイリング･リストには,"このような場合，家庭や職場にある通常の電話は規制がかかるが，街角の公衆電話は，非常用にもなっているので有用だ" という情報が流れる．なるほど，町を車で通ると，あちこちの公衆電話にそれぞれ10人以上の列ができていた．

５階の研究室がそれほど被害を受けていないのを見て安心し，８階の自分の部屋に行くと，３つある本棚のうち，２つの上部が倒れており，部屋は足の踏み場のない状態であった。翌日，部屋をかたづけたとき，倒れた本棚で私がいつも座っている椅子が曲がっているのを発見．もし地震が昼間だったらと考えて，ぞっとした．17日午前10時24分には，神戸外国語大学の方から "こちらは大丈夫" というmailが出されていた(ただし，中継の神戸大学のラインが復旧したのが翌日の朝であったため，神戸外大からのe-mailは，18日午前10時32分に到着)．

入院している家内の母が気がかりなので，一旦帰宅して，夕方近く研究室に出てくる．その間，家ではNHKテレビをずっと見る．合間合間に電話をかけようとするのだが，やはりかからない．実家の河内松原には，携帯電話でもかからないので少し不安になる．妹夫婦がいるアメリカには，携帯電話で無事を知らせる．入ってくる情報はテレビかラジオだけであり，電話が使えないため，個々に知りたいことがわからない．

午後５時前に研究室に到着．さっそく計算機に向かって情報を集める．メイリング･リストでは，実家のある松原に住んでいる人から，松原は被害が少ないと聞いて，間接的に実家がほぼ無事だろうとやや安心する．午後４時19分，日本のインターネットの管理者のメイリング･リストの１つに，ニュースグループで地震関係の情報を流していると報告があった．そのe-mailでは，あわせて，大阪大学の入口まではインターネットが到達すること，神戸大学はとどかないことが追記してあった．ただ，大阪大学の全学が停電で火事もあるらしいとあり，若干事実と違うので，火事はボヤでありすぐ消えたこと，吹田キャンパスは停電ではない(少なくとも工学部は)と，私から報告した．このリストには日本のインターネットの管理者が多く入っているので，これで神戸大学や大阪大学のネットワーク的な状況が理解していただけたと思う．

インターネットとは，大学や企業などの組織を結ぶネットワークである．最近ではパソコン通信のユーザも，インターネットとの間でe-mail等が使えるようになったので，組織や個人を結ぶ計算機ネットワークだと思ってほしい．日本のインターネットには，この２月１日の時点で1772組

織が接続しており，利用者は70万人ともいわれるが，パソコン通信の利用者200万人も入れれば，300万人に近い利用者がいることになる．世界的にも昨年の秋の時点で46,000の組織が接続しており，利用者は3,000万人とも5,000万人ともいわれる巨大な計算機ネットワークである(なお，あくまでも利用者数は推定)．

インターネットの特徴はいろいろあるが，利用法から考えると，電話のように個人同士の会話も，郵便のように手紙も，新聞やテレビのように多くの人に情報を送ることも，可能である．計算機を介在させるので，電話とは違って記録を残しやすいし，またその情報の再利用も簡単である．また，郵便と違って高速に相手の所にとどく．さらに，新聞やテレビと違って，参加者だれでもが容易に情報を発信することができる．今回の阪神大震災では，いま述べた長所が活かされた場合が多々あったように感じられる．

ニュース･システムは，パソコン通信の電子掲示板とは少し仕組みが違うが，利用者からみればそれほど変わらない．テーマごとにニュース･グループが作られ，その中で複数の会話が進行していく．1年前のロスの地震が，よく今回の阪神大震災と比較されるが，ロスの場合は地震後すぐに新しいニュース･グループが作られ，そこで安否確認等の情報交換がされたと聞いている．今回の地震でも，インターネットやパソコン通信のニュース･グループ(新規に作成されたものも多い)の中で情報交換がなされている．

インターネット上では，既存のニュースグループの1つで議論することが17日朝に提案されて情報交換が始まり，また，新しいニュースグループも作成されて，そちらでもやりとりが続けられた．パソコン通信でも臨時のフォーラムが開設されたと，17日午後10時にインターネット側に情報が流れてきた．17日には，被災を受けた場所についての問い合わせと，それに対する報告がほとんどだったが，18日ごろからは，現地に行ってから計算機の前に戻ってきて報告する人や，身近に被災者がいてその方たちからの電話等からの報告が，情報として流れはじめた．パソコン通信では数千の記事が流れたと聞いているが、インターネットでは，2月4日の夜の時点で約2000の記事が流れている．大震災専用のニュース･グループでも，関連した記事がたくさんあったはずである．

パソコン通信も含めて，インターネットでは，だれでもが情報発信者になることができ，そのため，被災を受けた人や組織の人，また土地感のある人が情報発信をするので，確度の高い情報となった．関西学院大学が崩壊したという情報も新聞で流れたそうだが，神戸方面の大学では，多くの大学の一部の建物が危険な状態になっており，その後で，実際に内部に入って見た人からの情報が流されて，だんだん詳細な事実が見えてきたのである．一番被害が大きいと思われた神戸商船大学が，2月4日にネットワークに接続してきたので，まがりなりにも，大学関係のインターネットは，それぞれの入口までは復旧したのではないかと思う．

インターネット上にはいま，情報を載せる仕組みとして"World Wide Web(WWW)"と呼ばれるソフトウェアがある．専門的には，このソフトウェアはサーバ(奉仕者)といわれており，このサーバの情報を読み出すソフトウェア(クライアント：お客)もいくつかある．WWWは，ハイパーメディアと呼ばれる構造を持っており，マルチメディアであって内部にデータベースを持っていると考えてもらってもよい．今回の震災では，ネットワーク上を流れる情報を貯蔵する仕組みとして，このWWWサーバが活躍した．情報は多くなればなるほど整理が必要であるが，複数作られたWWWサーバは，それぞれが独自の情報を持ちながら，他の有用な情報についても，簡単にそこにアクセスできる仕組みを作っている．

いま、17日以降のニュースを読み直したり，2月1日に流された情報のまとめの記事を見ながらこの原稿を書いているが，後者によれば，日本語のWWWサーバは20近く，英語の情報を持つものは30以上ある(リンクを張っているので数えることは無意味?)．これらのサーバには，17日に国土地理院が撮影した1000枚以上の空中写真の中からの阪神高速道路の倒壊状況や，職員などによって撮影されたと思われる甲南大学や大阪大学の校内の写真が見出される．また，神戸外国語大学では，18日以降の市内の写真を載せている．WWWサーバを最初に立ち上げた奈良先端科学技術大学院では，朝日放送の協力によって，17日午前7時撮影の街のようすの写真もあり，火事等の動画もある．

流されたり議論されたのは，まずは人の安否情報，次に建物や場所を指定しての被災情報，それから，現地で暮らす人びとのための生活情報であった．生活情報には，停電，ガスの供給停止，断水，電話，交通情報が，また募金，救援物資，ボランティア，献血の情報も流されていた．

今回の震災では，電話がかからないことがそれほど被害のなかった地域にもおよび，そのため，パソコン通信利用者も困っただろうが，同じように電話でインターネットを使っていた人びとは，ほとんど最初の数日はお手上げ状態であった．しかし，所属する組織がインターネットに専用線(高速デジタル回線など)で24時間接続している場合は，インターネットの有難味を十分に味わった．

また新聞やテレビ･ラジオからの初期の報道のような面として捉えた，または一部のめだつ所を報道したことに比べると，インターネットでは多くのあるローカルは良く知っている人達からの情報発信が点から沸きあがってくる情報で良く分かったような気がする．

いま，インターネットは，有線だけでなく，無線を使ったトータルなネットワークシステムとして構築されようとしているので，ますます災害時も含めた情報通信基盤となるであろう．

広島原爆 WWW プロジェクト

大場 充
(広島市立大学)

1. SEA 広島ニュースから

4月28日付日本経済新聞(社会面)で報道されたように,5月8日,広島市立大学,SEA広島,中国四国インターネット協議会(CSI)の有志を中心とした原爆 WWW プロジェクトが発足しました.これまで報告して来た通り,広島平和文化センター,CSI,広島市でも「被爆50 年の節目の8月6日を目指して,インターネットを利用したプロジェクトを」と,いろいろな人たちが企画を練り,それぞれの組織内で提案をしていました.SEA広島の企画もそのようなものの1つだったわけです.

4月の初め,広島市立大学情報科学部の2年次生に対するオリエンテーションがありました.各講座の紹介と教員の挨拶,学生の簡単な自己紹介が終わって,成績の悪かった学生への注意をしたあと,時間が少しあまりました.そこで,大場が,「インターネットで原爆の情報を世界に提供するプロジェクトを始めるので,ボランティアを募集してます.参加希望者は私まで連絡して下さい」と宣伝してみました.残念ながら,学生はただ聞いているだけで反応はありませんでした.ところが,次の日の昼食時間に,2人の女子学生が私の部屋をたずねて来て,「先生の昨日の話,私たち興味があります.もう少し詳しく聞きたいんですけど,たとえば何をすればいいのかとか....」といいました.イギリス紅茶をサービスして,「君たちにお願いしたいのは,取材とコンピュータへの入力です.やり方は佐藤先生が指示してくれるから大丈夫」と説明すると,「私たちやります」と心強い返事をしてくれました.「5月の連休明けから始めるので,それまでにもう1人,できれば男の子を見つけておいて下さい」というと,「ハイ」といって帰りました.

4月の中ごろ,同じ広島市立大学の情報処理センター助手をしている前田さんと話をして,CSI,SEA広島,広島市立大学を中心に,ボランティアのプロジェクトを立ち上げようと提案しました.最初前田さんは,広島市のプロジェクトにもかかわっていることから,「市のプロジェクトと競合するのはよくないのでは?」といっていました.大場が,「このようなことは行政機関がやるよりもボランティアがやった方が,ほかの国の人から見れば健全だし,変に政治色がなくなっていいと思う」と説得すると,「それもそうですね」と少し態度が和らぎました.また,「仮に,広島市が WWW の立ち上げに成功しても,同じようなものが複数あるだけの話で,良いほうが生き残ればよいのではないか?」と続けると,「わかりました.CSI の人達には私からつたえます.広島市の方には大場先生から説明しておいてください」という合意に達しました.,sp .3

次の日,広島市の『ゲートウェイ·センター構想プロジェクト』の推進者で広島市立大学の事務職員でもある西村さんにお会いして,「ボランティアのプロジェクトとして始めます」と伝えると,「前田さんから聞いています.ケートウェイ·センターの最初の事業として平和資料館の WWW を立ち上げるということで市役所の各部局と調整をして来ており,大体まとまりそうなので,それと競合するのじゃないかと心配なんですが」と先手を打って来ました.もともとそのアイデアを西村さんに吹き込んだのも大場だったこともあり,また予想した反撃だったこともあって,「原爆の WWW なんていくつあってもいいじゃないですか,提供する情報にオーバーラップがあれば,情報の信頼度も増すし,自然淘汰されて良いものが残ればいいと思いますよ」と切り返しました.そして,「このような政治的に微妙な問題は,前にもいったように学生や市民や生徒がやらないと宣伝ぼくて信用されませんよ」とダメを押しました.「われわれは立ちあげの早さで勝負しますから,そちらは提供する画面の質の良さで勝負してください」というと,「わかりました」と表面的には納得してくれました.

4月15日に庭で車を洗っていると,国際学部のベンソン教授が庭掃除に出て来ました.そこで,プロジェクトの構想を話し参加を求めると,「それはよい話た.ぜひ乗りたい」と好意的な回答,「過去に目を向けるのではなく,人類の将来に対する我々のメッセージを投げかけたいものだ」と抱負を語ってくれた.2人で話していると,同じ国際学部のドボルー教授がのこのこ出て来た.ここぞとばかりにつかまえて,「時間ありますか?」と日本語でたずねると,「あります」と答えたので,大場が説明しベンソン先生が説得にかかる.「それはよいアイデアかもしれない」とドボルー先生もあっさりと陥落しました.大場が,「あなたは日本語が話せるから現場の指揮を頼む」というと,ベンソン先生「それはよい,かれはハーフ·ガイジンだから」と得意なジョークを飛ばす.ドボルー先生も負けてはいません.「日本語はリテラシーの問題だよ」と切り返す.つまり,ベンソン先生は『文盲』だというのです.

4月22日の CSI の運営委員会での最後の議題が,原爆

WWW プロジェクトなどの支援に関する議論でした．半分CSI の事務局みたいな前田さんが，中国新聞からの『折り鶴キャンペーン』支援依頼について説明したのち，「メールしましたように原爆 WWW について大場先生からの提案がありました」と切り出しました．大場が，プロジェクトの概要を説明すると，議長の広島大学の相原助教授が，「これは面白そうなので積極的に応援したいと思います」と援護射撃をしてくれました．すると，安田女子大学の染岡助教授が，「撮影はわれわれ々が協力させていただきます」と参加の意志表明をされた．さらに修道大学の広通助教授が，「思想はないが，うちの学生にも手伝わせます」といわれ，一同爆笑．

SEA 広島ニュース前号を出した直後に，数人の方々からこのプロジェクトに関するご意見をいただきまた．その中に，三島食品の三島さんや広島大学医学部の佐々木先生，ハイエレコンの河野さんなどがおられました．これらの方々には，プロジェクトにご参加いただくようお願いしました．特に，三島社長は日本中を飛び回っておられたため，秘書の方にお願いして宿泊予定のホテルへファックスを送っていただき，宿泊先のホテルから，お忙しいなか，お電話をいただきました．ご迷惑とは重々承知していましたが，電話で三島さんを口説きました．三島さんのような企業の経営者にご参加をお願いした理由は，米国人が見たとき，経営者が入っていた方が内容に関する信頼性が高まるからです．米国では，それだけ経営者の社会的地位が高いからです．また，米国人の日本に対する固定観念の１つに『経営者の品位のなさ』があるように感じていたので，三島さんを利用してイメージアップを図ろうとも思っていました．経営者が，ボランタリーのプロジェクトに参加すること自体，日本では珍しいことですが，日本の経営者も社会貢献事業に積極的な態度をもっていることを示すことも狙いました．

これが，『プロジェクト発足』までの経過です．

2. 幹事会への mail

プロジェクトはその後ほぼ順調に進み，７月中旬にはテスト版のホームページができあがりました (幹事会宛の次の e-mail 参照)．

To: board@sea.or.jp
Subject: A-Bomb WWW Museum
Date: Thu, 13 Jul 1995 14:14:33 +0900
From: Mituru <ohba@edu.ipc.hiroshima-cu.ac.jp>

SEA 幹事のみなさん、大場 @ 広島市立大学です。

SEA 広島と西武さんたちが中心で活動している地域のインターネット組織 CSI が協賛して、ボランティアのプロジェクトが進行中です。プロジェクトでは、原爆の情報を WWW で世界に提供することを目指しています。

現在は、テスト中です。色々な方々から御意見をうかがって、最終版にすべく努力中です。御意見は、主に外国の方々にうかがっています。

先日、アメリカ人の方から、「情報が広島のものだけに偏っている点にすっきりしないものを感じる」との手厳しい御批判をいただきました。たしかに原爆は広島だけに落ちたわけではありません。アメリカの小学生がこれを見たら、原爆は広島だけに落されたと勘違いしないとも限りません。この点は、絶対に修正が必要だと感じています。

そこでお願いです。特に九州地区に関係のある方にお聞きします。いままで、長崎に落された原爆に関する情報を提供している WWW についてお聞きになったことはありませんか？ あったら教えて下さい。長崎の原爆に関する出版物で有名なものや良いものがあったら教えて下さい。英語で出版されているものに限ります。長崎の原爆の被害を物語るイラスト、写真で、コピーしてもよいものがあれば、所有者を教えて下さい。長崎のヒバク者で、そのご経験を語れる方をご存じであれば、その方のお名前、ご連絡先をお教え下さい。

ちなみに、テスト版は以下の URL で公開しています。

http://www.csi.ad.jp/ABOMB/

以上、御協力をお願いいたします。

この SEAMAIL がお手元にとどくころには，最終版の WWW ページが動いているものと思われます (編集部)．

オブジェクト指向を正しく進化させるために

ー YDOC での山崎利治さんのお話を聞いてー

伊藤 昌夫
(MHI エアロスペースシステムズ)

0. はじめに

去る 1 月 27 日, YDOC (横浜支部) の例会で, 山崎利治氏によるスピーチが行われた. オブジェクト指向と形式化手法の関わりについての話で, [1] で紹介されている VDM++ をベースにオブジェクト指向化された VDM-the Vienna Development Method(e.g. [6]) の記述について説明されたのち, オブジェクト指向と形式化手法の関わりについて, ご自身の考えを述べられた. 後半では, お話にもとづき活発な質疑応答が行われた. 出席者は約 30 人であり, 立見が出るほどの盛況であった.

私は, もともと, 記録を取るつもりではなく, 話を聞き終ったあとで, 自分なりに考えをまとめたいと思って, OHP をいただいた. 当然, テープはおろかメモもとっていない. したがって, あのときの山崎さんのお話を正しく再現することはできない. この拙文は, 一聴衆としての私の単なる感想文である.

主張と印をつけてカッコに入れた文は, 私が理解した(もしかしたら誤解した) 山崎さんの隠れた主張である. その後につけたパラグラフは, 山崎さんの話に刺激されて考えた私個人の意見である (つまりほとんどが私的感想である).

また, 当然ではあるが, 私の力量では, この大きなテーマをカバーすることは, 到底できようもない. したがって, 以下の文章は, かなりハスに構え, 眉に唾して疑いつつ, それでもなお, 一緒に考えながら読んでいただけたら幸いである.

さて, 山崎さんは, まず, 先に述べたように, 文献 [1] に示されている図形要素のクラス階層と定義について話された. 雰囲気を理解してもらうために, 図 1 に四辺形の VDM++ での記述を示す. 太字で示されている予約語の内, class や methodlist といった部分は, VDM にはない, オブジェクト指向化のために追加された予約語である.

1. オブジェクト指向は本当に必要なのか ?

主張: 「オブジェクト指向は, ほんとうに必要なのだろうか? オブジェクト指向が提供しているものは, すでに形式化手法が持っているのではないか? あるいは, それを改良することによって充足されるのではないか ?」

山崎さんは, その根拠としていくつかの点を挙げられた. たとえば, (a) 具象化 (reification) や詳細化 (refinement) を支援するための機構を持っていない. (b) 論証 (proof) のための具体的な方法がない. たとえば, OMT のオブジェクト

```
class 四辺形

    instance variables

        辺: seq of ベクトル;
        位置: ベクトル;

    invar-辺1: len 辺 = 4;
    invar-辺2: 辺(1)+辺(2)+辺(3)+辺(4) =0;

    invar-辺2': simple辺;
    invar-辺2": ∀i: inds 辺 ・ |辺(i)| ≠ 0;

    methodlist

        移動(x:ベクトル) ext wr 位置;
        pre: true;
        post++: 位置 = 位置 + x;

end 四辺形;
```

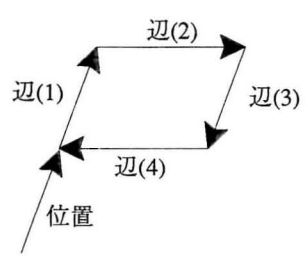


図1. クラス四辺形のVDM++記述例

モデル図を見て,「うん, 合ってそうだ」と納得する位しか方法がない. もちろん, 所詮人間が書くものだからという意味では, 形式化手法でもそうである. しかし, 単なる名前による一貫性のチェック以上のことが可能である. たとえば, 証明による述語の簡略化, 無矛盾性の検査等である.

