SEAMAIL

Newsletter from Software Engineers Association

Volume 6, Number 3·4·5 1991

目　次

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，1000余名を越えるメンバーを擁するにいたりました．法人賛助会員も約80社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州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．

ＳＥＡＭＡＩＬ　Ｖｏｌ．６，Ｎｏ．３～５　平成３年１２月２０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　サンビルト印刷株式会社　〒１６２　東京都新宿区築地町８番地
定価　１５００円（禁転載）

事務局から

☆

もう年の瀬が迫って来ましたが，わが Seamail はまだやっと３冊目です．

☆☆

毎月平均２回はさしあげている DM からもおわかりのように，イベントだけは次から次へと開催されているのですが，なかなかその成果が紙になってあらわれて来ない．それなのに全然イライラも罪悪感も感じていないらしいこのソフト屋さんたちは，いったいどんな人種なのかしらと，つい考えこんでしまいます．

☆☆☆

２年前の環境ワークショップ in 釧路の報告がやっとまとまりました．プログラム委員長の平尾さんが１人でやきもきして，効果のない督促を繰り返し，夏には一応編集が終わっていたのですが，例によって凝り性の編集長・岸田さんが，国内外の出張続きで忙しいというのに，構成を全面的に組立て直し始め，とうとう秋を越してしまいました．

☆☆☆☆

次号は，昨年夏の「若手の会」（CASE 演習）の特集号．これはほぼ完成間近ですから，年内には仕上がるでしょう（と，希望的観測!?）．

☆☆☆☆☆

急に寒さがやってきました．みなさま，お体を大切に．

☆☆☆☆☆☆

SEA 1991 年の主要イベント（実績と予定: 12/1 現在)"

1/7	SEA Forum「新春放談会」	東京: 事務局会議室（参加者 26 名）
1/30-2/1	第 3 回 ソフトウェア プロセス ワークショップ	伊東: 五景館（参加者 24 名）
2/5	ソフトウェア信頼性フォーラム	仙台: 宮城第一ホテル（参加者 100 名）
2/8	SEA Forum「これからの中堅技術者教育を考える」	東京: 機械振興会館（参加者 41 名）
3/14-15	春のセミナー ウィーク'91	東京: 青年会議所会館（参加者延べ 228 名）
3/25-27	第 3 回 テクニカル マネジメント ワークショップ	直江津: ホテル センチュリー イカヤ（参加者 15 名）
4/12	SEA Forum「テクニカル·マネジメント」	東京: 機械振興会館（参加者 30 名）
5/9-18	ICSE-13	Austin, TX, USA（参加者 10 数名）
5/30	SEA Forum「ソフトウェア開発のノンテクニカルな側面」	東京: 機械振興会館（参加者 30 名）
5/30	SEA 1991 年度総会	東京: 機械振興会館
6/10	ソフトウェア シンポジウム '91 併設チュートリアル	名古屋: 名古屋国際会議場（参加者延べ 168 名）
6/11-12	ソフトウェア シンポジウム '91	名古屋: 名古屋国際会議場（参加者 362 名）
6/13-14	3rd International Symposium on Future Software Environment	彦根: 彦根プリンスホテル（参加者 41 名）
6/24	SEA Forum「オブジェクト指向の実践」	東京: 機械振興会館（参加者 65 名）
7/25	SEA Forum「人間の知的活動支援システムを考える」	東京: 機械振興会館（参加者 40 名）
9/4-7	第 9 回 夏のプログラミング ワークショップ（若手の会）	盛岡（参加者 42 名）
9/20	SEA 特別 Forum in 北海道	札幌: 大同生命ビル会議室（参加者 54 名）
9/24-26	秋のセミナー ウィーク'91	東京: 青年会議所会館（参加者 延べ 134 名）
10/4	SEA 特別 Forum in 名古屋「実践のためのソフトウェア工学」	名古屋: ポートビル（参加申込 50 名）
10/18	SEA Forum「コンピュータ·ネットワークの最新動向」	東京: 機械振興会館（参加者募集中）
10/21-22	1st International Conference on the Software Process	Los Angels, CA. USA（参加者約 130 名）
10/23	SEA Semiar Special「ソフトウェア設計支援環境の未来像を探る」	東京: 青年会議所会館（参加者 16 名）
10/24-26	第 5 回 教育ワークショップ	北陸: 片山津（参加者 21 名）
10/28-30	Beijing Internatinal CASE Symposium '91	中国: 北京大学（参加者 約 70 名）
11/7-8	第 12 回 ソフトウェア信頼性シンポジウム	仙台: 国際センター（参加者 120 名）
11/22	SEA 特別 Forum in 北陸	金沢: 石川県地場産業振興センター（参加者 56 名）
11/27	SEA Forum「ソフトウェア技術の国際潮流を探る」	東京: 青年会議所会館（参加者 22 名）
12/18	SEA Forum「ソフトウェア技術の動向をどう捉えるか？」	東京: 機械振興会館（参加者募集中）
12/24-25	SEA 集中セミナー in 関西'91	大阪: 大阪大学工業会館（参加者募集中）