山崎さんの話の後に, 次の質問があった.

> — 形式化仕様はよいのだろうけれど, そこに行きつくには, 何等かの試行錯誤的な方法が必要なのではないか? それには, OMT などの図による表記は有効なのではないか?

この点に関して, 山崎さんの立場は一見不明である. 何もお答えにならず, 相変わらず超然とされていたからである. 少しこのことについて考えて見る.

図 2 に整理した 3 つの立場を示す. I は従来のモデルベースの形式化手法の立場である (以降, 話を単純にするために, モデル指向の形式化手法のみを対象にする. 山崎さんが挙げられた例がほとんど, Z や VDM の発展形だったこともある). II の立場は, 広く知られているオブジェクト指向分析設計法の立場である. そして III が, オブジェクト指向化されたモデルベースの形式化手法である. I, II, III の並びはほぼ時代順であり, III の立場は当然ベースとなった I と基本的なオブジェクト指向化の動機づけになったいう意味で II の立場から影響を受けている.

私の推測によると, 山崎さんは II のオブジェクト指向分析設計法ばかりか, オブジェクト指向化された形式化手法の III もどうでもよい, 山崎流にいえば"冴えない"と思っておられるのではないか? それがいいすぎだとすると, 少なくとも, いまわれわれ々に必要なのは, I の立場だと考えられているように, 私には思える.

モデルベースの形式化手法は, 集合論にもとづいている. その記述の一例を通して, 形式化手法の持つ"構え"を見て見たい. もし, 形式化手法に困難があるとしたら, それは見慣れない記法を使うとか, 数学を知らなければいけないとか, 支援ツールがないとかということとは, 本質的には無縁である. 摸式的に, 形式化手法 (といっても色々なアプローチがあるので, 先に述べたように, ここではモデル指向の Z や VDM を考える) と, 通常のプログラミング言語でのアプローチを比較して見る.

モデル指向アプローチによる形式化手法では, (抽象機械における) 状態を考える. 手続き指向アプローチでは, 状態及び手続きを記述する. とりあえず, 抽象状態機械におけるというカッコつきの部分は, これからの議論ではあまり関係がない.

図 3 に示したのは, 図書館問題を Z で記述したスキーマの一部である (手に入れやすい? [13] から選んだ). Copy とあるのは, 実際の本の集合であり, Book は, 抽象的な本の集合である (すなわち, この Seamail は特定可能な 1 つの Book の要素であるが, Copy は 1000 部以上存在する). ユーザは, 図書館利用者の集合である. したがって, それぞれの関係 (所蔵 stock, 貸出中 issued) が, (2), (3) において部分関数として定義される.

このように, この 2 つのスキーマを説明しようとすると, それこそ, 注記のためのカッコだらけの長い読みづらい文章ができてしまう. それを何らかの約束ごとにもとづいて端的に書けるところが, 形式化手法の優れている点である. しかし, こう書いてしまったのでは, 本質的な議論にならない. もう少し, 今回の本題に沿ってよく考えて見る. (3) を見てから, それを日本語で書くとどうなるのであろうか.

> — 私が<貸出中>といっているとき, その貸出中ということがらは, 所蔵している本と図書館利用者との関係を表わしている. このとき, 図書館利用者の全員が常に本を借りているわけではない.

上手に日本語にできたかどうかはわからないが, 多分このようになるであろう. 最初の構えに関するポイントが, ここに 1 つある. つまり, われわれは仕様を書くときに,「そんなことあなたもわかるよね!」と思いながら, 気軽に「貸出中に関しては, こうすること」といった仕様を, <貸

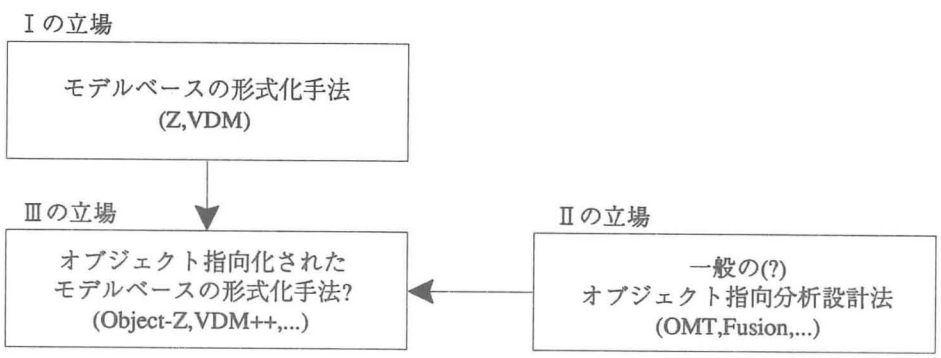


図2. 3つの立場

出中>の定義をすることなく書いてはいないだろうか？もし，定義する人がいたとしても，先の日本語での記述のうちの後半部分は抜け落ちてはいないだろうか？

人は何かを考えるとき，そのことがらを特徴づける積極的な要素はよく思いつくものである．しかし，その何かの直接に関連しないところやその変化については，なかなか考えにくい．少し乱暴な比喩を使えば，たとえば，スケジュールを立てるときに，うまく行く場合のスケジュールはきちんと立てられるが，そのスケジュールが一部でも狂うこと，およびその影響はなかなか考えにくい．それが，たとえばZを使う場合には，何らかの関係を明示する必要があり，その場合，10個弱ある集合間の関係の内から，どれかを選ばざるをえない．そのようにまじめに考えざるをえない分，少なくともこの種の思い込みは避けやすくなるのである．ただこのことも，逆にいえば，必要なことを常に考えなくてはいけないという意味で，大変ではあるが....

次に，少し飛んで，貸出スキーマ中の (16) を見てみよう．これも，先と同様に，日本語に直すことを考える (ちなみに'?'は，入力を表わしており，ダッシュは，ある状態に対して何等かの操作が行われた後の状態を表わしている)．

> "ここで，貸出という操作について定義する．新たな貸出中という状態は，古い貸出中という状態の関係に，今回貸出した本と，図書館利用者の関係を加えたものである"

これは，貸出という操作の定義である．このように，操作の定義は，明示的には行われない．操作前の状態と，操作後の状態を引き起こすものとして定義されるだけである．図4を見てほしい．ある操作 (ふつうその意味的かたまりを機能と呼んでいる) を表現するのに，Zのようにモデル指向の場合，その操作を，事前状態と事後状態によって表現する．操作そのものは明示的に表現されない．

一方，手続き的アプローチ (たとえば，c言語でプログラムを書く場合) では，状態とともに，主として操作そのものを記述する．これが，もう1つの形式化手法のポイントである．もう一度，乱暴な比喩を使おう．徒歩での移動を定義するとき，モデル指向では，最初ある地点にいる人が，移動後はその点と異なる地点にいるとだけ定義する．手続き指向ではどうなるだろうか？ 玄関を出てから，x1歩北に向かい，x2歩南に向かい... という形で，具体的に移動操作そのものを定義する．したがって，たとえば移動手段を車に変更する場合は，その記述を変更しなければならない．

上記は，少し不公平な説明であったかも知れない．なぜなら，ZやVDM-SL(VDMの仕様記述言語) は，仕様記述のための仕組みを用意しているのであって，そのままではプログラムとして走らせることはできない．コンピュータは所詮手続き的にしか動かないのだから．仕様記述言語にも，VDM-SLが持っているような命令的な記述方式も可能にしようとか，何らかの変換方法を規定しようとかいう議論もあるだろう．しかし，ここでは，もう少し違う視点から議論を進めたい．

Lib

stock : Copy $\rightarrow\!\!\!\!+$ Book	·····(2)
issued : Copy $\rightarrow\!\!\!\!+$ User	·····(3)
shelved : F Copy	·····(4)
users : F User	·····(5)
shelved $\cup$ dom issued = dom stock	·····(6)
shelved $\cap$ dom issued = ϕ	·····(7)
ran issued $\subset$ users	·····(8)
$\forall$ u: users · # (issued $\triangleright$ full) $\leqq$ maxloan	·····(9)

Issue

Δ Lib	·····(10)
c? : Copy	·····(11)
u? : User	·····(12)
c? $\in$ shelved	·····(13)
u? $\in$ users	·····(14)
# (issued $\triangleright$ {u?}) < maxloan	·····(15)
issued' = issued $\oplus$ {c? $\rightarrow$ u?}	·····(16)

図3. 図書館スキーマ(上)、貸出スキーマ（下）

先に，モデル指向の仕様記述言語が持っている構えとして，2つのポイントをあげた．1つは，状態の記述に当たって，特徴的な面ばかりを記述するのではなく，全体の中でその記述しようとする特徴的な面以外のことについても同時に示す．そうすることで，状態の界面をはっきりさせるということである．もう1つは，どのように状態が変化するかだけを記述して，操作そのものを規定しないということである．

前者は，人間は特徴的なものについては簡単に考えられるが，否定的な側面については考えにくいという点に逆らっている。後者は，人間は手続き的には考えやすいが，状態を中心としたモデル的には考えづらいという点に逆らっている．このことが，本質的な仕様記述の困難さであるような気がする．つまり，考えにくいことを考えることを強制されるのである．しかし，これらの強制されることがらは，正しくものごとを定義しようとする上で必要であるには違いない．そうだとすると，「考えにくいことを考えることを強制される」というのは，いいかえれば，「考えなくてはならないことを，考えさせるような枠組みを仕様記述言語は持っている」ということになる．

乱暴な比喩を多用しているついでに，もう1つ比喩を使おう．形式的仕様記述は，日本海に面した東尋坊の崖の上ぎりぎりに沿いながら，歩いて行くようなものである．一歩まちがえれば海に落ちるが，100メートルもそこから腰を引くのは好しとしないという立場である．一方，通常の記述は，広大な北海道の原野を歩き回るようなものである．どのようにも歩きまわれるが，正解の道は数本しかない．しかも，その道が正解であるか否かもよく見えない．下手をすると，落とし穴にはまり込んでしまうかも知れない．[13] の中で，山崎さんは次のようにおっしゃっている．

> － 何も数式的とか形式的な言語を使うことがよいのではなく，形式的な言語で考えて自然語ではなすことが大事なのです．何が根っこなのか，つまり何が大事だということが分るからです．仕様記述や設計記述をどう考えればうまく書けるかのヒントが，きちんと形式的開発法という立場で議論されているものの中に見つけることができると思います.(pp.170)
>
> － こういった書き方もやればできるよという力を密かに持っているということが，大事なんじゃないかと思っています.(pp.173)

憶測ではあるが，山崎さんは，すべてはそれがわかってから (図2のIの立場)，次のステップ(II や III の立場) に進めばよい．あなたは，もうその点を卒業したの？ とおっしゃっていたように私には聞こえた．

2. 認知的なクラスの識別が必要なのか？

主張： 「認識論的，経験論的にクラスを識別する方法を各オブジェクト指向の提唱者たちは主張しているが，もっと理性的な方法があるのではないか？」

オブジェクトの説明でよくいわれることに,"オブジェクトはモノという意味ですから，理解しやすいのですよ" というのがある．このことは，必ずしもクラス (注) の識別が容易であることを示しているわけではない．識別の困難さは広く認められていることである．たとえば，Firesmith は [2] の中で，延々と各提唱者たちのクラス識別方法が不十分であると指摘している (pp.169-229)．

> (注) 本稿では，あまり，オブジェクトとクラスを明確に使いわけていない．手法や言語によって，これらの定義がさまざまなこともその理由であるが，コンテキストで判断していただいた方がわりやすいだろうと思ったのでそうしている．ただ，原則的には，オブジェクトはインスタンスであり，クラスはそのテンプレートであるという使いわけをしている．

この問題には，基本的な誤りがある．前提がよくないのである．

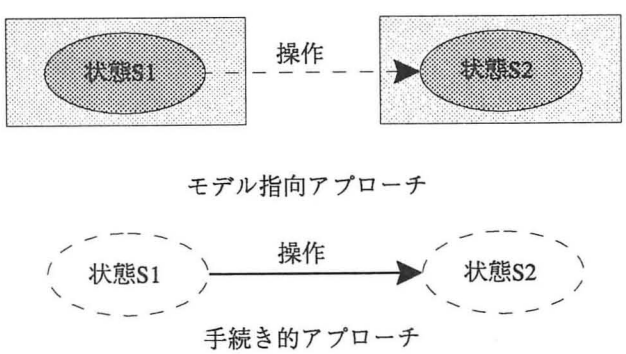


図4. モデル指向アプローチと手続き的アプローチによる機能表現の違い

オブジェクトをモノだというから,"実はここだけの話, お客さん, モノ以外にも概念, 内容物, 概念, デバイス..... もオブジェクトなんですよ"とつけ加えざるをえない. つまり, 限りない数え上げしか手がなくなるのであって, もっと違う軸をそこに見いだせば, (困難さはなくならないにしても) オブジェクトという単一概念でプログラムするという話はすっきりする.

たとえば, 現在の SEA 環境分科会 (SIGENV) の活動の中心となっている CRC (Class-Responsibility-Collaboration) や責任駆動設計 (RDD-Responsibility-Driven Design)[3,7] では, オブジェクトはモノですからなどとはいわない.「責任」を中心に世界を見るからである."いや, そんなことはない" と, 特に [7] を読んだ人は思うかも知れない. たしかに,"名詞や物理的な物や概念を参考にクラスを選びなさい" といっている (pp.37-40). しかし, このオブジェクト識別は, (多分しなくてもすむ) ほんの最初の一歩なのである. その後の本来的な責任駆動の設計方法によって, 必要なクラスを決定して行く

私の考えでは, RDD 手法の特性を明瞭にするならば, 物としてオブジェクトを見つけるといわない方が, 手法そのものは分りやすいのである. おそらく, この本の出版時点 (1990 年では), そこまで思い切りがつかなかったのだろう. 最近の RDD の記事を読むと, 責任中心にクラスを見つけ, 設計することを, 明確に抜き出している. そして, それは, OMT 等のデータ中心の方法とは違うとはっきりいっている. 最近の主張をもとにして, 少し, この RDD の構えを見てみたい.

ボーイング社の技術者が, 酒造システムの設計を, データ駆動設計 (一般のオブジェクト指向分析設計法) および責任駆動設計で実施した. その報告が [14] にある. その結果は次の通りである.

> ー 責任駆動設計の方が, データ駆動設計に比べて, カプセル化という点で優れていた. 責任駆動設計で設計されたクラスは, 大きなクラス凝集度を示し, クラス間の結合度も低い. このことは, メトリックスを用いても確認できた.

ポイントは, データ駆動設計の枠組みでは, データのカプセル化はできても, オブジェクト指向本来のデータとそのデータに関連した操作というカプセル化がしにくいという点にある."そんなはずはない!"と, たちどころに反論されそうである. たしかに, われわれが, オブジェクト指向分析設計法に期待したのは, そんなことではなかった, カプセル化こそがその優位点の１つだったのである.

だとすれば, われわれが考えるべきことは, 次の点になる. クラスとして識別されるものは, 何かの責任を持ったモノでなくてはならない. それは, 単なるモノではないのだ.

責任駆動設計では, 責任が操作に結びつき, モノがクラスの枠組みを作る. 単なるモノとその関係を見つけようとするならば, それは百科事典的な世界を作りあげることに他ならず, そこにあらわれるのは, 単に物理的, 抽象的に存在する物でしかないのである. かつての哲学者がそうしたように, われわれはそのような辞書的世界を作りたいわけではない. オブジェクト指向という枠組みで, 頑丈でかつ変更に強いソフトウェアを作ることを目標にしているだけなのだ. これは, 純粋に技術的な問題である!

もう少し分りやすい例を使おう. 責任駆動設計の推進者の１人である Wirfs-Brock は, 先の [14] についてのコメントともいうべき [15] の中で, 次のような例を用いて, 責任駆動設計の優位性を説明している (pp.52). ファイルキャビネットがあり, その中にはフォールダがある. フォールダの中には, ドキュメントが含まれる. データ中心設計では, ガイドしてもらえるのは, ここまでである. では, 責任駆動設計ではどうか？

ファイルキャビネットは, フォールダが最後にいつ参照されたか, キャビネットに残容量がどれだけ残っているのかを知っている. また, 誰が参照したかの履歴を保持している. これら役割/責任をもつ物として識別される. したがって, クラスは, 必ず何等かの責任, すなわち, 別のクラスに対して解放している操作を含むのである. そんなことはだれでも気づくなどといってはならない. その点に通常気づかないからこそ, ボーイング社の技術者達が経験したのと同じようなことが, 実際にあちこちで起こるのである.

もう１つ, かれらの経験に学ぼう. データ駆動設計で, 作られたシステム全体の制御は, あるスケジューラ･オブジェクトが一括して行う. 一方, 責任駆動設計では, 制御は段階的に委譲される. 設計としてどちらがよいのかといえば, もちろん後者である. オブジェクト指向は, 規模の増大による全体を見ることが困難なことへの対処として生まれてきている. 制御においてもそれが具現化された方がよい (もっときちんと議論しないといけない気もするが, とりあえずこの説明で勘弁してもらおう). では, 責任駆動設計では, その点でどうしてうまく行くのであろうか？

この答は, 比較的明確である. われわれがシステム化しようとするものの多くは, モノであるから, 自分から積極的に動かない. キャビネットは, 空き容量を知りたいがために, 順番に自分の引き出しを開けて, 残量を確認しているわけではない. ドアを開けて確認するのは, それを使う人間である. つまり, システム化しようとしている現実世界を動かしているのは人間であって, その対象ではない.