なお，来年のソフトウェア·シンポジウム'92 は，6 月 9-11 日，長野市での開催が予定されている（現在発表論文募集中）.

第5回 SEA 環境ワークショップ

― 実践的開発環境に関する集中討論 ―

報 告 書

1989年11月29日(水)～12月1日(金)

於： 北海道・釧路市

ソフトウェア技術者協会

はじめに

— Back to the Future: Part-IV —

ワークショップ実行委員長

深瀬弘恭

（アスキー）

1989年11月28日(火曜日),北海道・釧路空港は濃霧のため視界不良で離着陸不可能であり,われわれSEA環境ワークショップ参加者一同を乗せたジェット機は,やむなく千歳に向かった.あいにく,釧路行き特急列車の連絡が悪く,空港無料待合室で3時間の臨時情報交換パーティを開催.それから列車内でも飲み続けて,目的の釧路パシフィック・ホテルに辿り着いたのは,もうかなり夜が更けてからであった.

あれから,すでに2年の時間が経過した.

その間も,ワークステーションやネットワークなど,われわれのソフトウェア開発環境を支える基盤技術は目まぐるしく進歩し続けている.しかし,にもかかわらず,ソフトウェア業界全体を眺めて見ると,大多数のいわゆるSEやプログラマたちの作業環境はそれほど革命的に変化しているようには見えない.

ここに,ようやくまとめられた2年前のワークショップ・レポートの内容が,それほど新鮮さを失っていない(いや今日でもまだ十分にタイムリーである)というのは,考えようによっては悲しいことである.

畏友・熊谷章氏の愛読するイリヤ・プリゴジン博士の名著「混沌からの秩序」(みすず書房・刊)によれば,時間の流れはエントロピー理論によって不可逆であり,かつてH.G.ウェルズの夢見たタイム・マシンの実現は理論上不可能なことではあるが,ここにまとめられたワークショップの討論記録は,一種の仮想タイム・マシンとして,われわれの体感する物理的な時間と社会・心理的な時間との恐るべきギャップを計測する手段を提供してくれるものだといえよう.

それは,ソフトウェア工学の迷路に迷いこんだ沢山のマイケル君たちを日常生活の悪夢から救い出す程度の働きはもっていよう.しかし,私自身の私的な感想を述べさせてもらえば,あのワークショップで提案し,みごとに不発に終わったBOFのテーマ「企業間メトロポリタン・ネットワーク」の構想を実現するためには,しかし,もっと強力な装置を考案して,はるかな未来へのタイム・ワープを試みなければなるまい.

ともあれ,このレポートの編集に大変苦労された平尾一浩・岡本隆一・岸田孝一3氏の御努力に深く感謝する.

第5回 SEA環境ワークショップ
－ 実践的開発環境に関する集中討論 －
プログラム

1989年11月29日（水）

13:00 － 14:00　受付

14:00 － 17:00　**Session 1: MMLの確立を目指して**
- Chair: 中野 秀男（大阪大学）
- Speaker: 久保 宏志（富士通）
- 林 香（SRA）
- 三上 理（日本電気）

17:00 － 18:00　ホテルへのチェックイン

18:00 － 21:00　レセプション @ 釧路 Fisherman's Wharf

1989年11月30日（木）

09:30 － 12:30　**Session 2: HyperTextの光と影**
- Chair: 熊谷 章（PFU & 富士通）
- Speaker: 加藤 康人（PFU）
- 佐原 伸（SRA）
- 篠田 陽一（東京工大）
- 高橋 晃（釧路高専）
- 高橋 肇（日本電子計算）
- 新田 稔（SRA）
- 野村 行憲（岩手電子計算センター）

14:00 － 17:00　**Session 3: ソフトウェア工学は現場に生かせるか？**
- Chair: 岸田 孝一（SRA）
- Speaker: 落水 浩一郎（静岡大学）
- 佐原 伸（SRA）
- 新田 稔（SRA）
- 野村 敏次（日本電子計算）
- 野村 行憲（岩手電子計算センター）
- 濱田 勉（NTT九州）
- 平尾 一浩（ヒラタソフトウェアテクノロジー）