それをそのまま，プログラムの構造にマップするならば，当然，何らかの人間に相当する集中管理オブジェクトがいて，モノすべてを動かすのである．

そのため，先に述べたように，現実世界をマップしようとするデータ駆動設計の方式は，システム全体が集中管理方式として設計されるのである．一方，責任駆動設計においては，オブジェクトは，ある役割を積極的に果たす擬人的なモノとして設計される．視点は，システムのはるか上方にあって下界を見下ろしているのではなくて，いわば，クラスそのうちにいるのである (つまり，フォールダとしての私の役割は？ と考えるわけである)．

これは，客観的/主観的ということではなくて，単に視点を移動するということである (これは，SS94 イブニングパネルでの岸田さんの私への質問「オブジェクト指向は客観的な物の見方のはずなのに，内からの見方はそれに反するでしょう？」への，遅れて来た答になっていませんか？)

さらに，こういった単なるモノ以外の別の視点 (軸) に着目している手法は，RDD だけではない．ジャクソンシステム開発法 (JSD) もそうである．次章では，JSD について考えて行くが，この中で，もう一度オブジェクト(クラス) の識別について議論する．

3. JSD とオブジェクト指向

3.1 行動主導

このテーマでは，お話はなかった．しかし，ご存じのように，山崎さんは JSD に詳しい方であり，早い段階から JSD の紹介をされている ([4] の共訳者の一人)．本項での議論に関係すると思うので，少し JSD についても考えてみたい．

JSD でクラスと関連するのは，実体 (型) である [5]．実体は次のように定義されている ([4] pp.90)．

(a) 単にシステム内だけではなく，実世界に存在し，かつ
(b) 実世界での生起の時間順に行動を実行したり行動に影響したりする，かつ
(c) 1 個の存在としてみなすことができ，1 つの型の実体が複数あるときは，それらは一位に名前づけできなければならない．

(a),(c) の基準は，一般のオブジェクト指向のクラスの定義の一部といっても違和感はない．特徴的なのは，行動を含んだ (b) の基準である．つまり，実体は行動と組み合わせることによって識別される．もう 1 つ，別の定義を見てみよう．

> — JSD の実体は静的なものではなく，認識可能な時間軸上の行動形式を伴っている．「その実体はこの行動を実行したか」とか「その実体は何回その行動に影響したか」と質問し，当然その履歴に関心を持つ．

Jackson のシステムを見る目は，静的な実体に先立ち，動的な行動にある．行動は，実体に先だって定義されている (行動は，特定時点に，システム外部で生起するもので，それ以上には分解できないものである)．したがって，行動を伴わない実体はあり得ない．オブジェクト指向と比較するならば，JSD は行動指向とでも呼べる．実体は，単に行動の主体という消極的な位置づけである．[8] の JSD の紹介記事において，山崎氏は次のように述べている．

> — 実世界で観測できる行動様式によって実世界を規定しようというのが JSD の基本構想である.」(pp.103)

ここで，先と同じ視点が，同じように含まれることに気づかれないだろうか？ 行動は，実体の行動であるヨットという実体は，進水するという行動を持つ．人間がヨットを建造後初めて海に浮かべることを，ヨットの行動として考えるのである．このとき，われわれはヨットの内からシステムを眺める．

3.2 属性

JSD が ERA(Entity-Relationship-Attribute) 的手法であるというのは，JSD がオブジェクト指向であるというのと同様に，JSD に対するまったくの誤解である．それは前節の考察からも明らかであろう．JSD では，オブジェクト指向的には，モノを実体として重視しないのと同様に，いわゆる属性の扱いも異なる．

実体は，基本的に属性を持たない．

JSD は，行動主導である以上，行動が属性を持つ．たとえば，ヨットシステムの問題において，必要なヨット名は，実体ヨットの属性ではなくて，たとえば，「乗る」という行動の属性である ([8] pp.104)．行動主導とすることで，プロセス間の協調動作を仕様化段階で検討することが可能になる．たとえば，初期モデル化段階におけるプロセスの併合を考えるということがそれに当たる．

3.3 JSD vs オブジェクト指向

少し，乱暴な分け方ではあるが (論文だとしたら少しためらうという意味で)，表 1 を見て欲しい．

この表は，今回の議論に関連ありそうな典型的な手法を，機能，データ，状態の 3 つの視点から，基本的にどの順番で分析･設計がなされるかを示したものである．構造化分析設計 (SA/SD) は，よく知られているように機能分割を中心とする手法であり，機能，データ，状態の順で解析を行う．データ，状態の順番は逆でも構わない．リアルタイム系は逆にした方がよいであろう．

一般のオブジェクト指向分析設計 (OOA/D) は，データ型としてのクラスを先に見つけるということで，データ，状

態，機能と考えることができる．オブジェクト指向手法といっても多くの手法があり，考える順番が不明瞭であったり，単にこれらはスパイラルに実施されるとだけ規定している手法もあるが，少なくとも典型的にはこの順番と考えることができる．特に，リアルタイム系の Shlaer-Mellor の手法は，明瞭にこの順番を規定している．

JSD は，何度も繰り返すが行動主導である．もちろん，行動は状態とは異なるが，先に見たように時間軸上で行動は考慮されるという意味で，カッコ内は状態とした．次にデータが規定され，最後に機能が規定される．この機能というのは他と少し位置づけが異なる．この機能は対象システムが持つべき機能である．JSD ではその設計法のかなり後の部分で，この機能について検討する．では，それまでは何をしているのであろうか？

JSD の D は設計であるが，その中には行動を軸とした現実世界の分析が含まれる．名前は設計法であるが，そのプロセスの多くは，一般の分析行為である．この点については，すぐ後で，分析プロセスの覆っている範囲ということでオブジェクト指向分析の場合を考える．

モデル指向の形式化手法では，そのプロセスは明瞭ではないが，状態中心で考えるということで，状態を最初にしてよいだろう．状態及びその変化は定義されたデータ型を用いて説明される．最後に必要な機能が追加されていく．

しかし，ここでは思考プロセスには深入りせずに，簡単にシステムを捉えていく中心にあるものについて簡単に触れ，次に本項での本題である分析プロセスの覆っている範囲について考える．

システムを考える上での中心がそれぞれ違っているというのは，当たり前のことである．つまり，作ろうとするシステムが動作するドメインによって，相応しい手法は異なるというだけのことである．万能薬的手法は存在しない．ここでは，単純にこの表で比較する限り，モデル指向の形式化手法は，オブジェクト指向よりも JSD に近いことが分るだろう．その点だけを指摘しておいて次に移る．この表を，そうだと納得してもらうにはより深い議論が必要であろう．さて主題である．

「いわゆるオブジェクト指向分析は，本当に分析手法として充分な範囲を覆っているのだろうか？」

オブジェクト指向の喧伝者の多くは，分析からプログラムまで継ぎ目がないと主張している．これは，逆にいえば，継ぎ目がないことにより，その分析といっている作業が，プログラムに寄り過ぎてはいないだろうか？　そのことが，クラスを識別することが困難であるということに，本質的に繋がっているのではないか？

ある程度の批判を承知でいおう．分析でたとえば「本」というクラスを現実世界のものとして識別したら，その「本」クラス及びそれに伴う属性，多くの場合はメソッドも，すべてコードに反映される．したがって，継ぎ目がないといい得るのである．しかし，早い時点で何故こんなことを決める必要があるのだろうか．間違って識別するその危険性が，クラス決定を慎重にせざるをえなくしているのではないか？　もし，このような状況において，クラス決定は大したことはないんだという人間がいたら，それは単に楽観的なだけではないのだろうか？

"いや，そうではない" という反論が即座に返って来ることも承知している．これらは，詳細化の過程で，整理統合，精緻化されるものである．それならば，JSD における仕様化段階をよく味わってほしいと思う．仕様化段階初期において，たとえば先のヨットシステムの例でいうならば，最初のシステム仕様図において，「貸出し」は現れない．解くべき問題は，平均乗艇時間を求める問題であるにも関わらずである．それは，Jackson の明瞭な"実世界"と"モデル" の区分にもとづいている．あるいは"モデル"は機能よりも安定しているからである．広く流布しているオブジェクト指向分析において，単にモデル化だけをしようとしていて，本当に実世界を記述しようとは思えない．

別に，JSD がオブジェクト指向分析より優れているなどというつもりはまったくない．先に述べたように，それぞれの手法には適用するのが望ましいアプリケーションないしは問題領域があるからである．しかし，JSD の構えを知ることは重要である．ここで，2 章の「認知的なクラスの識別が必要なのか？」に戻ろう．属性やメソッドをカプセ

表1. 手法と代表的な考え方の手順

手法	代表的な考え方の手順
SA/SD	機能 ➡ データ ➡ 状態
OOA/OOD	データ(型) ➡ 状態 ➡ 機能
JSD	行動(状態) ➡ データ(型) ➡ 機能
モデル指向の形式化手法	状態 ➡ データ(型) ➡ 機能

ル化したクラスを早期に識別することは，困難である．それは，実現に寄りすぎているからである．つまり，いくら認知だからわかりやすいといったところで，われわれはモノを或いは現象を何等かの恣意に基づいて見るからである．

たとえば，お酒は，私にとっては飲み酔うべきものとして見えている．一方，調理中の人にとっては (キッチンドリンカでない限り)，同じものは料理に深みをつける材料の一部として見えている．オブジェクト指向分析におけるクラス識別が，先に見たようにプログラミングに密接である以上，この問題に対する解決策は単純である．一つは，より設計サイドに近づけた議論を行う (解決策 1)．もう一つは，逆に，より分析がカバーする範囲をより本質的な部分へと拡張することである (解決策 2).

解決策 (1): プログラミングにおける現実的要請を反映する．

解決策 (2): 実世界の分析とモデル化を明瞭に区分する (分析プロセスとして，あるいは図式として)

(1) は更に細分できるだろう．

(a) クラス間の静的，オブジェクト間の動的依存性を最小化する

(b) 既存クラスを勘案する．ただし，こう書くと,"それは分析ではない!" という矢がとんできそうであるから，抽象的な"パターン"を考慮するということでもよい．しかし，私には，たとえば，よいオブジェクト指向プログラマがここにいたとして，かれがシステムを分析するときに,"既存のクラスを考慮することはない"といったら,"それは嘘だ!" といい返すことができる．よいオブジェクト指向プログラマは，既存クラスについてよく把握しているものである．だからこそ，再利用が可能になるのである．そういう人間が，既存クラスを念頭に置くことなく分析図が書けるとしたら，私には驚異である．もっとも，よいオブジェクト指向プログラマは分析などしないといわれれば，それはそれで終わってしまうのだけれども．

上記の (1.b) の解決策の 1 つとして先に述べた RDD 的アプローチの採用をあげることができる．それから動的な関連重視という意味では，これから出て来る改訂版 OMT をあげることができる (e.g. オブジェクトインタラクション図の記述).

(2) に相当するものは，なかなか見当たらない．あえていえば，最近，改訂版 OMT (e.g.[10])，改訂版 Booch [11]，と軒並み採用している Jacobson [9] の use case に期待が持てる．

そういえば，これらの改訂版は，Fusion [12] を含め，第 2 世代オブジェクト指向手法と称して，その分析･設計要素が，図式の細かな違いを除き共通化してきた．それでは,"第 1 世代はいったい何だったのだろうか？" と思う．少しまじめに考えて見れば，これら第 1 世代の，手法的な未成熟さに，そしてバランスの悪さに，他の人たちはなぜ気がつかなかったのだろうか？ いくつかの手法を並列してツール化している人間から見てさえ,"とても明らかなことなのに!"。といういい方はかなり傲慢か．少なくとも，後からいうのは卑怯ではある．

use case は，システムと 1 つ以上のアクタ (人間および自律的に動作するもの) のあいだの相互のやり取りの流れを記述したものである．ユーザの視点でシステムを見ているという点で，(2) はもちろん充足していない．しかし，先にいったように期待は持てるのである．[10] における分析プロセス間の相互関係を見てみよう (図 5).

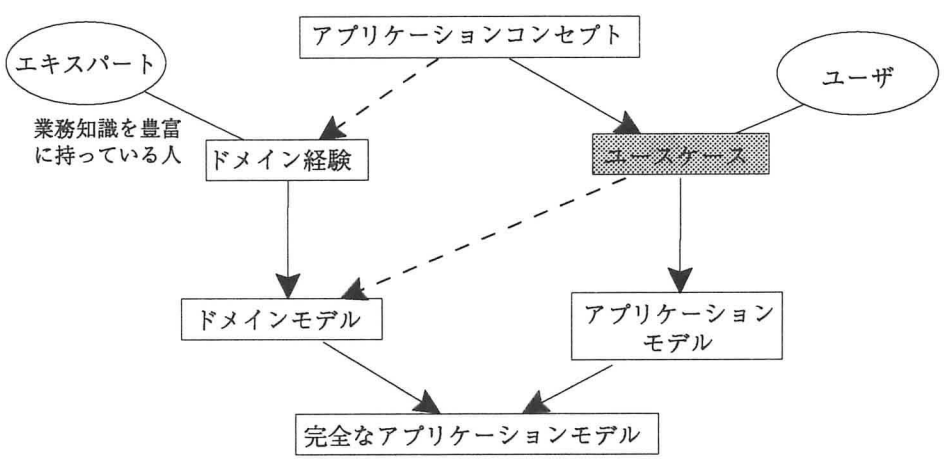


図5. 分析プロセス間の関係 ([10]の図2. を少し変更した)

途中で分岐している部分の左側は，システムが使用される純粋な現実世界である．右側は，作るべきシステムのアプリケーション領域である．原則としてこれらは，同一には検討，記述できない．少しややこしいが，同一に記述するということは，現実世界とシステムの関係を記述することであり，その関係は，システムの実現により変化するからである．すなわち，システムのない現実世界とシステムのある現実世界といった時間的に異なる世界は，同一平面に記述できないからである．もちろん，時間的な前後で2枚書くという方法がないではないが，汎用性を考えるならば，ここに示されているように分けて考え，記述した方がよい．したがって，アプリケーションモデルには，作ろうとするシステムの記述にのみ専念する．

このことが，以前から私が強調している「クラスの識別において，認知的に考える必要はない」ということの意味である．認知的な役割の多くは，ドメインモデル側に反映されているはずである．また，この新しいプロセスは，ドメイン･モデルの存在必要性を明示している．これらのことは，use case の議論をすることによって，明らかになったのであり，現実世界とシステムのモデル化を明瞭に区別する，という分析法の正しい姿に繋がるという意味で，use case に期待できると先に書いたのである．

話は変わるが．よいオブジェクト指向プログラマは，おそらく，分析も設計もしていない気がする．少なくとも，OMT や Shlaer-Mellor といった第1世代の手法を使っては．それは，オブジェクト指向プログラム言語の性質のよさと，それに対する適用するプログラム規模が今まで大きくなかったために，可能であったのだと思う．しかし，用意した容器に容量以上の内容物を入れようとするのが世の常であるから，そうして見て初めて，不十分さに気づくということはあり得る．しかし，そうして気づかなくとも，目をちょっと他に向ければ (たとえば JSD に)，その不十分さ，そしてその欠如に対処する方法は見つかるような気がする．

4. まとめ

なにしろ，本稿は，論文ではなくて感想文であるから (いいわけ?)，全体構想などなしに，思いつくまま書いている．そのくせ，各項目すべてに渡って，強いいい足りなさを感じる．いつまででもこの原稿を書いていけそうだ．それは，きっと形式化手法やオブジェクト指向の持つ豊饒さのためだろうと思う．そのことは，この両者がこれからも充分に発展し得る余地があることを期待させる．

ごく簡単に，この感想文で書いてきたことを振り返りたい．

- 形式化手法の難しさは，正しく考える枠組みを強いるところにある．しかし，それは職業人としてのプログラマがしたがうべき枠組みである．
- クラスの識別は，純粋技術的な問題解決という点から議論されるべきである．認知的視点ではない！　そのとき有効なのが，RDD で使われるような「責任」の視点であり，JSD で使われる「行動」の視点である．このときの視点は，遥か上方から見下す視点ではなく，目はクラスの中にある．
- オブジェクト指向分析の最大の問題は，分析プロセスの貧困さにある．したがって，より本来的な分析技術を取り込む必要がある．そのためには，(問題) ドメインの解析，use case の利用は，その解決策のヒントないしは候補になり得る．

もう1つだけ，オブジェクト指向と形式化手法の関連についていっておきたい．オブジェクト指向は，抽象データ型をもとにしているという説明がある．一方，形式化手法にも，抽象データ型をもとにしているもの(たとえば，今回触れなかった代数的仕様記述) がある．少なくとも，何等かの形でデータ型を用いる．したがって，オブジェクト指向と形式化手法は親和性がよいというのは，今回見てきたそれらの「構え」からすると，あまりにも短絡的な説明である．より深くこれらの持つ「構え」の違いを見つめるべきである．その上で，両者のよい点をとり込む形で，新たなものを生み出す努力を，われわれはしていかなくてはいけないと思う．

5. 最後に

この部分は，岸田さん，熊谷さんと一緒に，名古屋伏見の「鷹」へ行った翌日に書いている．このお店には，全国の名酒が多種並んでおり，京風のおつまみを出す店である．ユニシスの支店の真向かいにあり，山崎さんが名古屋時代によく行かれたというお店である．人がよく行く飲み屋は，その人なりを少なからず反映している．

また，山崎さんほど，こうして原稿を書きながら，氏を思いだすときに，必ず日本酒が一緒にあらわれる人も少ない．そして，山崎さん程，詩的な生き方をしている人を私は知らない．

下手ではあるが，以下の詩を敬慕する山崎さんに捧げて，独断と偏見に満ち，例によって controversial な私の感想文の終わりとしたい．

ディオニュソス讃歌

山崎さんの話は，お酒がそうであるように，
いつも私に多くの刺激を与えてくれる．

乾いた形式化手法の話に，
山崎さんは，豊かな意味を与えてくれる．

その豊かな意味は，大量のノイズを伴っている．

それこそが，

われわれがものごとを理解しようとするときに
必要なものだ.

そして，意味がおぼろげながら見えてきたとき，
自分の不勉強さを思い知らされる.

また
"こんなのはどうでもよいんですが"
といえるところに徹底し，徹底し，徹底して，
思考する者への，嫉妬を感じる.

次のお話を伺える日を楽しみに．そして，私も少しは勉強しなくては!

参考文献

[1] Lano, K. and Haughton. H. (eds.), "Object-Oriented Specification Case Studies", Prentice-Hall, 1994.

[2] Firesmith, D., "Object-Oriented Requirements Analysis and Logical Design", John Wiley & Sons, Inc. 1992.

[3] Wirfs-Brock, R. and Wilkerson, B., "Object-Oriented Design: A Responsibility-Driven Approach", in OOPSLA'89 proceedings, pp. 71-75, 1989. (この訳を，Seamail vol.7 no. 4-5-6-7 で読むことができる)

[4] Jackson, M. 著, 大野，山崎監訳, "システム開発 JSD 法", 共立出版株式会社, 1989.

[5] Birchenough, A. and Cameron J. "JSD and Object-Oriented Design", in JSP & JSD: The Jackson Approach to Software Development, pp. 293, IEEE, 1989.

[6] Jones, C., "Systematic Software Development using VDM" Prentice Hall International , 2nd Edn. 1990

[7] Wirfs-Brock, R. et al., "Designing Object-Oriented Software", Prentice Hall, 1990

[8] 山崎「一プログラマの足跡 - 山崎利治著作撰集-」, 山崎利治著作撰集編集委員会, 1993. (この著作集は，山崎氏の還暦お祝いのときに参加者に配られたものであり，残念ながら，書店等で入手することはできない.)