18:00 － 21:00　**BOF Session: 企業間ネットワークの構築と活用**
- Chair: 深瀬 弘恭（アスキー）
- Speaker: 参加者全員

1989年12月1日（金）

09:30 － 12:30　**Session 4: ツール・インテグレーションの理想と現実**
- Chair: 岡本 隆一（ケーシーエス）
- Speaker: 青木 淳（富士ゼロックス情報システム）
- 魚田 昌孝（ケーシーエス）
- 酒匂 寛（SRA）

第5回 SEA 環境ワークショップ
― 実践的開発環境に関する集中討論 ―
参加者名簿

実行委員長	深瀬 弘恭	アスキー
プログラム委員長	篠田 陽一	東京工大（現在は北陸先端大学院）
同上	平尾 一浩	ヒラタソフトウェアテクノロジー
	青木 淳	富士ゼロックス情報システム
	魚田 昌孝	ケーシーエス
	岡本 隆一	ケーシーエス
	落水 浩一郎	静岡大学
	加藤 康人	PFU
	岸田 孝一	SRA
	久保 宏志	富士通（現在はファコム・ハイタック）
	熊谷 章	PFU & 富士通
	酒匂 寛	SRA
	佐原 伸	SRA
	高橋 晃	釧路高専
	高橋 肇	日本電子計算
	中野 秀男	大阪大学
	新田 稔	SRA
	野村 敏次	日本電子計算
	野村 行憲	岩手電子計算センター
	濱田 勉	NTT 九州（現在は NTT ヒューマンインタフェイス研究所）
	林 香	SRA
	三上 理	日本電気
Paper 参加	渡邊 雄一	電力計算センター（現在はアスキー）
事務局	宮田 郁美	SRA

Session 1

MML の確立を目指して

(1989 年 11 月 29 日 14:00 − 17:00)

Chair 中野 秀男（大阪大学）

Speaker 久保 宏志（富士通）
林 香（SRA）
三上 理（日本電気）

討論の狙い

MML (マイクロ・メインフレーム・リンク) というキーワードは，しだいに一般化してきたワークステーションや LAN を，何とか既存のメインフレームを中心とするシステムの秩序を破壊せずに現場に持ち込もうという，いささか苦し紛れのアイデアを象徴しているように感じられます．概念としては理解できても，実際に MML を具体化しようとすると，あまりにも多くの技術上の難問と，それらを解決するためのたくさんの選択肢があり，とくにメインフレーム・サイドからのアプローチを試みようとした場合には，出口のない迷路に入り込む危険が高いようです．

このセッションでは，いくつかのドメインを想定し，また，初心者からウィザードまでの多彩なユーザ層を考えて，システムの機能やセキュリティ，管理・運用，故障対策などの諸問題に関して，MML 具体化の正しい選択肢を見出すガイドラインを設定したいと考えています．

Session 1: MML の確立を目指して

目次

Seesion 1: MML の確立を目指して (Pre-Workshop Position Paper)

チェアマンとしての事前の意図

中野 秀男（大阪大学）

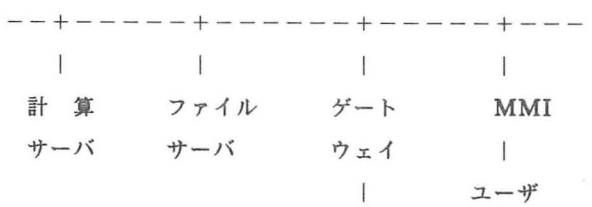
これまで何回かの SEA 環境ワークショップでの WS による分散開発環境をどうするかの議論から，MML の話になり，その「締め括り」を釧路でやろうとしています．チェアマンの立場からポジションペーパを書きます．いま思っている MML セッションの展開です．

議論は，MML のあり方でなく，正しいメインフレームの使い方になるような気がしています．ますます，パワーフルになる WS や PC を考えたとき，ソフトウェア開発環境としての計算機システムはどのようなものがよいかを議論したいと思います．

- 計算サーバ（高速演算）
- ファイル･サーバ（大容量記憶）
- MMI
- ネットワーク

の形で計算機システムを捉えればと思いますが，まず対象とする開発プロダクトの運用機が依然として大型機なのか，分散指向の WS ネットワークなのか，それらの組合せなのかを，分けて考えたいと思います．開発機と運用機が違う場合は，その違いをどのように吸収するかの提案になるでしょう．