[9] Jacobson, I., "Object-Oriented Software Engineering A Use Case Driven Approach", Addison-Wesley, 1992.

[10] Rumbaugh, J., "Using use cases to capture requirements", Journal of Object-Oriented programming, Vol.7 No.5, 1994. (ちなみに OMT の新しい本がもうじき出版されるはず.)

[11] Booch, G., "Object-Oriented Analysis and Design with application (2nd. ed.)", Benjamin / Cummings, 1994.

[12] Coleman, D., et al., "Object-Oriented Development -the Fusion Method-", Prentice Hall, 1994. (日本語訳も出版されている．日本語版は手元にないので，出版社等書けないのですが).

[13] 山崎, "形式的ソフトウェア開発", Seamail vol. 7, no. 4-5-6-7, pp. 170~176, 1993 (92 年の若手の会で山崎さんが講演して下さったものをテープ起こしした原稿である．とても分りやすいお話しであった.)

[14] Sharble, R., Cohen, S., "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", ACM SIGSOFT, vol.18 no. 2, pp. 60-73, 1993.

[15] Wirfs-Brock, R., "How Designs Differ", Report on Object Analysis Design, vol. no. 4, pp. 51-53, 1994.

哲学的暴走･脱線顛末記 Part-II

栗原 貴・伊藤 昌夫・青木 淳
(SEA 哲学分科会: まだそんなのできていませんが!)

はじめに

去年の秋の SEAMAIL の記事 (Vol.9, No.5 の "哲学的暴走･脱線顛末記" および Vol.9, No.6-7 の "SS94 特集: ソフトウェア･レミンジニアリング･パネルの記録") を読んで，仕事でオーストラリアに滞在しておられる栗原貴さんから，編集部宛に手紙がとどきました．そこで，もとの記事の関係者お２人 (伊藤昌夫さん & 青木淳さん) に，回答文 (?) をしたためていただきました．栗原さんの御指名は４人でしたが，熊谷さんは例によってモノグサのため，岸田さん@編集部は雑用に追いまくられて，まだ原稿をいただいていません．

nakajima @ 事務局

禅とオブジェクト指向

栗原 貴
(在 AUSTRALIA)

ソフトウェア技術者協会のみなさま

拝啓．御無沙汰しております．

こちらにいると日本語の新しい情報はみな貴重品ですが，なかでも SEAMAIL は一番楽しみにしています．私にとって SEAMAIL はソフトウェア･エンジニアリングの最新の情報ソースでもありますが，それ以上に内容がユニークでワクワクします．特に「ソフトウェア･エンジニアリングを仏教の言葉で語る」というのがおもしろいと思います．

他人を語ることによる自己紹介

本来言葉で表現できない悟りを表現するために，禅では，逆説や否定，そして「たとえ」など，多彩な表現が使われます．私もそれにあやかった自己紹介をしてみます．方法としては，SEAMAIL 誌上で活躍されている方々を何人か取り上げさせていただきました．人選･内容に他意はありません．あくまで，みなさんに「私の感じ方」を明らかにするために取り上げてみただけです．

青木淳さんは，すっかり佛教文化に (思想だけでなく) どっぷり漬かった立場で物事を見ておられる様子．でも，こういう態度が私は大好きです．マネジメントの重要さはもちろんですが，それだけが独り歩きしてしまっては，宙に浮いてしまいます．あくまでプログラムを作る者の視点を忘れてはいけないと思います．青木さんのプログラマ道？ (そういう言葉があれば) をきわめようとする姿勢の清らかさには，思わず合掌してしまいます．ちなみに，青木さんの書かれた「オブジェクト指向システム分析入門」はすでに拝読しました，他にも著書があるなら，ぜひ読ませていただきたいと思います．

伊藤昌夫さんの，西洋哲学の知識を縦横に用いて，さらには道元まで引用して哲学的方法論をに探求されるパワーには，ただ圧倒されるばかりです．理想の道具をあらわすのに「見えない」という表現を使われていますよね．私もソフトウェアの特徴はそのメタ性にあると感じていました．それをつきつめて，具体的にイメージを持っておられる所がすごい! 生半可な定力 (精神の集中力) ではないと感じ入ります．

熊谷章さんのご苦労は，私にも理解できます．東洋思想には，実体験をビッタリ表現する言葉がたくさんあります．これらの言葉は，使うべきところに使えば，これ以上はない効果がありますが，ふつうは討論に用いません．それを討論の場に持ち出すことは，大きな冒険だと思います．あえてそうされる熊谷さんの真の意図は解りませんが，結果として SEAMAIL の内容に幅が出てくることはたしかでしょう．より一層のご活躍を期待します．

岸田孝一さんの語り口の柔軟さ，そして知識の深さは，すなおにすごいと思います．私の批評できる範囲を超越しています．あらゆる思想や機縁 (その場その場の状況) をも自由自在に操るさまは，佛教でいうところの仏の姿そのものかも知れません．

かく申す私は，しっかり曹洞宗です．ただし在家 (僧侶ではなく一般の信徒という意味，TOOL 供給者じゃなくて単なるユーザー) ですので，伊藤さんとは若干立場が違います．

公案という再利用

禅宗では，よくできた問答は．公案 [1] という形で再利用されます．SS94 のイブニング･パネルの最後の方で岸田孝一さんがパネラに鋭いところを突いた質問 [2] を投げて

おられます.「オブジェクト指向を主張する人が，一方で客観的な見方を否定する発言をするのは矛盾している」というこの問いかけには，ソフトウェア再利用の本質に関わる矛盾が含まれていると思えるのです.

[1] 禅宗では，古えの祖師たちが行なった禅の本質に関わる問答を，公案と称して大切にします．そして，それをいろいろと用いて，教育やら禅のレベルの認可などに使います．いわゆる禅問答です.

[2] SEAMAIL Vol.9 No.6-7 Page16.「SS94 のイブニング・パネル」より.

岸田さんの出された公案

「オブジェクト指向は客観にもとづくからこそ再利用を可能にするはずなのに，(オブジェクト指向をすすめる者が一方で) なぜ主観を強調するのか？」

私の最初の回答は次の通りです.

「世の中に，作成者の主観に寄らずに構築されたオブジェクト・ライブラリあるいはフレームワークがあるのでしょうか？」

解説してみます.

オブジェクト指向のライブラリあるいはフレームワークは，機能が統合されているからこそ，再利用率を上げることができます．その統合方法の決定は，作成者の主観によります．ですから，オブジェクト指向では，その背後にある主観まで含めて，すべてが伝われば，作成者と同じ高いレベルで再利用ができる条件が整います (同じレベルで再利用ができるとはいいません).

しかし，これで議論は終りません.「主観が先か？ 客観が先か？」というこの議論は果てしなく続いてしまいます.

客観主義： 作成者の「統合方法」自体を，客観情報化して，ライブラリに保存する必要があります．客観情報化ができてこそ，再利用ができます.....

主観主義： ライブラリに保存された作成者の「統合方法」を読み解く作業は，作成者の主観を理解する作業です .....

この議論は，いいかえれば,「主観」とは何かを追及する試みだともいえます.

主観客観論議を終らせる試み

さて，禅では，仏法を探求することは，自己を探求することです．そして "自分自身を題材に取って" 自己を探求します．これは，主体が主体自身を探求する作業です．頭を使ってこれを追及しても，すぐに行詰まります．そこを超えるためには修行が必要ですが，うまく行かない理由は，禅でも模型的に説明されます.「いま，まさに活きている自分自身という一つのものが，同時に追及する側とされる側の両方になれない!」と.

ここでおもしろいのは，この説明を裏返しに見ることができる点です．主体が決して対象として扱えない情報とは何か？ つまり，主体に取って決して客観的にならない情報とは何か？．それは主体自身です．さらに，主体をその働きによって定義づけるなら,「主体の働き」に関連した情報だとも，いいかえられます.

そこで，あまたの先人たちがいわれるところを使って,「主観とは主体の働きに関連した情報である」と公理を置いてみましょう．すると，興味深い定理が導き出せます.

「主観を直接に客観に展開することはできない」

別の主体が同じ主観を持つことを妨げる理由はありません．前提と公理から，同じ状況で同じに働く主体があるとすれば，その限りでは同じ主観を持つということができます.

「同じ働きをする主体は同じ主観を持つ」

ゆえに，送り側と受け側の双方の主体が協力して,「たとえ」その他の方法を使い，受け側に主観を再現することが可能なはずです.

「主観は伝達可能」

岸田公案に対する第 2 の回答

人間がプログラムを作り再利用する主体である以上，そのプログラムに関する主観を含めて再利用する方が自然です．しかし，逆に主観を伝達不可能と置いてしまうから矛盾が出る，と解きます.

終りに

SEAMAIL 編集部が，私たち会員の受信だけでなく，発信を待っているということですので，思うところを書いてみました．一応文章の形にまとめてはみましたが，内容には，自分で読み返してみても，いろいろに不満を感じます．たとえば，一般に使われている主観という概念が，上の公理で置いたものと同じであるという検討が抜けていますよね．ですが，まあ今回はこれで勘弁してください.

オーストラリアでは 2 週間以上の夏休みが普通です．その休みを利用してここまでやってはみたものの，SEAMAIL で活躍されているみなさんの文章力は凄いですね.

ともかく，今年もよろしくお願いします.

敬 具
1995 年 1 月 3 日

鏡の国のアリサ

伊藤 昌夫
(MHI エアロスペースシステムズ)

第1章　鏡の中の自分

そのことだけは，たしかでした．鏡を見ているときに，鏡を見ていないことは．だって，私は，鏡の中の別の私を見ているだけで，ひんやりとしたガラスを見ているわけではないんですもの．おかあさんは，いつもおかしなことをいうわ．いつまでも鏡を見ているんじゃないよ！　って．私は，私を見ていて鏡を見ているんじゃないわ．それに鏡を見ないかぎり，私は私ってわからないんですもの」

アリサの家は，古いわらぶき屋根の家．じっとしていると，囲炉裏の中で枯れ木がはぜる音と，雪の降る音が聞こえてくるだけです．そうそう，ときどき，屋根から雪のかたまりが落ちる，どさっという音が一緒に聞こえてきます．新雪の香りにまぎれてただよう生木のこげたチクチクする匂いで，ふっと，われにかえります．

「つまんない！　私の相手をしてくれるのはテディだけだわどうして，正月なのにみんなでかけてしまうのかしら？　あのやんちゃなおじいちゃんにあうくらいなら，テディと遊んでいるほうがいいわ」

テディは，去年の８月に生まれたばかりの子犬です．

「さあ，テディ，何して遊ぼうか？　でも困ったわね．テディは，百人一首もトランプもできないし．お話しでもできればいいんだけど...」と，アリサは，いまはおとなしくなったテディの背中を，ゆくっりとなでます．

「ほんとうに柔らかな毛だこと．さわっているとほっとするし，前に家にいたジュンを思いだすわね」と，アリサはちょっとかなしい顔をします．ジュンは，去年，夏の雷とともに，突然いなくなってしまったのです．ジュンは雷がきらいだったのです．

「きっと，雷さんのいないところで，幸せに暮らしているわ」囲炉裏の揺れる火に，ほっぺと瞳を赤く染めながらアリサは，ジュンと暮らした日々を思いだします．

「でも，大人のいうことってほんとうにヘンよね．どうして，アリサはいろいろなことをおぼえたり思いだしたりするの？　っておとうさんに聞くと，それは頭の中の脳っていうところでおぼえていて，そこで何かを考えると，脳を作っているシンケイサイボウがピカッと光るから思いだすんだ,っていうけれど，嘘よね．テディもそう思うでしょ．大人はいつもむずかしい言葉でだますんだから．だって，いまだって，こうやってジュンのことを考えると胸がいたくなるし，はっきりと思いだすのは，こうやってテディをなでているときですもの．私には，テディの柔らかな毛が，記憶そのものだわ．私の指先が感じて，心臓がいたくなって思いだすのよ．だから，よくわからないけれど，脳っていうのがあるならば，それは頭のなかだけではなくて，私のからだもそうだし，私の身のまわりにもあるわ．

ふう～！　むずしいことを考えていたら眠くなっちゃたわ」

ZZZZZ
ZZ
ZZZZZ

第2章　鏡の家

「テディったら，じっとできないんだから！」

テディがいなくなりました．大声でを呼びますが，なき声も物音もしません．アリサがいる囲炉裏のうしろの方には，いくつかの小さな部屋があって，順番にのぞいて行くのですが，テディはいません．最後の部屋は，お母さんの部屋です．ここには，大きな姿見があります．「やっぱり女性はとことん好いてくれる男性と一緒にならないと．そのためにも，私は美しくなるわ」といいながら，お母さんがいつも見ている姿見があります．テディは，その姿見の前の座布団の上で，すやすや眠っています．

「テディはほうとうに幸せそうだわ．アリサはとってもつまんないのに．テディはきっと自分の姿を見てたんだわ．それで，楽しくてずうっとお話してたのね．鏡ってほんとうに不思議よね．おかあさんは，自分を鏡に映して，自分を見てるって思っているけど，私が鏡を見るとき，そこにいるのは私じゃないわ．絶対！　鏡の中に入れたら，それがたしかめられるんだけど....」

アリサがじっと鏡を見ていると，不思議な気分になってきます．鏡に映る自分の姿がゆらゆら揺れはじめてきます．「ヘンね？」といいながらアリサが手を出すと.....

「あれっ，向こう側に入っちゃうわ！」

アリサは，しばらく探険をしたくなりました．だって，いままでこんなことはなかったのですから．この機会を逃すと，もう二度と鏡の世界に入れないかもしれません．

「えぃっ！」アリサは鏡の国に転がりこみます．そこは，アリサのいつもいる世界とほとんど同じです．でも，まわりを見わたすと，少しようすがちがいます．

「やっぱり，そうだったんだわ」とアリサは，小さな赤い唇をきっと結んでうなづきます．それというのも，いつも，鏡を斜めから見ると，ちゃんと向こうのようすがのぞけるのですけれど，のぞかないときは絶対にようすがちがう，と思っていたのです．「私が見ていないときも，同じようなかっこうで，扉は扉で，椅子は椅子なのかしら？　あやしいわね．きっと，私が見ていないときは，ふらふらしてい

るに違いないわ」

やはり，アリサが思った通りでした．正面から見たときに鏡に映らない椅子はだらんと寝転がっていますし，箪笥はだらしなく壁にもたれています．みんな，アリサが，じっと見ていると，あわてて，気をつけの姿勢になります．ただ，見ているのが鏡の国にいるアリサだとわかると，なんだという顔をして，また，だらしないかっこうに戻ります．

ほかにも，おかしなことがあります．かの女は，表玄関から靴をはいて出ようとします．しかし，いつまでたっても玄関に近づくことができません．それどころか，反対にある土間がどんどん近づいていきます．もう少しで，土間においてある石の流しに足を突っこみそうになりました．

「あっ！」

流しを避けようとして，足を引いた途端に，そこから流しを飛び越えて外に出てしまいました．

「おかしいわ．でもここは，鏡の世界だから，きっとすべてが逆なんだわ．しばらくすれば，きっと慣れるわね.」近くに行きたければ，遠くに行けばいいし，遠ざかりたければ，近くによればいいのです．

「でも，鏡の中っていわなくても，ふつうそうよね．大人は気がついていないけれど，〈多いは少ない〉で〈少ないは多い〉の．いつも，わかってくれないのよね．お食事のあとで，いちごを食べるとき，アリサが，ワッ，たくさんのいちご！ っていうとき，少なかったらいやだってことなんだよね．アリサが部屋が暗いわっていうときは，部屋を明るくしてほしいっていうことなんだよね．大人ってどうしてそれがわかんないのかしら？ こんなこというと，お父さんいつもむずしそうな顔をして，言葉は文字通りの意味しかないっていうんだけど嘘よね．ほんと，大人は分ってないわ．とくにひどいのは，おじいちゃん．本当に強引なんだから．わしが，話す言葉の意味は，わしがこうと選んだことだけを意味するんだなんて，傲慢だわ」

アリサが，裏山の道を下って行くと (ほんとうは，上っていたのですが)，平らな所に出ます．「あっ！」そこには，とてもとても大きな家が一軒，雪を押しのけるように立っています．

「おかしいわ．こんな所に家なんかなかったわよね．ほんとうに家があったら，もうとっくに気がついているはずだわ．このあたりはいつもジュンとお散歩していたんだから．でも，どんな世界も可能よね．一体だれが住んでいるのかしら？ 隣のクリプキ家の人みたいに，すぐに人を指さして，やいやいいう人だったら，アリサ嫌だわ」

アリサは，ちょっとびくびくしながら，それでも自分を勇気づけるように，歌いながら，手を振って歩いて行きます．

「あれっ，どんどん家が小さくなって行く！」

アリスは，近づいているはずです．もう，すっかりわかっていますから，遠ざかるは，近づくだってことを．たしかに近づいているはずですが，小さくなって行くのです．「不思議だわ！」

どんどん近づきながら，家はどんどん遠ざかっています．えぃっと手を伸ばすと，玄関に触りそうな距離になったとき，すっかりその家は，ふつうの大きさでした．小さな声でアリサは，「こんにちは」というと....

第3章　七福神

ちょっとおかしな家でした．「どうぞ」という声にすいよせられるように中に入っていくと，大黒天，恵比寿，毘沙門天，福禄寿，寿老人，布袋の七福神のうちの６人がいます．

「お客さんがいらっしゃるとは珍しい．いつもはわれわれがでかけるのだが．自己紹介しましょう．私は，インドのマハーカーラ，大黒天です」

どっぷりとふとって，ほんとうに，お金持ちのように見えます．

「そんなこと，聞かなくても分るわ」

釣り竿を持っているのが，恵比寿さん（あれっ，今日は鯛をもっていないわ)．こわそうな顔をして，法塔と法棒を持っているのが毘沙門天さん．大きな袋を背負っているのが布袋さん．福禄寿と寿老人はそっくりです．

「あれっ．どっちが福禄寿さんだっけ？ まあいいわ．名前がなくてもお話はできるんですもの．それにほんとうは，１人かもしれないし」

ときどき，福禄寿と寿老人は，影のように重なって１人になったかと思うと，また，２人になったりするのです．アリサは思います．

「たいしたことじゃないよね．テディのことをとっても好きだけど，私のベッドでおもらししたりすると，とっても憎たらしくなったりするのと同じよね．みんないろいろな姿をしているのよ」

こわい顔をした毘沙門天がいきなり喋ります．

「什麼物恁麼来？」

「えっ，シモブツインモライって，なんなの？ それって日本語なの？」

アリサの言葉に真っ赤になった毘沙門天が，「そちに聞く．什麼物恁麼来？」

「こわい．今にも，左手に持ってる棒が飛んで来そうだわ．一体なんなのよ？」

「あなたは，誰で，どこから来たのかって聞いているんだよ」と，布袋さんがやさしく教えてくれます．

「それなら，そういってくれればいいのに．私はアリサで，鏡の向こうから来たの．毘沙門天さん」

ポコッ．ポコッ．ポコッ．ポコッ．ポコッ.....