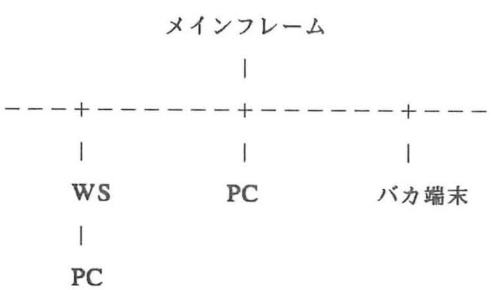
ネットワークの形態はともかく，下図のような構成を考えます．１台が複数の機能を持ってもよいし，同じサーバが複数台あっても目的にあわせて選べるものとします．



まずは運用･管理ですが，方針としては，人間が運用できることはほとんど計算機のプロセスで行えるから，そのような仕組みを考えたいと思います．具体的には，自動バックアップやディスクのあふれ防止等です．大型計算機サイドからは，もっと運用上のノウハウが出て来るものと期待しています．

１つのドメインでは NFS のようなファイル･システムの相互マウントが掛かって，ゲートウェイを通してファイル転送や遠隔ログインができた場合のセキュリティ等についても，議論したいものです．もともと，この環境ワークショップでの当初のねらいはそこだったと思うのですが，そこに至るまでにグチが多く出て，また WS に関係ない人が多くて MML の話題になったと理解しています．

下図のような計算機システムでの開発環境を考えたとき，各タイプのマシンとネットワークに要求されることを提案するセッションにしたいと考えています．チェックリストのようなものを作って，分散環境のレベル分けができればいいのになあ，と思っています．



Seesion 1: MML の確立を目指して (Pre-Workshop Position Paper)

ポジション・ペーパ

久保 宏志（富士通）

かなり前のことですが，ヴァル研究所の島村社長から，ファラオに至る開発の歴史についてお話をうかがったことがあります．いっぺんに島村社長とファラオが好きになりました．島村さんが，パイオニアの情報システム部に所属していたころからずっと，エンドユーザのためのプログラミング環境を追求してきた結果として今日のファラオがあります．はっきりした哲学をもって，いわば究極の「エンドユーザ・プログラミング環境」を追求しておられます．着実にゴールに近付いておられます．そんな印象を受けました．

そんな出会いで島村さんと知合いになり，私がお世話させていただいている日本科学技術連名主催の「ソフトウエア生産における品質管理シンポジウム」の冒頭を飾る特別講演者にお願いするくらいに惚れ込みました．9月 13 日に「あるエンジニア達の開発の軌跡 - エンド・ユーザー・コンピューティングに至るまで -」という題で，お話を聞けることになっています．ここでもう一回勉強できます．

自宅の PC9801 VX21 にはファラオが載っています．まだ使っていませんが近々利用を開始します．ある程度私が使いこなせるようになったら，我が女房にも使わせてエンド・ユーザ・コンピューティング環境としての完成度を，計算機にはずぶの素人の我が女房の物差しではかってもらいます．もちろん私の物差でもはかります．

時期はたぶんファラオを使い始めてからになるとおもいますが，ヴァル研究所のセミナーに出席して，使い方の勉強もするつもりをしています．このセミナーでは，実務にファラオを実用している方からお話がきけると聞いていますので，自分の実感を他人のそれとつき合わせてみることができます．

ここまでの勉強ではまだ MML にはいっていません．マイクロの部分だけです．MML のメインフレームの部分とリンクの部分は，座学と会話と見学とで補うことになります．肝心なのはユーザインタフェースですから，このような方法でも，かなりの部分を実感をもって評価し判断することができるのではないかと，今のところ楽観しています．

ここまでやれば，エンド・ユーザ・コンピューティング環境について具体的に根拠をもって語れるようになるのではないかと，考えています．

こんなところが，MML セション志願のポジション・ステートメントです．

Seesion 1: MML の確立を目指して (Pre-Workshop Position Paper)

機能分散型結合

— ターゲットマシンリンクシステム —

三上 理(日本電気)

1. 機能分散型結合

現在では，ワークステーションは個人の作業環境として注目され，メインフレームは計算や巨大なデータ管理のためのコンピュータとしてその地位を保っている．こういった状況のもとでワークステーションとメインフレームの共存方法を考える場合，これまでのようにメインフレームを中心にした利用機能を提供すると，個人環境としてのワークステーションのよさを生かしきれない場合がある．たとえば，メインフレーム用のソフトウェアを開発する場合，メインフレームを最終的なターゲットマシンとしてとらえ，ソフトウェアをコンパイルし実行するコンピュータとしてのみ利用できればよく，設計，コーディングといった作業はワークステーションで行える方が便利である．すなわち，利用者が両者の間で利用機能を固定，分散させ，その機能をうまく利用できるようにする共存方法が考えられる．