「えーん」 アリサは泣きまねをしました．アリサは泣きまねがとっても得意なんです．それに，その棒が当たっても何も痛くありません．音がするだけです．

「なんで，私を叩くのよ？」

毘沙門天は，さらに赤くなって，「答えになっていないからだ．アリサとは何だ？ そのアリサという名前を持ったおまえは何者だ？ 鏡の向こうとは何だ？ 説明になっていないぞ」

「アリサはアリサじゃないの！ 鏡の向こうは鏡の向こうだわ」と思いましたが，痛くないとはいえ，叩かれるのはこわいので，一生懸命考えます．

「アリサは，牛瀬村1832番地に住んでいる7才の，とても可愛くてみんなに大事にされている女の子．おかあさんとおとうさんがいて，おとうさんはちょっとお酒のみで，それも久保田ってお酒しか飲まないから，困っています．あっ，これは関係ないわね．でも，なんだかこうやって私のことを説明していくと，だんだん私のことじゃなくなっていく見たいね．だって，何かを説明するというのは，その他のことを説明しないということですもの．困ったわ」

アリサは，両方の手の指を胸の前でもじもじさせながら，一生懸命考えます．「う〜ん，こんなこと考えていたら，アリサ頭痛くなっちゃう．もう，いいわ．まともに相手するからいけないのよ」

「実は，私は弁財天です．どこからも来ていないわ．さっきからあなたと一緒にいるじゃない」といって，毘沙門天のほっぺをぱちんと叩くと，逆に叩き返されないように，あわてて頭の上に手をやって，目をしっかりつぶりました．

ところが，毘沙門天は，「汝に30棒を放す」といってそそくさと，部屋に入っていきます．なぜか，無事にすんだようです．

「わからないけど，毘沙門天さんが納得するなら，会話は成立よね．でも，こんな暴力に訴えるなんてコンリンザイ嫌だわ」

「こっちにおいで．弁財天さん」と，福禄寿さんか寿老人さんかが手招きします．そこは，ちょうど，毘沙門天さんが入っていった部屋です．せっかくうまくいったのに，また，会うと思うとゆうつでしたが，こちらに来てからずいぶんと大胆になったアリサは，(福禄寿老人)/2 について部屋に入っていきました．

「どうぞ，こたつに入ってください」と，大黒天さん．だけど，入るところはありません．アリサは「ちょっと，移ってくださいませんか？」と，ていねいに頼みました．みんな，はいっといって，ビデオカメラに顔を向けて，ニコッと笑うと「うつった」といって「パチパチ」と拍手をします．ちょっとむっとしましたが，かまわず，床の間の前に無理やり入ることにしました．

床の間の掛け軸が少しヘンです．ふつう，掛け軸には，「色即絶句」とかかっこいいけど，意味のわからない言葉が書いてあるのに，そこには「今年の目標：親切(しんせつ)」としか書いてありません．

「そうよね．言葉はわかりやすくないとね．でもアリサ，別にふりがながなくても親切くらい読めるし，意味も知っているわ．それに，みんな，全然親切じゃない．なんでもそうだわ．Aだって標語を出す人に限って，〜Aなのよね」と，思いながら，そして，鏡の国の中ではどうしてこんなむずしいことばかり考えるのかを考えながら座りました．

こたつの上には，お雑煮とおいしく焼き上がった鯛と蜜柑が出ています．お雑煮は，白味噌の蕪入り丸餅です．

「やっぱりお雑煮はこれでなくっちゃ．鯛もとってもおいしいし．でも，アリサ，目が怖いから恵比寿さんの方へ向けちゃおうっと．蜜柑も食べたいけど，ちょっと遠い場所にあるし，そばに座っているみんなに何か意地悪されそうだから，あきらめようっと」

恵比寿さんは，お正月用にせっかく飾りつけをしてあった菊の花を噛りながら1人さみしそうでした．

第4章 熊さんと薔薇と

「いやあ，まいった．ちょっと，外へ出ない．マダラ薔薇が咲く頃だ．見たくなった」と，布袋さんが唐突にアリサに声をかけます．

「そうするわ」 アリサはそのマダラ薔薇を見たくなりました．だって，マダラ薔薇なんて見たことがないんですもの．慎重に，進む方向を間違えないようにしながら，布袋さんと外に出ます．しかし，そんなことを気にすることなく布袋さんは，ずんずんと，草むらの中に，走って行きます

「たいへん，追いつけないわ！」

そのうちに，布袋さんの姿が少しずつ変わって行きます．この世界は，なぜか揺れています．見るものがはっきりとしないのです．しかし，それはアリサが鏡に入る前にいた世界もそうだったかもしれません．よく考えると，意識するものすべて，春先の発光する風を通して見る世界のように，いつも揺らいでいたようにも思います．

布袋さんは，ますます大きくなるとともに，手足がふと

く短くなり，全身つやびかりする黒い毛で覆われはじめました．袋はもう持っていません．そのかわりに，とても強そうなウオッカの瓶を持っています．そういえば，ほんのりと顔が赤いようです．「あれっ，へンだわ」とアリサが思っているうちに，なんと，布袋さんは熊の姿に変わってしまったのです．

「ちょっと，待って．熊さん，あっ，ごめんなさい．布袋さん！」

「えっ，なんだい．君が好きなように僕のことを呼んでくれればいいんだよ．なんだったら，タオのプーさんでもいいし．そんなことどうでもいい」

「じゃあ，間違えないように，熊さんと呼ばせてもらうわ」

会話をするために，少しのあいだ，熊さんは走るのをやめました．そうすると，風景はどんどんと飛んで行きます．だけど，もうアリサには慣れっこです．

「"恋人が語る，心の薔薇について"，という詩を知っている？」と，突然熊さんがたずねます．

「知ってるわ．おとうさんから聞いたもの．アリサのおとうさんはね，何でもおぼえているのよ」

ちょっと，どうでもいいことをおぼえすぎよねと思いながら，アリサは熊さんにうなずくと，その詩を静かにうたいはじめます．

みないつもと違い，そして壊れている　すべてはすり切れ，もう古くなった
道沿いの子供の泣き声，木材運搬車のキーキーいう音
農夫が何度も鋤く，冷たい土くれの音
それらは，私の心の奥底で，薔薇を咲かせるあなたのイメージをおかしくする
あまりにもおかしくなり過ぎて，イメージは形をなくし，
何がちがっているのかすらわからない．

私は，そのイメージを新鮮にしたくて，
緑の丘にすわる
大地と空と湖とともに，
もう一度作る，金の小さな箱のように
私の心の奥底で，
薔薇を咲かせるあなたのイメージを夢見て

W.Yeats: "The Lover tells of the rose in his heart"

「ほんとうにいい詩だ．ちょっとかっこよすぎるのが難点だが.」

「物質とはイメージの総体である．イメージは，実在論者のいう実在の地点と観念論者のいうイデアの中間にあって，それこそが，いま，われわれが対象にすべきものだ」と，またしても，突然熊さんは喋りだします．アリサがそこにいることなどお構いなしに，１人で喋ります．

アリサは，「う～ん」とうなると，「なにそれ，ジツザイとかカンネンとかって．教えてくれますか？」とていねいにたずねて見ました．ところが，熊さんは，何も返事してくれません．「ねぇ，教えて！」とアリサは太い腕を掴んで揺するのですが，何も答えてくれません．「ほんと，隣村のユイマ君と同じだわ．かんじんなところに来ると黙ってしまうんだから」アリサは，ちょっとだけ，むっとします．

「ねぇ，ねぇ，教えてよ．熊さん！」

「熊さん，ねぇ！」

第５章　夢をみたのはどっち

アリサは，熊さんを大きく揺さぶり続けました．そうすると，さっきは大きくなった熊さんが，こんどは，しだいに小さくなって，しかもどんどんと犬のかっこうになっていきます．

ZZZZZ
ZZ
ZZZZZ

「あれっ，テディじゃない？」

子犬のテディは，ゆすぶられているものですからちょっとおびえた目をして，じっとアリサを見つめます．

「アリサは夢を見ていたのかしら？　それにしても，いろいろ考えてばっかりいたわ．疲れちゃった．ほんと！　あたりまえのことがみんなわかってないんだから」

あわてて，テディを下にそっとおろすと，ちょっと不思議なきもちになりました．なぜって，時計を見ると，アリサがこの部屋に来てからたった数分しかたっていないのです．

「夢だからかしら？　でも，この部屋に来たのも，夢の中のようだった気もするわ．まったくヘンね．それにしても，どうして，こんなことが起こるのかしら？　きっと，この『鏡の国のアリサ』を書いている人が悪いのよ．ほんと疲れちゃうわ．たぶん，いいたかったのは，非現実的事象および非人間的象徴としての動物等を用いた，主として視覚的言語の経済性および必然性による換喩的，隠喩的修辞技法によって，および"どら"へたくそなパロディを通して，私たちが科学の対象としている実在以外にも，生活世界にはいくらでも実在があって，それは共有可能であるという点においてまったく同一の実在であるということをいいたかっただけなんでしょうけど，ほんとうに文章力がないわね！　私ならひとことでいえるわ.」

アリサは，ここまで読んだあなたの心に，いま*います*！

オブジェクト指向への密教的接近

青木淳
(SRA, Boulder)

1. 栗原さんへ

お手紙拝見しました．ご指摘の通り「あくまでもプログラムを作る者の視点を忘れてはいけない」というのが，私の行動の指針であり，私の判断の基底でもあります．

ソフトウェアに携わる方々を学者と職人とに大別すれば，私は明らかに職人の立場に立ちます．１年間に少なくとも１万ステップ程度のプログラムを実際に組みあげるのでなければ，ソフトウェア作りに関して，偉そうな言及をしてはいけないと思っています．

学者と呼ばれる人のなかには，ここ何年もの間，１行のプログラムも作っていないのに，その道で第一人者であるような言動をされる方がいます．そのような方が参加して行われる議論は戯れの論であり，一切無戯論如来 (文珠師利菩薩) は決して微笑みません．

大方の学者さんは，作る手が記憶している体験や経験を軽視するようです．本や論文に書かれていることの方を，目の前にあるプログラムよりも大事にします．多くの学者さんたちとつきあいましたが，とても感心できませんでした．しかし，なかには，私などの話をよく聞いてくれて，学説などに縛られず，実際に手を動かしてプログラムを作る先生もおりました．このような学者さんは，きっとほんとうの研究者なのでしょう．

また，管理職と技術職についても同様です．私は，課長や部長などの管理者が棟梁 (職匠) であってほしいと思っています．技術のことがわからない人が，一つの技術者集団の支えとなる人であるはずがありません．癖のある人々を組むことは，一様でない界面を持つモジュール群を組むことと似ており，プログラムさえ作れない人が，いわんや組織作りをやです．

さて，私はオブジェクト指向を筏にして，ソフトウェアの激流を渡ろうとしている優婆塞 (うばそく) です．栗原さんからのお手紙のお礼といっては何ですが，近ごろオブジェクト指向について感じていることを書いてみたいと思います．加えて，私がそう感じるように至った仏典 (九想詩と理趣経) の現代語訳(私の拙い訳ですが) を添付します．ソフトウェア技術者は幻想 (仮想現実＝モデル) を作る人であり，それが決して虚しく (空しく) ないというのが，私の考えです．

2. オブジェクト指向にちなんで

現在，私のまわりには米国のデキル方が多いので，その方々と数々のレビューなどを行なうと，自分との相違に気づかされます．同僚の酒匂さんと私の違いも，この相違に起因することがわかっています．多くの方々のオブジェクト指向分析設計の考え方やソフトウェアの捕捉の仕方も，この相違に近いものが感じられます．

たとえば，身近なロイドさん (Lloyd G.Williams さん，Colorado 大学教授を経て，いまは独立コンサルタント) や，ブレントさん (Brent Reeves さん，Colorado 大学で中小路久美代さんと同窓，Gerhard Fischer 先生のお弟子さん) と私とを比べると，オブジェクトに対するとらえ方がまるっきりちがうことに気づきます．どうも，ロイドさんやブレントさんはオブジェクトを客観的にとらえているようです．SRA での同僚の酒匂寛さんも同様です．一方，私はオブジェクトを主観的にとらえる癖があるようなのです．

例をあげれば，旅費精算のシステムに登場する伝票を理解するのに，"自分で伝票になってみなきゃ伝票はわからん" というのが私だとすると，伝票の特徴や性質を個条書きにして列挙し，属性や操作そして抽象化した上位のオブジェクトを導出するのが，酒匂さんをはじめ，ロイドさんやブレントさんのようです．私は属性や操作やスーパクラスを策定するのに時間がかかるのに対して，かれらはアッという間にやってのけます．私の最悪の場合は，プロトタイプを作ってみなきゃ，何ともいえないという状況になることが多いのに対して，かれらは見切りが早いというか .... 何といったらいいのでしょうか ....「必然」が多いのですね．

なんだか，私のオブジェクト指向分析設計は，プロセスがちがうようです．もっと正確にいえば，私のオブジェクト指向分析設計は，ブーチさんやランボーさんらが提唱するプロセスとは根本的に異なっていて，結果を説明するために，かれらの手法を使っているのではないかということです．「自然」に出たものを「必然」のように説明しているといっても，過言ではありません．ロイドさんやブレントさんと，通じない英語で分析設計結果をレビューするときに，私の分析設計の結果にマジックがあるといわれるゆえんです．英語がへたで説明できないのではなく，論理で説明できないところがたくさんあるのです．酒匂さんと議論になるのもこの点です．

それから，Smalltalk のテキストエディタでプログラムを入れているさいに，私は便利なエディタの機能を使わずに，いちいち手で入力することがあります．それでなくとも，タイピングの早い技術者に比べて，私のキーストロークが遅いので，見ている方はイライラするようです．便利な編集機能を，私が知らないわけではなく，知っていながら，わざわざ面倒くさい方法で入れるのには，私なりの理由があ

ります.

Smalltalk のプログラムを書くのは, あるオブジェクトのメソッドを作るわけですから, そのオブジェクトになりきりたいので, 面倒をいとわず, ていねいに一文字一文字を入れて, そのオブジェクトと同化 (入我我入：他が自らに入りて自らが他に入るように) しようとするのです."伝票にならなきゃ伝票はわからん" という真意が, ここにこめられるわけです. 禅問答のような説明の仕方で申し訳ありません.

ブラウザを使うときも, あるクラスを選択したり, インスタンスやクラスのボタンをクリックしたりするときに, なりきるオブジェクトを変更しているという立場に立つのが私で, 多くの方々は, ブラウザを調べるツールとして使っているのですね. マウスで選択したものになるのではなく, あくまでもオブジェクトの世界から遊離した自分 (神) が客体を観ているように使うのです. このことは拙書にも書きました. [1][2][3]

オブジェクトモデルやダイナミックモデルをうまく構築するのに, 神の目 (パースペクティブ) のようなものを必要とするのが, 一般のオブジェクト指向技術者だとすると, どうも私は対象とするオブジェクトになりきって, まわりのオブジェクトを自分との距離 (へだたり) として認識し, その自分に他のオブジェクトを映し出して, そのようすを, オブジェクトモデルやダイナミックモデルにやきなおしているようです.

オブジェクトモデルを紙に書いてみると, 自分が神になったかのごとく, 遠近法的な視点が得られます. だから, 大方の技術者は, 紙面に書かれたオブジェクトモデルで満足するのだと思います. 一方, 私は常に注目するオブジェクトを決めて, そこからまわりのオブジェクトを見るように, オブジェクトモデル図を翻訳しているようで, 多面的で, 幾何学的で, 次元が多くて, 羯磨曼荼羅(かつままんだら) みたいになっています. 情報を遠近や配置で表現しようとしているのです.

多くの方が書かれたオブジェクトモデルのなかからは, この主体的なオブジェクトの世界観が感じられないのです. ちょうどロイドさんやブレントさんのオブジェクト観のように. または, 酒匂さんの見切ったような達観したオブジェクトモデルのようです. 比喩的に表現すれば, 冷たくて寒く温かみがない分別智のごとくで, 無分別智ではないように感じられます.

むかし Blizzard というプロジェクトを始めたとき, 各種のオブジェクト指向の手法を調査して, 直観的に 3 次元グラフィックオブジェクト指向クラスライブラリがほしいと思ったことがありました. いまでは, その理由がとてもはっきりしています. あるオブジェクトからまわりを見渡せ, 注目するオブジェクトを渡り歩け, それを幾何的にとらえる認知システムのような分析設計ツールが必要だったのです. しかし, これを実現するには, 現在のコンピュータを持ってしても無理がありますね. コンピュータの情報表現は, 幾何学が不得意ですからね. 形と形の演算がきわめてむずかしいのです.

羯磨曼荼羅は「自分に由って」いますので, 神の目 (必然) を必要とせずに, とても自由(自然) です. 臨済録に頻繁に現われる自由とは, このことを指すのだと思います. 制限や制御から解き放たれるという消極的な意味のリバティを, 積極的な意味の自由と訳したのは, 先人の誤訳だといわざるを得ません. オブジェクトが自分に由って立つのは, この時空が帝釈天 (たいしゃくてん) の宮殿を飾っている網の目の鏡の珠 (珠網) のように, その珠にそれ以外の珠がすべて映って, その反射のなかに自分の珠も映って, フラクタルになっているからです. 仏教では, これを業熟体 (ごうじゅくたい) といいます.

この帝釈天の宝珠の網のような縁 (遠近や配置) によって, オブジェクトがとらえられているということを, 多くの方々が得心することを望みます.「随所に主となれば, たちどころにみな真なり」です. これが私が欲しているソフトウェア知識表現で, これから先, 私が一生をかけて作ろうとしている随所に主となるためのツールでもあります.

[1] 青木淳, オブジェクト指向システム分析設計入門, SRC
[2] 青木淳, オブジェクト指向分析設計テクニック, SRC
[3] 青木淳, Smalltalk ソフトウェア開発, SuperASCII 91.6-92.5

3. 空海　性霊集　九想の詩

＜新死の相＞

この世の月日は短く, 死後の黄泉(よみ) の年月は長い. 現世の命の早く過ぎゆくことは, かげろうのごとく, たまゆらを生きて, やがて死に, 去ってゆく. あわただしきこと風雲のごとくにして, 貧りの住家を辞し去り, かこい火のたち消えるように, 欲望の城をやめ, 去ってゆく. 命はすでに満了して, 閻魔庁の死籍簿に名が記されたのである. 生きとし生けるものの寿命は, 霞のようにはかなく, 三十三天の力を借りても, つぐないの手だてもほどこしようがない. こうして, 詩に詠みあげて, 悲しみ歎くばかりである.