機能を分散させる共存方法の場合，現状ではワークステーションからメインフレームの端末エミュレータやリモートジョブエントリを用いなければならず，利用者は複数の作業環境を持つことになる．

図 1-(1) は，上記のソフトウェア開発を例にしてワークステーションとメインフレームを従来の方法で利用する様子を示したものである．

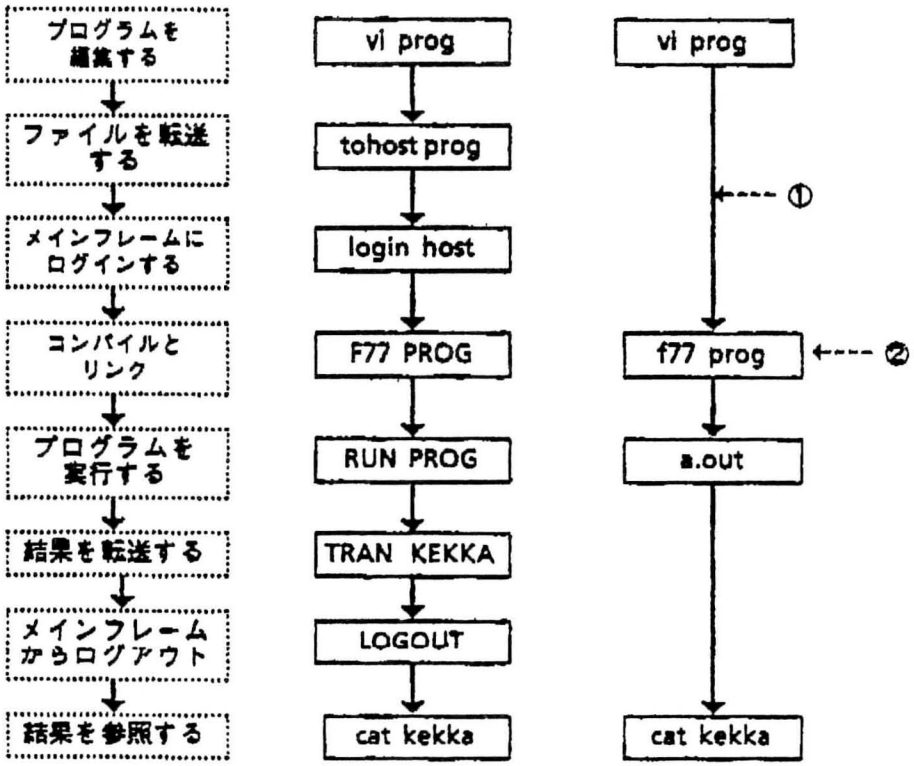


図1 操作方法の違い

この例からもわかるように，

(1) 作業環境の違い
(2) 操作方法の違い
(3) データの移動

といった負担が利用者に強いられる．

これを図 1-(2) のように，複数の環境を意識せずに操作できることが，利用者にとっては望ましい．図 1-(1) の [1] ではファイルの転送が，[2] ではメインフレーム上でのジョブ起動が自動的に行われ，メインフレームはバックグラウンドプロセッサのように動作するわけである．以上のような観点から，筆者らはターゲットマシンリンク (TML) システムというシステムを構築している．システムというシステムを構築している．

2. TML システム

TML システムでは，手もとの unix ワークステーションとネットワークによって接続されたメインフレームとの間で，オペレーティングシステムレベル(コマンドレベル)での統合利用環境を提供するシステムである．図 2 にその利用イメージを示す．図に示すように，TML システムは，Unix オペレーティングシステムの形式のコマンドおよびファイル名をシステム内部でメインフレームの形式に変換する機能を持っている．したがって，利用者はワークステーション形式のコマンドでメインフレームを利用でき，図 1-(2) のような操作が可能になる．

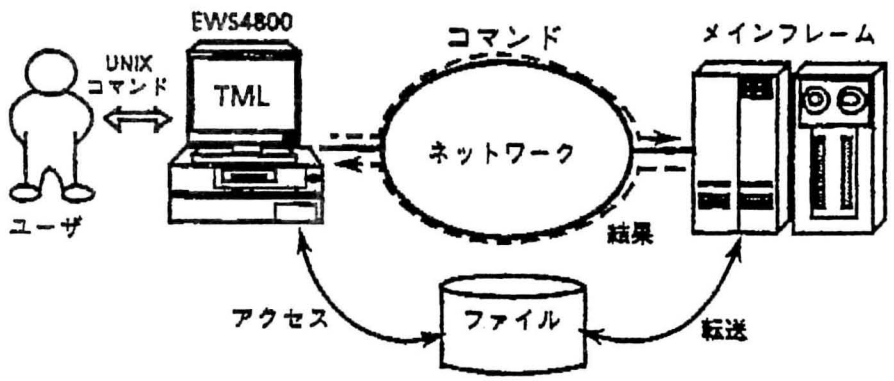


図2　TMLシステム

3. 構成

図 3 が TML システムの構成を図に表したものである．TML システムは，大きく分けて 5 つの部分より成る．

(1) コマンド処理（インタプリタ部）

コマンドの解析，実行コンピュータの判定，変換，実行を行なう．Unix シェルとの互換機能もワポートする．

(2) コマンド変換記述

メインフレームに対するコマンドは，コマンド変換記述によって，メインフレーム形式に変換される．変換できるコマンドは利用する機能によって変化するため，この記述はシステムとは独立にカスタマイズできなければならない．

(3) 仮想共有化ハイル機能

ワークステーションのファイルとメインフレーム上のファイルとの対応付けを行なう．ファイルの名前，ファイルタイプ（テキスト，オブジェクトなど），言語タイプ，Unix ファイルのアクセス権などがその対象である．また，ファイルの転送，キャッシングの機能も持っている．

(4) インタフェイスモジュール

メインフレームとのデータ交換，および仮想共有化ファイルへのインタフェイスである．

(5) 通信機能

メインフレーム用の端末エミュレーションおよびその下位プロトコルとして，TCP/IP，X.25 などが利用できる．また，マルチウィンドウ環境下では，端末エミュレーション画面の表示，入出力の切り換えが可能である．

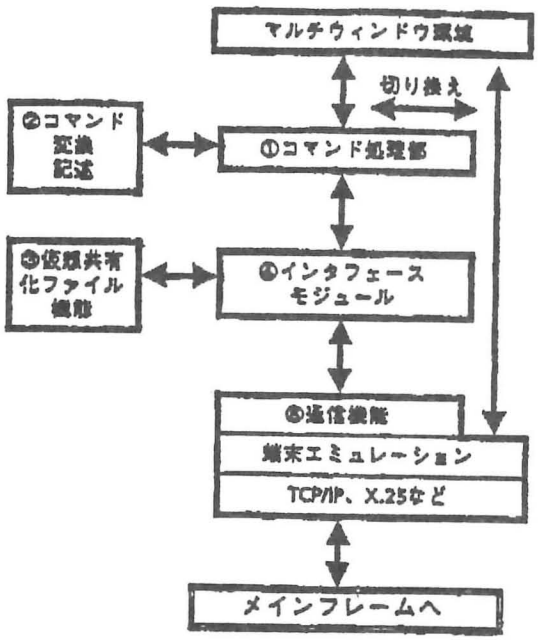


図3 TMLシステムの構成

参考文献

(1) 三上他：マイクロメインフレームリンクによるソフトウェア開発環境の実践方式，34 回情処全大，pp.1171-1172 (1987).

(2) 三上他：複数のオペレーティングシステムとターゲットマシンリンクシステムの開発，オペレーティングシステム研究会資料，No.37-1 (1987).

(2) 沢田他：ターゲットマシンリンクシステムの異ホストコンピュータへの適応性に関する考察，37 回情処全大，pp.283-284 (1988). の開発，オペレーティングシステム研究会資料，No.37-1 (1987).

Seesion 1: MML の確立を目指して (Pre-Workshop Position Paper)

Mac のためにメインフレームを使いたい

林　香（SRA）

1. 現状は？

今のところ MML といえば，メインフレーム・マシンに PC を接続して端末代わりに使っている以外に，目ぼしいものは見当たりません．端末としてのインテリジェンシイやファイル転送の機能を競ったり，接続の方法が同軸ケーブルの直結だ，いや PC の LAN とつなぐことができる，とかいったところで，ワイワイやっている程度でしょう．

変わったところでは，Jstar がメインフレーム・マシンとネットワーク接続できるようですが，これとて，ただのターミナル・エミュレーション以外の何者でもありません．