＜膨脹の相＞

墓地の丘は虚しく広く, 人里を遠くはなれて, 訪れる者とてなく, 白く清らかな月光のみが, 広野をわたって過ぎ去ってゆき, 風はしめやかにして, 色あせた樹々の葉に吹きわたる. 悲しみをこらえて四方を見渡せば, ただ一個の

屍がある．裸のままに松林の丘に臥して，髪はざんばらで，長い夜を経て死去した人であり，仏法の半分を求めんがために，身を捨てた雪山童子のような死にざまではない．生前には獣や魚の肉を飽食したであろうに，いまでは百獣の餌食となっている．

＜青瘀の相＞

地獄の使者の手からは，永久に逃れるすべはなく，墓穴は深く底なしである．美しいかんばせはすでに光を失い，宝鏡のようであった容貌は自ら壊れて形を失ってゆく．屍は風に吹きさらされる灯火のごとく，また，花の散り落ちた枝にも等しい．日が経つにつれて，しだいにくずれおち，月が毎夜来たりて，さらに黒ずみを増してゆく．白いウジが穴のなかにうごめき，肉のついた骨のうえには青いハエがたかっている．愛した者の昔の面影はなく，その変わりようを悲しみ，またかくも醜悪な体をいとしんだ我が身がはずかしい．

＜方塵の相＞

さまざまなものから成る人体は，離散にあたってまことにおぞましく，体と心を機能せしめる諸要素の理(ことわり)は，滅びやすくて頼みにできない．体力や気力は肉体より去って帰らず，水分や蛋白なども朽ちはてようとしている．体は青く黒ずみ，腐ってただれた雑草のように膿んでくずれてくる．体の穴という穴から流れ出る膿み汁は，あたり一面に散って，臭くて汚いことのきわみである．猛獣は傍らに寄ってうずくまり，不吉な悪鳥が鳴いては肉をさらってゆく．肉体はやがてこの野の塵となって残るが，魂はいったいどこへ帰り去るのであろうか？

＜方乱の相＞

よこしまな煩悩は，断ちがたい網であり，迷いの世界に輪廻する体に，不変の果報などあるはずもない．飛ぶ矢の速いように，人の命も速く尽き終わり，一身の空なることは，朝露のようにはかない．その玉のような美貌も，いまは膿血にくずれ，麗しい肢体も腐り壊れてゆく．臭気は風にのって，遠方までも届き，腹部の脂肪も沸き流れて，炎のようである．なお残る美しい衣服は，肉の腐爛に恥じるようで，立派な枕もいまは見るかげもない．歎き悲しんでもなすすべもなく，涙を拭いつつ，往路と違った道を帰るのである．

＜蝎骨連の相＞

世のなかの人々は，死の影におびえて恐れるが，その無常の影から脱却するためには，人目の届かないところで善行を積むしかないことを知らず，夢のなかの蝴蝶のように，世間の浮き雲にのって飛ぶだけである．人の命は，電光よりも速く過ぎゆき，やがては松下の塵と成りはてる．生前には都会で栄華をきわめた人も，いまでは白骨と化して横たわっている．死者を甦らせるという黄色い白鳥も，あなたの名前を呼んではくれないし，また，ここは寂しい山野だから，死者を復活させるという青い柳の田畑もまったくない．春が来て百花が競って咲き誇り，ただよう香りも芳しく，明るい月がむなしく山を照らしている．永遠にものさびしさだけが支配して，ついに白骨の人はふたたび春を知るよしもない．

＜白骨連の相＞

ここはさびしいところで人影もなく，しめやかにして人里もはるかである．朽ちはてた一連の白骨が，はっと眼に入るように，沢のなかに横たわっている．松や柏が茂ってほどよい影をつくり，乱れ茂る茅がしめった白骨の周りをおおっている．常に風雲にさらされており，さらに霜露がしたたり加わる．日が経つにつれ，白骨はますます枯れてゆき，年を越す毎に白さを増してゆく．死者を復活させようとして，青い柳を植えても，甦るすべはなく，名医を招いても，もはや施す手立てもなかろう．

＜白骨離の相＞

人の永らえることはなく，夢がむなしく消えるのと等しい．人が住む欲界は水の泡と同然である．人間が現実に住んでいるこの世界は苦しみ多く，厭い離れなければならないところであり，楽しんで生活できるところではない．人の皮膚や血液は，月が欠けて，また満ちて丸くなるのとは異なって，もとのようにならないし，人の骨は青い柳にふたたび花が咲くのとは違う．人の爪や髪はそれぞれに草にけがされて溶け，頭蓋や頸骨は東西に離散する．散乱した骨の半ばを落葉がおおい，その骨のあたりに菊の花が秋に咲くけれども，どうして美しいと褒められようか．あふれ落ちる涙を禁ずることができず，いたずらに故人を憐れんで声をあげて泣くのみである．

＜成灰の相＞

自然の山川は，永く万世にわたって存在するが，人間の命は短く，長く生きても百年である．身体のすべての骨は消滅し，棺桶すら朽ちはてて灰塵と化してしまう．魂はいまや依りどころがなく，おそらくは地獄に墜ちて，塚の主は不在であろう．せめて碑文にその名を書き留めて偲ぶよすがとし，塚穴の底にあなたをおさめることにしよう．月日の移るにつれて，骨もやがては黄白の土と化し，ついには風に散って山に帰るであろう．かの三乗の教えのみが常住不滅の宝であり，この教えにしたがって修行を積まねば，八苦に悩むただの人にとどまるのみ．かつてこの人の活動を支えていた六識は，いまいったいどこにあるのであろう．その身体を支えていた諸要素は破滅して，いまは名前が虚しく残るのみである．冬には苔が塚の周りを青く色どり，夏には草がぼうぼうと塚のうえに根をはって生い茂る．だ

れが置いたのか, 墓塚の袋のなかには, 供えの食物がいまなお残っており, 松の木のしたに見える髪の毛はなお黒く見える. 丘のうえには青黒い雲が集まって薄暗く, 夜の松風がひゅうひゅうと鳴りわたっている.

4. 理趣経

正しくは「大楽金剛不空三摩耶経 (大いなる楽は金剛のごとく不変で空しからずして真実なりとの仏の思いを示せる経)」といい, または「般若波羅蜜多理趣品 (真実なる智慧の道理)」ともいう. 唐の都長安の大興善寺の三蔵沙門なる大広智の不空が代宗皇帝の詔を奉じて訳す.

はじめに (序説)

このように私は聞いた. あるとき, お釈迦さまは, すべての如来が持っている金剛のように不変不壊な力によって, きわめてすぐれた境地に到達された. そして, すべての如来の智慧を象徴する宝冠によって, 頭を浄められ, 三界の主となられたのである. このように, お釈迦さまは, すべての如来の一切のものを知る最上の智慧を得られ, 心と体の完全な自由を得られた. そのうえ, お釈迦さまは, すべての如来の一切の行為が平等な慈愛によることを示す種々さまざまな事業を遂行された.

かくて, お釈迦さまは, あますかたなく, すべての人々の世界における, すべての人々の願いを, みなことごとく完成させて, 過去と現在と未来との三世のすべてのときにおいて, 御自分の身(からだ) と口 (ことば) と意 (おもい) との三つの働きは, いささかも停滞するところがなくなった. こうして, お釈迦さまは「あまねくすべてを照らすもの (大日如来)」と一体になられたのである.

この大日如来は, この世界のはてにある他化自在天 (他を救うことが自在なところ) の王宮におられる. ここはすべての如来が常にゆきかい, その優れたさまが褒めたたえられている大宝殿である. くさぐさの宝が錯綜し, 吊りおろされた鈴や鐸そして絹旗が微風に揺れ動き, 宝珠のついた髪飾りや装身具, 半月や満月の形をした宝石などが, いとも美しく飾られている.

この美しい宮殿のなかに, 大日如来は八十億もの多くの菩薩とともにおられた. その菩薩のなかでも, 金剛手菩薩大士 (金剛のように堅固な菩提心の体現者), 観自在菩薩大士 (大いなる慈悲の実現者), 虚空蔵菩薩大士 (一切の事物を包容してあらゆる存在をさまげない福徳者), 金剛拳菩薩大士 (大いなる三密の行者), 文珠師利菩薩大士 (最上なる智慧の完成者), 纔発心転法輪菩薩大士 (すばやくそしてたくみな説法者), 虚空庫菩薩大士 (無尽にして無余なる供養者), 摧一切魔菩薩大士 (外に憤怒を内に悲憂をいだいて悪を摧く奉仕者) の八大菩薩は座の中心であり, これらの大菩薩たちに尊敬されて, 取り囲まれて, 大日如来は正しい法 (おしえ) を説かれたのである.

その法は, 初めによく, なかによく, 終りによく, 表現の内容も巧妙で, いささかの誤りもなく円満であり, 清らかで澄みわたったものである.

第1段　欲望は浄らかなり (大楽の法門)
金剛さったの章

その最初の教えは「一切の存在 (法) は清浄である」という教えである. このことについて, 大日如来は十七の清浄なる菩薩の境地をあげて, 次のように説かれた.

1. 男女交合の妙なる恍惚は, 清浄なる菩薩の境地である.
2. 欲望が矢の飛ぶように速く激しく働くのも, 清浄なる菩薩の境地である.
3. 男女の触れ合いも, 清浄なる菩薩の境地である.
4. 異性を愛し, かたく抱き合うのも, 清浄なる菩薩の境地である.
5. 男女が抱き合って満足し, すべてに自由, すべての主, 天にも登るような心持ちになるのも, 清浄なる菩薩の境地である.
6. 欲心を持って異性を見ることも, 清浄なる菩薩の境地である.
7. 男女交合して, 悦なる快感を味わうことも, 清浄なる菩薩の境地である.
8. 男女の愛も, 清浄なる菩薩の境地である.
9. 自慢の心も, 清浄なる菩薩の境地である.
10. ものを飾って喜ぶのも, 清浄なる菩薩の境地である.
11. 思うにまかせて, 心が喜ぶことも, 清浄なる菩薩の境地である.
12. 満ち足りて, 心が輝くことも, 清浄なる菩薩の境地である.
13. 身体の楽も, 清浄なる菩薩の境地である.
14. 目の当たりにする色も, 清浄なる菩薩の境地である.
15. 耳にするもの音も, 清浄なる菩薩の境地である.
16. この世の香りも, 清浄なる菩薩の境地である.
17. 口にする味も, 清浄なる菩薩の境地である.

なにがゆえに, これらの欲望のすべてが清浄なる菩薩の境地となるのであろうか? これらの欲望をはじめ, 世のすべてのものは, その本性は清浄なものだからである. ゆえに, もし真実を見る智慧の眼である般若を開いて, これら一切をあるがままに眺めるならば, あなたたちは真実の智慧の境地に到達し, すべてみな清浄でないものがないという境地になるであろう.

金剛手菩薩よ. この清浄なる境地を生み出す真実なる知慧の理趣 (みち) を聞かされたならば, すべての障りは速やかに消え去り, 光り輝く菩提 (さとり) の道場 (にわ) に入ることであろう. すべての障りとは, 貪りや瞋りなどの煩悩

の障り, 法 (教え) を聞きえない障り, 悪業のみをなす障り等で, これらすべての障りを積み重ねても, 地獄の境涯に落ち入るようなこともなく, 重い罪を作ったとしても, それを消滅することはむずかしいことではない.

この正しい法門をただ聞くだけでなく, よく身に受持し, 日に日に読誦し, よく心に思惟すれば, この世において, 父母によって生まれたこの身体のままで, すべてのものの片寄らぬ本質を見きわめ, 金剛のごとく不壊で安らかな境地になるであろう. こうして, あなたたちは, なにものにおいても, 拘束を受けることなく自由となり, はかり知れぬ快楽と歓喜に満ち満ちることになるであろう. そして, 十六の大菩薩によって象徴される十六の生の段階 (禅定) を進んで, 金剛のごとく不壊なるもの (金剛さった) を本質とする私「あまねくすべてを照らすもの (大日如来)」の境地を獲得することであろう.

このような大日如来の説法を拝聴した金剛手菩薩は, すべての如来の大乗 (こよなき) 現証 (さとり) の三摩耶 (思い) を示し表わす曼荼羅に住している持金剛者のなかにあって, 殊に勝れた者となり, この世の悪を調伏して余すところがない一切義成就菩薩となられた (一切の道理にかなったことを成し遂げた).

かくて, 金剛手菩薩は, この「欲望はすべて浄らかなり」という教えを表わすために, 顔を和らげ, 微笑まれ, 左手に教えを象徴する金剛慢の印を結び, 右手で菩提心 (自分をよく知ろうとする心) を象徴する五股の金剛杵 (五股杵) を揺り動かして, 勇み進みゆく勢いを示された. そして, 大いなる楽が金剛のごとく不壊で空しからずという境地を示すために, この教えを一字で表わす聖音「フーン」を唱えたのであった.

第2段　さとりはすべてにあまねし (証悟の法門)
大日如来の章

続いて, 大日如来は, すべての如来の現等覚 (さとり) を生み出す教えを説かれた. それはすなわち, あらゆるものの本性は, 悩みを離れ, 寂静であるという教えである. これこそが真実なる智慧の理趣 (みち) である.

一つには, 金剛のような現等覚である. これは, すべてに普遍で平等なることを知る悟りであり, 金剛のように堅固で不壊な大菩提である. これは金剛平等 (大円鏡智：澄んだ水の表面に喩えられる阿しゅく如来の智慧) といわれる.

二つには, すべての利益が等しいことを知る現等覚である. この大いなる菩提は, すべての利益に基づき, 慈悲によるがゆえに差別がない. これは義平等 (平等性智：澄んだ水の水面が同じ高さであることに喩えられる宝生如来の智慧) といわれる.

三つには, 法 (教え) が平等であるという現等覚である. この大菩提は, ものの自性が清浄であり, すべてのものにあまねくゆきわたり, その教えはすべての人々を教化するからである. これは法平等 (妙観察智：澄んだ水の水面にすべてを映し出すことに喩えられる阿弥陀如来の智慧) といわれる.

四つには, すべての業が平等であることを知る現等覚である. この大いなる菩提は, 私欲にとらわれず, すべての分別動作をそのままに, 如来の種別を超えた平等の境地に通ずるからである. これは一切業平等 (成所作智：澄んだ水がすべてのものの成長を育むことに喩えられる不空成就如来の智慧) といわれる.

この「さとりはすべてにあまねし」という教えを四段にわけて説き終り, 大日如来は金剛手菩薩に呼びかけられた.

金剛手菩薩よ. もしあなたたちがこの四つの悟りを生み出す法を聞いて, これを読み誦え, 身に保ち持っていたならば, たとえいまここで数えきれない重い罪をなしたとしても, 必ず一切の地獄に堕ちるという報いを超えて, そのままで無上正等覚 (さとり) を証することであろう.

かくて, 大日如来は, 第1段で, すべてが大楽であることを示したのに続いて, 第2段では, それを実現する方法を説き終り, これらの四つの現等覚を, より一層に明かにしようと考えて, 顔を和らげ, 微笑まれ, 手に智拳の印を結び,「一切の法は自性平等 (法界体性智：澄んだ水がいたるところにゆきわたることに喩えられる大日如来の智慧)」という教えを一字で表わす聖音「アーハ」を唱えたのであった.

すでに, 大日如来は, 御自分の智慧を主客の両面から説き終られて, この理想の世界を八大菩薩によって象徴される八つの法門として, もっと具体的に説明しようとされた.

第3段　悪にその性なし (降伏の法門)
釈迦牟尼如来の章

大日如来は, したがえにくいものどもを, 心身を制御することによって, 煩悩や悪行に打ちかつという調伏の修行に励む釈迦牟尼如来 (お釈迦さま) の姿になり, すべての法は平等で, 善悪を離れているという最も勝れた境地をもたらす教えを説かれた. これは真実の智慧の理趣 (みち) である.

貪り (むさぼり＝貪欲：自己の欲するものに執着して飽くことを知らないこと) は, その本質からすれば, 善悪の分別も, それに執らわれた表現も超えたものであり, 人によっていかなる善にも活かすことができる. 貪りがかかるものである以上, 瞋り (いかり＝瞋恚：自分の心に逆らうものを怒り恨むこと) も同様で, 善悪の分別も表現も超えたものである. したがって, もし自我に執着することなく,

それを善に活かしきれば，邪悪に打ち勝つための大きな瞋りが生み出される．このように，瞋りが善悪の分別を超えたものであるとすれば，痴しさ (おろかしさ＝痴愚：根本の真理を知らないこと) も同様で，善悪の分別を超えたものである．ゆえに，小さな自我に執らわれず，それを善に活かすときは，これが愚かとか，あれが賢いなど，物事の理非をあれこれといいたてるような，小さな痴しさを超越して，大きな痴しさの境地に至るであろう．

このような三毒と呼ばれる貪りと瞋りと痴しさの悪しき心の働きは，すべて相対的な区別にたった認識に過ぎず，すべてのものは善悪の区別や表現を超えたものなのである．したがって，すべてのものが，その本質において，現象のうえに見られる相対的な区別や表現を超えたものである以上，この本質を知るための真実の智慧の理趣も同様に，かかる区別や表現を超えたものでなくてはならない．

こうして「悪はもともと悪ならず」という善悪を超えた法門を説き終わり，大日如来は金剛手菩薩に呼びかけられた．

金剛手菩薩よ．もしあなたたちがこの「悪にその性なし」という理趣を聞いて，これを読み誦え，身に保ち持っていたならば，たとえ三界の一切の生き物を殺したとしても，決して悪の報いを受けて地獄に堕ちることはない．それどころか悪しきものを調伏 (うちなび) かしたことの果報を得て，速やかに無上正等覚 (さとり) を証することであろう．

かくて，金剛手菩薩は，この悪を調伏する教えを，より一層に明らかにしようと考えて，手に降三世の印を結び，蓮華の花のような美しい微笑みをうかべながら，一方では，はげしい忿怒の表情で，眉をひそめて睨みつけ，鋭い牙を出しながら，法力を以て仏敵や怨敵や魔障などのすべての悪を降伏する(討ち倒す) 姿に変身された．そして，この教えを一字で表わす聖音「フーン」を唱えたのであった．

第4段　ものすべて浄らかなり (観照の法門)
観自在菩薩の章

さらに，大日如来は，善悪が共にその本性において平等にして清浄であることを示すために，「ものすべて浄らかなり」という真理を自由に観る智慧を生み出す理趣 (みち) を説かれた．この境地は観自在菩薩の境地である．これは「四種の不染」の教えと呼ばれる．

この世間における一切の貪欲は清浄である．何となれば，すべてのものの本性は清浄であって，善や悪の貪欲などという区別は現象のうえにあらわれたものに過ぎず，決してその本質にまでさかのぼって汚すことができるものではない．このように貪欲が清浄であるとすれば，この貪欲の不足によって起こる瞋恚も痴愚も，その心に根拠がなく，すべて清浄である (金剛法菩薩).

このように，貪欲も瞋恚も痴愚も，この世のすべての垢 (けがれ) が清浄であるがゆえに，これらの垢から生じる一切の罪の業(おこない) も，その本性は悪ではなく，きわめて清浄である (金剛利菩薩).

また，この世における一切の構成要素も，その本性は善悪の区別なく清浄であるから，これらの構成要素が一時の存在を保つために，仮に組み合わさったもので，すべての生き物は清浄である (金剛因菩薩).

さらに，この世のなかで働いている理性も，宇宙の本性そのものであり，大日如来の最高智のあらわれなのだから，とても浄らかで，現象のうえの区別や表現で汚されるものではない (金剛語菩薩).

このように，貪欲の本質が清浄であり，諸々の貪欲から起こる瞋恚も痴愚も清浄であるならば，これらの三毒から生じる悪業も，本質は清浄であると観なければならない．世界を構成する一切は清浄であり，それを自覚する智も清浄である．この本質の清浄性を証明する四つの見方 (四種の不染) を説き終わり，大日如来は金剛手菩薩に呼びかけられた．

金剛手菩薩よ．いまここで私が説いてきた真理の理趣を，身に受持し，読んで唱え，よく心のなかに考えたならば，たとえこの現実世界のなかに生活し，さまざまな欲望のなかにあっても，その本性までが汚染されることはない．それはちょうど，濁った池のなかに根を沈め，泥で自らを養いながら，そこから咲きい出る白や赤の蓮華の花が，決して泥に汚されないのと同じである．垢は美しいものにかぶさる仮染めの塵にみたてることができ，その本来の清浄を観ぬくときには，無上正等覚(さとり) を証することであろう．

かくて，すべてのものの真実を観ぬくことがまったく自由な観自在菩薩の姿をとられる大日如来は，この「ものすべて浄らかなり」という真理の意味を，より一層に明らかにしようと考えて，顔を和らげ，微笑まれ，左手に持つ蓮華を，右手で開花させるような，開敷蓮華の姿勢 (泥なかにあって清浄を保つ蓮華の法門) を示めされた．そして，諸々の欲望がものの本性まで汚染できないことを観察し，すべての生き物が汚されることがないという真理を一字で表わす聖音「フリーヒ」を唱えたのであった．

第5段　すべてに富めり (富の法門)
虚空蔵菩薩の章

ついで，大日如来は，三界のすべてのものの主であり，そのすべての福を集める一切三界主如来の姿となられ，すべての如来がすべての人々に灌頂 (めぐみ) する真実なる智慧の理趣 (みち) を説かれた．これは灌頂の智慧をみきわめる「四種の施行」である．

まず，自ら智慧の水をすべてのものに灌ぐものとなり，こ

れによって真理を悟って成仏する．最上の宝を他人にも自分にも施し，三界のすべての心に願うところとなり，三界の法王の境地に到る (灌頂施).

また，あらゆる人々に義利 (よきもの) を施し，世の生活に不如意なものを取り除く．そうすれば，この世の一切の願いは満ち足らされるであろう (義利施).

次に，如来の法 (おしえ) を人々に施すことにより，一切のものが法性 (普遍の真理) を獲得することであろう (法施).

最後に，生 (いのち) の資 (もと) となる種々さまざまなものを施す．この行ないによって，世のすべての飢えたるものは，ことごとく苦しみから救われ，身口意 (しんくい) も安らかで楽しいものとなるであろう (資生施).

かくて，一切の如来の灌頂智蔵という真理の体現者であり，虚空 (おおぞら) のように無限の福徳を身に持つところの虚空蔵菩薩は，この「四種の施行」の真理を，より一層に明らかにしようと考えて，顔を和らげ，微笑まれ，金剛と宝珠とを連ねて作った冠を頭に頂いて，「すべてのものに智慧の水を灌ぐ」という教えを一字で表わす聖音「トゥラーン」を唱えたのであった．

第6段　真実なる活動 (実働の法門)
金剛拳菩薩の章

次に，大日如来は，一切の如来の活動が完全無欠であることを見まもる智慧を備えた得一切如来智印如来の姿となられ，すべての如来の身口意 (しんくい) の活動が，そのまま私たちのうえに加持される (およぶ) ことを知る真実なる智慧の理趣(みち) を説かれた．これは私たちに真実なる働きをもたらす「四種の智印」の教えである．

まず，あなたたちの小さな我に執らわれることを止め，何ものにも執着しない自由な活動を行なうようにすれば，あなたたちの活動が，そのまま一切の如来の身 (活動) と同じものになり，あなたたちの身体のうえに，諸々の如来の活動 (宇宙の活動) を実現することになろう (身印).

また，真実なる智慧の教えを聞いて，邪悪な見解を排斥すれば，一切の如来の語 (言葉) が，あなたたちの言葉と同じになり，如来の法 (宇宙の音声) を体現することになろう (語印).

さらに，真実なる智慧の真義を悟り，一切の如来の智慧の働きを知ったならば，あなたたちの身体のままで，諸々の如来の心の活動 (宇宙の真理) を実現できる三摩地 (境地) に到達することになろう (心印).

最後に，これらの活動のすべてが，如来の心のままに働くことになれば，あなたたちの業 (おこない) が，一切の如来の金剛のごとく不動な業と等しくなり，諸々の如来の最も勝れた完成の境地 (宇宙そのもの) に到達することになろう (金剛拳印).

この「身口意の全活動の完全性」の法門を説き終わられて，大日如来は金剛手菩薩に呼びかけられた．

金剛手菩薩よ．もしこの三つの活動の完全性という真実なる智慧の理趣を聞いて，これを読み誦え，身に保ち持っていたならば，一切の自由を獲得することであろう．また，一切の智慧の自由を得て，自分の知能活動が，如来とまったく同等になるであろう．さらに，小さな我を離れるときに，あなたたちの身口意の活動は，如来の身口意の活動と等しくなり，すべての行動の完成成就を得ることができるであろう．すなわち，身と口と意とのすべての活動が，金剛のごとく堅固で不動なる完成の境地に到達するのである．そうして，あなたたちは，ただこの身のままで，無上正等覚 (さとり) を得るのであろう (即身成仏).

かくて，大日如来は，この真実なる活動の理趣である「四種の智印」を教えるために，金剛拳菩薩の境地に立たれ，この「四種の智印」の教えを，より一層に明らかにしようと考えて，顔を和らげ，微笑まれ，この自由の境地を表わす金剛拳の印を結び，金剛のごとく堅固で不動なる全活動の完成を一字で表わす聖音「アハ」を唱えたのであった．

第7段　言葉に相なし (字輪の法門)
文珠師利菩薩の章

続いて，大日如来は，すべての分別と戯れの議論を離れた智慧の完成の境地に立たれ，この境地を象徴する一切無戯論如来になられて，アの音から始まる一切の語音を自由自在に操って，言語活動のうえに現われた「言葉に相 (すがた) なし」という真実なる智慧の理趣 (みち) を説かれた．これは「四種の解脱」といわれる教えで，文珠師利菩薩の智慧である．

すべての諸法は，その本質において，いかなる実体をも持たないのであるから，空である．宇宙に存在するあらゆる事物は，もろもろの事物が縁起によって成り立ったもので，固定的実体がないのである．これを諸法の空と呼ぶ．

すべての諸法は，その本質において，定まった形相がないのであるから，無相である．だから前述の諸法の空というのは，表現上の仮の相 (すがた) なのだから，もし諸法の空に固執したならば，逆に空から遠ざかってしまう．これを諸法の無相と呼ぶ．

すべての諸法は，その本質において，特定の方向や目的を持たないのであるから，無願である．だから前述の諸法の無相について，その相から離れたいという願いに執着することは，逆に空から遠ざかってしまう．これを諸法の無願と呼ぶ．

このように，すべての諸法は本質において空であり，固定した相も，目的も方向も願いもないのであるから，すべての諸法は自由で光り輝いている．真実なる智慧の理趣に

照らしてみれば，ものはみな清浄なのである．これを諸法の光明と呼ぶ．

かくて，清浄で邪気のない文珠師利菩薩の姿に変身した大日如来は，この「ものも言葉もすがたなし」という教えを，より一層に明らかにしようと考えて，顔を和らげ，微笑まれ，自らの智慧を象徴する鋭い剣を振るって，一切の如来を切る姿勢をとられ，この教えを一字で表わす聖音「アム」を唱えたのであった．

第8段　輪のごとく欠くるところなし (入大輪の法門)
纔発心転法輪菩薩の章

ついで，大日如来は，世の一切の如来の悟りを示す輪のごとく円満なる境地に立たれ，この境地をそのまま名とする一切如来入大輪如来 (人々のために慈悲の心を起こして真実なる法輪を回転させる纔発心転法輪菩薩の如来) になられ，完全にして円満なること大いなる輪のごとき境地に入るための真実の智慧の理趣 (みち) を説かれた．これは「四種の輪円」と呼ばれる教えで，宇宙の秩序を表現する曼荼羅 (シェーマ) の一つである．

まず，あなたたちが，自分自身の心も，世のすべてのものも，堅固不動にして，連なる輪のごとく完全無欠であることを悟れば，たちどころに一切の如来の環のごとく完全なる智慧の境地に到達することができる (如来の法輪)．

次に，世の義利 (利益) の偏在はありえず，それはすべてに平等円満であり，あたかも虚空 (空間) のごとく尽きることなく，宝を取り出すことができるという虚空蔵菩薩のような円満なる境地に入ることとなろう (大菩薩輪)．

さらに，すべてのものの円満で平等なることを悟れば，たちどころに大乗の妙なる完全なる真理の境地に入ることであろう (妙法輪)．

最後に，すべての事業の円満で平等なることを悟れば，たちどころに一切のものの働きが円満で安全なる境地に到達するであろう (事業輪)．

かくて，この「輪のごとく欠くるところなし」という真実の智慧の理趣を説かれた大日如来は，纔発心転法輪菩薩の姿に変身されて，より一層にこの教えを明らかにしようと考えて，顔を和らげ，微笑まれ，この教えを象徴する金剛輪を右手の中指に乗せて，左手で一つ股の金剛杵 (独股杵) を持って，この杵で右の輪をクルクルと回転させて，この輪に喩えられる真理の働くようす (曼荼羅) を示された．そして，この完全無欠な輪のごとき境地を一字で表わす聖音「フーム」を唱えたのであった．

第9段　供養とは何ぞ (供養の法門)
虚空庫菩薩の章

それから，大日如来は，世の一切の如来を種々に供養するための広大無辺なる儀式の蔵という名前を持つ一切如来種種供養蔵広大儀式如来 (虚空を庫にしてすべてのものに惜しみなく円満にものを与えて欠けることがない虚空庫菩薩の如来) となられ，一切の如来を供養する非常に勝れた境地を生み出す真実なる智慧の理趣 (みち) を説かれた．これは「四種の供養」と呼ばれ，すべてのものを利して養う四つのすぐれた行ないの実践である．

まず，菩提 (さとり) を願う心を発すれば，それがそのまま諸々の如来の最も喜ばれるところとなり，広大な供養になる．なぜならば，真理の具現者である一切の如来と菩薩は，真理を知り，悟るものが一人でも多いことを願っているからである (発菩提心供養)．

次に，この世のすべてのものを救済することは，それがそのまま諸々の如来を供養することになる．なぜならば，すべてのものを苦海から救い出して涅槃に度 (わた) らせることが，如来の本願だからである (救度供養)．

さらに，妙なる真理を記した経典を身に受持することが，そのまま諸々の如来への供養になる．なぜならば，経典に書かれた内容は，如来の悟りであり，それを人に知られることを欲するからである (妙経供養)．

このように，般若波羅蜜多 (最高の真理) を記した経典を，身に受持し，読誦して，自分で書き写し，他人にも書き写させ，よく心に考え，十種の修行を実践したならば，それがそのまま諸々の如来の供養になるであろう (般若供養)．

かくて，この「真実の供養」の教えを説き終わった虚空庫菩薩は，より一層にこの教えを明かにしようと考えて，顔を和らげ，微笑まれ，一切の事業が空しからずという境地を一字で表わす聖音「オウム」を唱えたのであった．

第10段　大いなる忿り (忿怒の法門)
摧一切魔菩薩の章

続いて，大日如来は，すべての障害を克服するための悪に対する強い忿怒を内に潜めた智慧の境地に立たれ，その境地を象徴するために，拳を強く握り，その境地をそのまま名とする能調持智拳如来 (外に憤怒を内に悲憂を懐いて一切の悪を摧く摧一切魔菩薩の如来) となられて，「世の一切の悪を調伏する智慧の蔵」という真実なる智慧の理趣 (みち) を説かれた．これは「四種の忿怒」といわれる教えである．

まず，一切の有情(生きとし生けるもの) は，その本質において平等であり，自他の差別は現象のうえだけのものだから，すべて自他の差別によって起こる忿怒も，その平等性の発露そのものであり，対立的な忿怒ではない．これは忿怒の平等と呼ばれる．

また，一切の有情は，現象のうえからみれば，惑と業と苦の三つのあり方のなかにあって，苦しむ存在であり，こ

れは調伏しなければならない存在である．一切の有情は平等なのであるから，忿怒は調伏の働きがある．これは忿怒の調伏と呼ばれる．

世の一切の有情は，その背後に潜む法性 (不変の本質) の現象界における差別と展開に他ならないのであるから，この法性を強く自覚することが，そのまま大いなる忿怒である．これは忿怒の法性と呼ばれる．

世の一切の有情は，その本質においては，いかなる行為も行動も完全で不変な金剛のごときものなのだから，現象の差別を離れ，この本来の金剛性を強く自覚すれば，我執にとらわれ束縛された行動を調伏できる．なぜならば，すべての生きとし生けるものを調伏することは，さとりのためだからである．これは忿怒の金剛と呼ばれる．

ではいかなる理由によって，世の一切の有情を調伏することができ，菩提を開くことができるのであろうか．それは，現象の差別を撤廃させることによって，その背後の清浄な菩提を見成するときに，大いなる忿りを離れるのである．

かくて，摧一切魔菩薩は，この「大いなる忿り法門」を，より一層に明らかにしようと考えて，顔を和らげ，微笑まれ，手に金剛牙の印を結び，身体全体を金剛夜叉の姿に変身させて，すべてのものを恐怖させ，仏道に引き寄せようとされた．大いなる忿怒は，そのままで大いなる歓喜となり，この教えを一字で表わす聖音「ハハ」を唱えたのであった．

以上，大日如来は，第 3 段から第 10 段までの 8 章において，第一段で示された大日如来の心の理念と，第 2 段で示された大日如来の心の発現とが，八大菩薩によって表現されるさまざまな徳目を通して，いかに我々の現象世界に実際に現われるのかを説き終られた．かくて，大日如来は，ふたたび八つの徳目を一つに普集 (まとめ) て説かれる．

第 11 段　平等にして差別なし (普集の法門)
普賢菩薩の章

ついで，大日如来は，はかり知れぬものの動きと，その根底にある平等な静けさを表わす境地に立たれ，その境地をそのまま名とする一切平等建立如来 (すべてのものの心のなかの理想である如来蔵を人格化した普賢菩薩の如来) となられ，実在の世界にあっても，現象の世界にあっても，すべてのものが最も勝れた形で現われるという真実の智慧の理趣 (みち) を説かれた．これは普賢菩薩の徳として表現される「四種の出生」である．

まず，すべてのものは，その本質において平等であり，現象界で見られる区別は仮のものに過ぎない．したがって，ものの本質を観る根本の理性 (真実なる智慧の理趣) は，万物の平等性に他ならない (阿しゅく如来：第 3 段と第 7 段の統合)．

また，自他の執着を離れて，すべてのものの平等を自覚すれば，この世の一切の義と利は，自他の追及の対象ではなく，あらゆるものの義と利になるからである．したがって，ものの本質を観る根本の理性 (真実なる智慧の理趣) は，万物に普遍している共通の利益に他ならない (宝生如来：第五段と第九段の統合)．

次に，すべてのものは，共通する清浄なる法性があるのであるから，ものの本質を観る根本の理性 (真実なる智慧の理趣) は，万物に普遍している共通の法性それ自体に他ならない (阿弥陀如来：第 4 段と第 8 段の統合)．

最後に，すべてのものの事業 (三業：働き) は，実在界の本質に根ざす事業そのものなのだから，ものの本質を観る根本の理性(真実なる智慧の理趣) は，万物の事業それ自体に他ならない (不空成就如来：第 6 段と第 10 段の統合)．

かくて，この「平等にして差別なし」という実在界および現象界の普遍的原理を説き終わり，普賢菩薩と等しい金剛手菩薩の姿に変身した大日如来は，一切の如来と菩薩が相互に関連して集会した三摩耶 (実在と現象の差別がなくなった曼荼羅) に入られて，「世の一切は空しからず」という境地を一字で表わす聖音「フーム」を唱えたのであった．

第 12 段　有情みな加持す (有情加持の法門)
外金剛部の諸天の章

ここで，大日如来は自らの姿に戻られて，「世の一切の有情 (生きとし生けるもの) は互いに加持し合う」という真実なる智慧の理趣 (みち) を説かれた．これは現象界にある諸々の低い次元の神々 (外金剛部の諸天) に力を与えて悟りを獲得させるための「四種の蔵性」と呼ばれる．

まず，一切の有情は，その本質において，如来となりうる清浄な可能性を内に蔵している．これは如来蔵と呼ばれる．なぜならば，大日如来のあまねくゆきわたる光明は，一人一人を離れて，全一なる一切我として，広くあちらこちらにゆきわたっているからである．これは普賢菩薩の徳として人格的に表現されている (大円鏡智)．

また，一切の有情は，その本質において，平等の堅固にして不変なる智を蔵すること金剛のようである．なぜならば，虚空蔵菩薩の智慧の水によって灌ぎ浄められているからである (平等性智)．

さらに，一切の有情は，その本質において，妙なる法を内に蔵する．なぜならば，すべてのものは，この正しき理を覚えるとき，自己と他者のために，誰でもが理解できる言語を活用して，その真理を再現して伝達できる．これは観自在菩薩のごとくである (妙観察智)．

そして，一切の有情は，その本質において，羯磨 (はたらき) が本来清浄にして純粋なる活動を内に蔵する．なぜな

らば, すべてのものの働きは, 宇宙本然の活動の発現であり, あなたたちの身口意の三密の活動が, そのまま如来の三密の活動となるからである. これは金剛拳菩薩の徳目である (成所作智).

かくて, 如来や菩薩のはるか外にある仏法の守護神である外金剛部の諸天も, 自己の本質において, 宇宙そのものである大日如来の化身であることを悟り, 大日如来自身の境地に入り, 歓喜の声をあげ, より一層に「有情みな加持す」という真理を明かにしようと考えて,「金剛のごとく不壊にして自由なる真理」を一字で表わす聖音「トウリー」を唱えたのであった.

この段において, 現象界にあるものすべてに, その本質が清浄な仏性の存在であることが説かれた. この現象界に普遍な実在の理 (俗諦の総説) を説かれ, 以下, 13 ～ 15 の各段において, これらの真理を享受する天部の神々が, その真理を称賛する.