もちろん，Mac とメインフレーム（IBM しか相手にしていませんが）との接続においても同じ状況です．どうたって PC に置き変わるかという点が，最大の関心事になっています．

2. なぜなのだろう？

なぜ MML は，ターミナル・エミュレーションの枠から抜け出すことができないのでしょう？通信プロトコルがいまいち標準化されていない，それ以前にだいたい後悔されていない，などの技術的な問題もあるのでしょうが，何かもっと根本的なことが忘れられているような気がします．

メインフレームを中心にして問題解決を図ろうとすること自体に問題があるのではないでしょうか？　つまり，メインフレームの上でアプリケーション・ソグトウェアを動かし，マイクロはそのための入出力装置だという前提でシステムを考えるのが間違っているのではないでしょうか？

,ti +1 これではマイクロはいくら成長しても，手足以上の役割を発揮することはできません．ターミナル・エミュレーション以上のことを期待されていないのですから．

3. どうすればよいのか？

よく，Unix や Mac などの上でメインフレーム用のソフト開発を行えばよい，そのためにはどんな環境があればよいか，といった議論がされていますが，あまり有益であるとは思えません．

Unix や Mac の上にメインフレームの疑似環境を作るのは大変な（というより不可能に近い）ことです．特に，方言だらけの COBOL や複雑な DB のシミュレーションなど，考えただけで寒気がして，頭が痛くなってきます．こんなことで苦しむのなら，いっそメインフレームを中心にしたアプリケーション・システムの構築そのものを止めてしまえばよいのです．

メインフレームは，マイクロのバックエンド・プロセッサにしてしまいましょう．そうすれば，今までとはまったく異なった新しい MML の形が見えてくるのではないでしょうか？　今のところ，この希望に微かな光をあててくれそうなのは Mac だけです．

IBM メインフレーム上のアプリケーションのユーザ・インタフェイス部分を Mac に任せようと試みている Mac Workstation や，Mac を軸に表計算ツールや SQL をベースにしてアプリケーションを作り上げ，IBM メインフレームは馬力のある DB マシンとして，バックエンドに押し込めようという CL/I のプロジェクトなどに，期待が持てます．

これらが実現されれば，たとえば Windz や Excel のセルの１つにメインフレーム上の DB アクセス処理を貼りつけるだけで，簡単にアプリケーションを具体化できるようになるでしょう．従来のように，プログラミングに多大な労力を割り当てなくてもすむのです．そうすれば，要求分析や設計にもっと力を入れることができるようになるでしょう．

Seesion 1: MML の確立を目指して (Pre-Workshop Position Paper)

何のための開発環境か？

渡邉雄一（電力計算センター）

1. 開発環境を論じるときの疑問点

現在のソフトウェアの開発のあり方を見る限りでは，個々のアプリケーションの開発対象（分野，内容，機種，動作のための基本的ソフトウェア環境）がほぼ確定して，決まってくる．逆にいえば，個々のアプリケーションに依らない汎用的で包含する機能を持ちつつ，その必要部分を場合場合に応じて使えるような開発環境はまだない．それに一番近い所にあるのが Unix 系のワークステーションなのだろうが....

よく世間では「メインフレーム（大型汎用機）は実行用マシン，Unix は開発用マシン」というが，実際単純にそう決めつけることはできないだろう．問題は，そこにあるソフトウェアの構成だからだ．しかし現実のハードウェア／ソフトウェアを基準として MML (Micro Mainframe Link) を考えたとき，どうしても水と油のようなものを感じてしまう．それは，それぞれに長所短所を抱えているからである．

そのようなことから「Unix だって，いや Unix マシンこそ実行用マシンに適している」という意見もあると思うが，それについては他の方に論じて頂くことが適当だと思われるので，ここではその逆のケース，すなわち『大型汎用機は開発用マシンとしてどの程度不適当か？』ということを考えてみたい．

2. メインフレームに何が期待できるか？

メインフレームといってもさまざまであろうかと思うが，ここでは私の（IBM 互換といわれる富士通製の各種製品の利用）経験を中心にその問題点を幾つか挙げてみたい．

2.1 まず，大型汎用機でのネットワークは大変である．

何が大変か？」といわれれば，（技術的ではなく人間的な問題といわれると苦しいが）「何が大変で何か簡単か？ということが分かっている人がほとんどいない」ということが大変なのだ．

さらに計算機の技術のなかでも，ネットワークに関連したところは「お金との勝負」みたいなところがあって，ちょっとした世の中の技術の趨勢により，実際に得られるネットワークからの御利益は雲泥の差がある．スピードしかり，機能しかり．だが，それを分かって，コストパフォーマンスも含めた機器構成をすべきなのに，それを分かってるものが皆無なため，何処のセンターも泥沼に足を突っ込むこととなっているようだ．