第 13 段　この理趣をたたえて (諸母天の法門)
七天母の章

まず, 天母 (めがみ) である七人(炎魔天母, 童子天母, 毘しゅぬ天母, 倶吠羅天母, 帝釈天母, 暴悪天母, 梵天母) が, この教えを聞いて, 大いなる歓びに充ちて, 大日如来に礼拝し, 七人の天母たちの願いである四摂を述べた. すなわち, 四摂とは,「未だに仏道に近づかないものを近づけること (徴召)」,「近づいたものを仏道に導き入れること (摂入)」,「仏道を破壊したり有情 (生きとし生けるもの) を殺すような不善な心を滅ぼすこと (能殺)」,「仏道に住した修行者を大悟させること (能成)」であり, これらの四つの願いを異口同音に述べ, これらの願いを一字で表わす聖音「ビヨー」を唱えたのであった.

第 14 段　ふたたび理趣をたたえて (三兄弟の法門)
三高神の章

ついで,「力はすべてにあまねし」という有情加持の法門を聞いて, 歓喜の心を抑えることができない三人の高神たち (梵天：ブラフマン, 大自在天：シヴァ, 那羅延天：ヴィシュヌ) が, 大日如来に礼拝し, 自らの心に生じたさとりを一字で表わす聖音「スヴァー」を唱えたのであった.

第 15 段　みたび理趣をたたえて (四姉妹の法門)
四天女の章

ついで,「力はすべてにあまねし」という有情加持 (生きとし生けるものの加護) の法門を聞き終って, この法門の四つの部門を, それぞれ受けた四人の天女たち (適悦：ラテイ, 死天妃：マラニー, 猪美妃：ヴアーラヒー, 成就美妃：シッデイカーシー) は, 自らの心中のさとりを一字で表わす聖音「ハム」を唱えたのであった.

第 16 段　究極の真理とは何ぞ (各具の法門)
四波羅蜜の大曼荼羅の章

ここで, 大日如来は, 限りなくはかり知れない究極の真理を表わす境地に立たれ, その境地をそのまま名とする無量無辺究竟如来となられて, ここまで説いてこられた教えが, 平等で円満で金剛のごとく不変な真実なる智慧の理趣 (みち) であることを重ねて明かし説かれた.

まず, 究極の真理(般若波羅蜜多) は量に限りがない. したがって, それは一切の如来に及んで欠けるところがない (無量).

また, 究極の真理(般若波羅蜜多) は辺に限りがない. したがって, それは一切の如来に及んで例外がない (無辺).

次に, すべてのものの構成要素は清浄で同一な性に基づいているのだから, 究極の真理(般若波羅蜜多) は一性である (一性).

また, すべてのものは, それ自体が絶対究極なのであるから, 究極の真理 (般若波羅蜜多) は究竟である (究竟).

かくて, この「究極の真理 (般若波羅蜜多)」を説き明かされた大日如来は, その教えを聞き終えられた金剛手菩薩に呼びかけられた.

金剛手菩薩よ. 究極の真理 (般若波羅蜜多) を聞いて, 身に受持し, 読誦し, 心に思い考えるならば, そのものは, この世における自分と他人のためのすべての行為が, そのまま自他の区別を超越して, 如来や菩薩の行為と等しくなるであろう.

第 17 段　真実なる智慧の理趣 (深秘の法門)
五種秘密三摩地の章

かくて, 大日如来は, 宇宙の秘密の法性に得て, 戯れの議論と分別を超えた境地に立たれ, その境地をそのまま名とする得一切秘密法性無戯論如来となられ, 比べるものが何もない無始無終の法門を説かれた. これは「大いなる楽は金剛のごとく不変で空しからずして真実なり」という真理 (この経の題) を知る真実なる智慧の理趣 (みち) である.

菩薩は小さい我を離れ, 自らの欲望そのものを, 普遍的な大きい我とするがゆえに, いついかなるところでも, 大いなる楽を得る最勝の境地に到達する.

菩薩は最勝の大楽の境地に到達するがゆえに, 一切の如来の最勝の大いなる菩提 (さとり) を完成成就することができる.

菩薩は一切の如来の最勝の大いなる菩提を完成するがゆえに, 強力な悪を摧く一切の如来の最も勝れた徳を得ることができる.

菩薩は強力な悪を摧く一切の如来の最も勝れた徳を持っているがゆえに, 三界のいかなるところにあっても自由な

境地に到達できる.

菩薩は三界のいかなるところにあっても自由な境地に到達するがゆえに, 世のすべての生きとし生けるものが生死にさ迷うを止める.

このように, 菩薩は大きなる精進の心をもって, 世のすべてを救い, 利益し (たすけて), 安楽ならしめ, そのすべてを最勝で絶対で究極の境地 (涅槃) へ度 (わた) らしめるのである.

それはなぜかを偈 (うた) によって示そう.

菩薩は勝れし智慧を持ち,
なべて生死の尽きるまで,
つねに衆生の利をはかり,
たえて涅槃におもむかず.

世にあるものもその性も,
智慧の及ばぬものはなし.
ものの有姿もその本性も,
一切のものはみな清浄し.

欲がこの世をととのえて,
よく浄らかになすゆえに,
すぐれしものもまた悪も,
みな残らずにうちなびく.

蓮華が泥から咲きいでて,
花はよごれによごされず,
すべての欲もまたおなじ.
そのままにして人を利す.

大いなる欲はきよらかで,
大いなる楽に富みさかう.
三界の自由を身につけて,
堅くゆるがぬ利を得たり.

金剛手菩薩よ. すべてのものの本初である「真実なる智慧の理趣 (みち)」を聞いて, 日々の朝に読み誦え, あるいは聞くだけで, 一切の安楽と快適を得ることができ,「大いなる楽は金剛のごとく不変で空しからずして真実なり」という境地に到達できるであろう. この世のこの身のままで, すべてのものの自由と快適を得て, 十六の大菩薩が表わす十六の生の段階 (禅定) を通って, 如来の金剛のごとく究極普遍の境地に達することができるのである. これを一字で表わす聖音で唱えよう「フーム」.

おわりに (流通分)

かくて, 一切の如来と持金剛菩薩たちは, みなこの座に来て集まり, この教えを空しくせずに, 滞らせずに, 速やかに完成させようと欲して, みな共に法を説いた大日如来の智慧の姿である金剛手菩薩 (＝普賢菩薩＝金剛さった) を称賛していう.

よいかなよいかな大いなる行者,
よいかなよいかな大きな楽,
よいかなよいかな大乗の法,
よいかなよいかな大いなる智慧.
よくこの教えを説くときは,
理趣の経に金剛の力あり.
この勝れた経を身に持てば,
すべての悪魔に害されない.
如来と菩薩の境地を得て,
あらゆる悟りを成就する.
すべての如来と菩薩たちは,
共に教えを説き終わり,
聞く者すべて成就させようと,
心歓ばぬものはなし.

ここに般若理趣経を終わる.

哀しきスーパーハッカー

青山 幹雄
(新潟工科大学)

かれの名前はK.M, 31才．3才時に両親が離婚．父を知らない．ロスアンジェルスでウェイトレスをしていた母の手で育てられた．孤独がそうさせたのか, 16才になるまでに, コンピュータネットワークに取り憑かれていた．このときに, 初めてハッキングの快感を味わった．通っていた高校のコンピュータシステムに侵入したのだ．かれはただデータを見ただけだった．自分の成績を改ざんするような真似はしなかった．

サイバースペースの迷える子羊．だれかに知ってもらいたい．見出してもらいたい．それが, かれのひそかな願望だったのかもしれない．1億円のデータを盗んだ容疑でノースカロライナ州ローリーの自宅で逮捕されるまで, かれは世界で最も警察が探し求めていたハッカーだった．

ハッカーとして最初に逮捕されたのは10代の時であった．電話会社パシィフィク･ベルのロスアンジェルスにあるコンピュータ･センタからマニュアルを盗んだ容疑だった．しかし, 判決は執行猶予になった．かれは, すぐに, 近くの大学のコンピュータ･センタに侵入した．その結果, 6ヶ月の実刑処分となり, ムショ暮らしを経験した．

刑務所から出るまでにAT&Tの電話ネットワークに精通するようになった．勝手に, 課金されない電話番号を作ったり廃止できた．かれは, ハッカー仲間からは「コンドル」と呼ばれていたが, AT&Tはジェームズ･ボンドと呼んでいた．かれが使っていた電話番号の最後の3桁がいつも007だったからだ．

かれは, なかなかユーモアのセンスもあった．ドライブスルーのファーストフードの店では, 窓口から無線を使って注文を取っていることに気づいた．近くに停めた車の中から, 客が注文する声を無線で傍受した．かれは, 無線をスピーカーシステムの周波数に合わせて, 窓口のふりをして客をからかったりした．

だが, かれには, どこか冷酷なところがあった．DECのコンピュータ･プログラムを盗んだ容疑でFBIがかれを逮捕したときに, かれを逮捕に追いやったのは, かれの友人だった．

「どうしてこんなことをしたのだ？」

「あなたは, 社会にとって危険なのよ」

かれは, ふたたび, ムショで1年を過ごし, その後は, 保護観察処分になった．この処分の間, コンピュータ中毒を矯正するための12段階のプログラムが課せられた．1990年に出獄したときは, まず, コンピュータやモデムに一切触れることを禁じられた．その後, 1992年まではある私立調査機関に勤め, 調査などの仕事をしていた．

だが, かれの兄弟がヘロイン中毒で死んだことが, 眠っていたかれの牙を目覚めさせた．ある日, パシフィック･ベルのセキュリティ担当者のボイスメールの盗聴容疑で, FBIの捜査官が家宅捜査令状を持って訪れたときに, かれは再び逃亡者となった．

この3年間の逃亡の間, かれは, IRC(Internet Relay Chat)を使って仲間と連絡を取っていた．かれは, 常に, 自分の存在を知らせたがっているようだった．

そして, ついに, 1994年のクリスマスに, コンピュータ･セキュリティの第一人者である, カリフォルニア大学サンディエゴ校の下村努氏のコンピュータに侵入し, かれのセキュリティプログラムを盗んだ．しかし, このとき, かれは無意識に逮捕を望んでいたのかもしれない．

下村氏は, 必死になってかれを探した．そして, 下村氏とFBIがかれのノースカロライナの自宅に踏み込んだとき, かれはコンピュータの前に座って, 何かを探しているようであった．

「やあ, ツトム．君の腕前には敬意を表するよ！」

(Time, Feb. 27, 1995　他より)

Call for Participation

IWSED-95

International Workshop on Software Engineering Data

Utilization of Defect and Cost Data in Process and Risk Management

October 9-11, 1995

University of Maryland, College Park, Maryland, U.S.A.

The theme of the IWSED-95 is the Utilization of Defect and Cost Data in Process and Risk Management. The past workshops have emphasized the identification and use of software engineering data for quality control and project management. As opposed to using software engineering data for the management of individual projects, the IWSED-95 focuses on how product and process data can be used to improve the process itself, i.e., the way future projects will be carried out.

The main goal of the workshop is to involve all participants in discussions and active interchange of experiences in using defect, cost and risk data to improve the software process. Specific topics to be addressed are the following:

- How the data about the process and products can be used to identify and initiate required changes in the process.
- What is the impact of process variance on the reliability of data collected from individual projects and how to deal with such variance.
- How to use defect and cost data in process risk management.
- How to compare data between different organizations and whether maturity models can be used to help in this task.

Preparation for the Workshop

In order to facilitate in-depth discussions during the workshop, all participants will be required to fill out a questionnaire in advance about their software engineering data. This questionnaire includes questions about the participant's software development organization, its objectives for data collection, what kinds of data have been collected, and how data is used. All information collected through questionnaires will remain confidential. However, based on the returned questionnaires, some participants may be contacted to inquire whether they are willing to give presentations during the workshop.

This preparatory work will allow all participants to relate better to each others' situations and data during the workshop.

The first day of the workshop is dedicated to presentations by participants. We expect to have a two presentations on each topic area. The topic areas will be selected based on the questionnaire feedback but we expect them to similar to the areas listed below:

- setting up a measurement program
- cost data
- risk management
- process definitions and measurement
- defect data
- experiment design

The second day of the workshop is spent on working groups that address the issues presented and discussed during the first day. The results of this work will be presented in the morning of the third day. The workshop concludes in a tutorial on Quality Improvement Paradigm and the Experience Factory. This tutorial is shared by the participants of the meeting for the International Software Engineering Research Network (ISERN).

Registration

The workshop attendance is limited to 40 participants. Anyone interested in participating should contact the workshop organizers to obtain a copy of the workshop questionnaire and further details about the workshop.

The cost of participation to the workshop is $300 per participant, which includes workshop materials and meals, including two dinners.

Important Deadlines for the workshop are as follows:

August 11, 1995	Deadline for returning the IWSED-95 participant questionnaire.
August 31, 1995	Deadline for workshop registration.
	Registration after this date may be accepted, subject to availability.
August 31, 1995	Potential presenters of experience reports will be contacted.
September 11, 1995	Final program will be released.

Location

The workshop will be held at the Inn and Conference Center at the University of Maryland, College Park, Maryland. Accommodation is available at conference site and at a nearby hotel at the rate of $62-$69 per person per night. A quota of 20 hotel rooms have been reserved for IWSED-95 participants. Participants are recommended to use the bottom half of the attached registration form to make their hotel reservations at this hotel (Tel: 301-985-7303, Fax: 301-985-7445). An alternative hotel acts as an overflow accommodation site: Holiday Inn - College Park, 10000 Baltimore Blvd, College Park, MD 20740, Tel: 301-345-6700, Fax: 301-441- 4923.

Program Chairs

Victor R. Basili(University of Maryland)
Koji Torii(Nara Institute of Science and Technology)

Program Committee:

H. Guether(Allianz-Lebensversicherung)
Baerbel Hoerger(Daimler-Benz AG)
Tony Jordano(SAIC)
Kari Kansala(Nokia Research Center)
John Marciniak(Kaman Sciences Corporation)
Yasuhiro Mashiko(Matsushita Communication Industrial Co. Ltd.)
Kenichi Matsumoto(Nara Institute of Science and Technology)
Takamasa Nara(Hitachi, Ltd.)
Domenico Natale(SOGEI)
H. Dieter Rombach(University of Kaiserslautern)
Keishi Sakamoto(OMRON Corporation)
Richard Stenglein(Motorola)
Helmut Woda(Robert Bosch Gmbh)
Mary Zajac(AT&T Laboratories)

Contact and Further Information

Further and updated information about IWSED-95 is available from the following sources:

Jyrki Kontio, IWSED-95
Organizing Chair
University of Maryland
Computer Science Department
A.V. Williams Building
College Park, MD 20742,
U.S.A.
Phone: 301-405-2743
Fax: 301-405-6707
Email: jkontio@cs.umd.edu

FTP site for questionnaires and
other information:
ftp.cs.umd.edu
directory: pub/sel/iwsed95

WWW home page:
http://www.cs.umd.edu/events/iwsed95/

日本人のプログラム委員の連絡先は次のとおりです。お気軽にお問い合わせください。

松本 健一
奈良先端科学技術大学院大学
Tel:07437-2-5311
Fax:07437-2-5219
Email:matumoto@tsugumi.ics.es.osaka-u.ac.jp

増子 泰弘
松下通信工業株式会社
Tel:045-544-3386
Fax:045-544-3385
Email:mashiko@isd.mci.mei.co.jp

奈良 隆正
株式会社 日立製作所 情報システム開発本部
Tel:044-549-1111
Fax:044-549-1212
Email:t-nara@system.hitachi.co.jp

坂本 啓司
オムロン株式会社
Tel:0775-65-5048
Fax:0775-65-5579
Email:sakamoto@eftses.krc.omron.co.jp

参加希望者へのアンケート質問票 (Question 数は全部で 64 問) は，上記の ftp site または WWW page から取り出してください．インターネットへのアクセスができない方は，日本人プログラム委員のどなたか，または SEA 事務局まで御連絡ください．

SEA 1994 年～ 1996 年の主なイベント（実績と予定）

1994	9/3	全国縦断 Forum in 沖縄 (オブジェクト指向)	那覇市
	9/5	TTC ネットワークセミナー	沖縄
	9/19	Forum (インターフェイス･エージェント)	東京
	9/21 ～ 24	第 12 回 夏のプログラミング･ワークショップ	盛岡
	9/30	全国縦断 Forum in 横浜 (コンポーネントウェア)	神奈川
	10/18	Forum (障害者と計算機･ネットワーク)	東京
	10/27	全国縦断 Forum in 広島 (情報技術最前線)	広島
	10/27 ～ 29	第 8 回 教育ワークショップ	四国
	11/7 ～ 9	第 2 回 IWSED (ソフトウェア･データに関する国際 WS)	広島
	11/10 ～ 11	全国縦断 Forum in 関西 (プロセス & メトリクス)	京都
	11/16 ～ 19	Kunming International CASE Symposium(KICS'94)	中国･昆明
	12/2	全国縦断 Forum in 北九州 (コンポーネントウェア)	北九州
	12/15 ～ 16	第 15 回 ソフトウェア信頼性シンポジウム	大阪
	12/20	Forum (CSCW)	東京
1995	1/23	Forum (マルチメディア)	東京
	2/8	全国縦断 Forum in 仙台 (コンポーネントウェア)	仙台
	2/20	Forum (インターネット最前線)	東京
	3/22	Seminar & Forum (ドメイン･エンジニアリング)	東京
	4/6 ～ 8	第 7 回 テクニカル･マネジメント･ワークショップ	長崎
	4/14	全国縦断 Forum in 名古屋 (コンポーネントウェア)	名古屋
	4/23	Workshop on Asian Approaches to Software Engineering	米国･ Seatle
	5/25	Seminar & Forum (コンポーネントウェア)	東京
	6/13 ～ 16	ソフトウェア･シンポジウム'95	滋賀
	7/20	全国縦断 Forum in 松本 (コンポーネントウェア)	松本
		**********	
	9/8	SEA Classic セミナー (第 1 回)	東京
	9/28 ～ 29	全国縦断 Forum in 広島 (コンポーネントウェア)	広島
	9/19 ～ 22	第 1 回 ソフトウェア･デザイン･ワークショップ	新潟
	10/9 ～ 11	第 3 回 IWSED (ソフトウェア･データに関する国際 WS)	米国･ Maryland
	10/13	SEA Classic セミナー (第 2 回)	東京
	10/25 ～ 28	Changsha International CASE Symposium(CICS'95)	中国･長沙
	11/10	SEA Classic セミナー (第 3 回)	東京
	11/16 ～ 18	第 9 回 教育ワークショップ	新潟
	12/12	SEA Classic セミナー (第 4 回)	東京
1996	1/12	SEA Classic セミナー (第 5 回)	東京
	2/8	SEA Classic セミナー (第 6 回)	東京
	6/13 ～ 16	ソフトウェア･シンポジウム'96	広島

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