しかしそのような状況の急激に改変させるようなインパクトを与えるような可能性をメインフレームのネットワーク環境が持ちえていないことも事実である．

具体的に，メインフレームの現在のネットワークでは「A 地区のマシン配下の端末から B 地区のマシンに Logon して C 地区のマシンに帳票を出力する」といったことは比較的容易に行える（ただし同じメーカの同じ OS であればの話だが）．このようなシステムで問題なのは，

- どのマシンに Logon しなければいけないか？
  YP のような物は当然ないし，ユーザ ID，パスワードはマシンごとに設定されているから．
- どのマシンで実行すべきか？
  同じジョブクラスでも，マシンによって設定が違う！（逆にジョブクラスがそのマシンに設定されているとは限らない）．またファイルは（JES/MAC や CTC 結合した隣接のマシンでなければ）共用は不可能なので，基本的に個々のマシンで完備していなければならない．
- どのマシンへ出力すべきか？
  どの出力クラスも，マシンによって設定が違う！

といったことを常に意識していなければならないからである（もし，そのような意識をしないで快調にマシンを使えているなら，そのセンターの管理者はかなり偉いと私は思う）．

2.2 運用に小回りがきかない

運用は大変である．真の意味で使いやすい構成を考えることは，膨大な知識と絶ゆまぬ努力が必要である．しかし，そのような努力は多くのユーザから全く見えないところでなされている．これはマシンの大小に限らずどこでも同じであろう．そのためには，当然運用で柔軟にカバーすべき点が多いのである．が，ソフトウェアの開発環境などという概念が形成される以前から存在する基本ソフトウェアであるため，「みんなで使うための機能

として，ユーザよりもマシンのことを第１に考えられている」ため，柔軟性に欠け，ひいては環境を悪化させることとなっている．

現在のメインフレームを見ていると，みんなで使うプログラムをメモリに常駐するための仕掛けなどは，苦労して良く考えられていると思う．そして分かっている人がそれを各センターの実情に合わせて使えば便利である．しかし，それを十分カバーするだけのＣＰＵパワーが今日ある．むしろＯＳの実現のために苦労して作られてきた機能が裏目に出ることが少なくない．特に顕著なのがリエントラントな構造のモジュールである．そのような機能をフルに利用すれば，よりハイスピードなプログラムはできるだろうが，それを姑息（こそく）に使うアプリケーションを少なからず作ってしまった点に，問題が多いと見るのが妥当ではなかろうか？

一方で，そのような欠点に目をつぶりながらも，現状のメインフレームの開発環境を見ると，それなりに良い点も少なからず見受けられる．

(a) ターゲット環境に同等の環境が得られる．

コンパイラやデータベースのアクセスルーチンなどは，絶対に運用環境に相当するものが必要である．

(b) 限られた範囲だが，ドキュンメントツールなどが充実してきている．

コンパイラと連動して，ソースプログラムを解析してドキュメントするツールであるとか，画面フォーマットを出力したりデータ構造図を出力するようなものが提供されてきて，実用上かなり満足できるものである．

これらはいずれも，開発環境と運用環境が同等であることの利点を生かしたものである点が注目される．

3. 大型汎用機ユーザから見て WS/PC に何が期待できるか？

3.1 メインフレームでのソフトウェア資産の欠如を補う豊富なツール群

前項で挙げたメインフレームの利点の裏返しであるが，ターゲットシステム上のコンパイラ，ＤＢ／ＤＣなどのターゲットマシンに備わってるソフトウェアを，ＷＳなどの開発環境上でエミュレートすることが，十分できていないし，多分無理だろう．期待してはいけない！！！

3.2 小回りのきく最新技術－特にプリンタ周りのハード／ソフトに注目！－

個人的には，メインフレーム回りのプログラムの機能切り分けから始めて，総合的なシステムの再構築をしたユーザのみが，21 世紀の情報処理分野を戦っていけだろうと考えている．そのための１手段として，スピードの限界が一番顕著なのがプリンターまわりであろう．とくに現実には，書類の偽造などを防ぐため，2P，3P といった用紙の使用が義務づけられている分野では，切実な問題である．それに対応するためには，１台のパソコンに２０台程度のプリンター接続できる機能が欲しいと思っている．これが実現できれば，現在のオフコンはパソコンにとってかわられるだろう．そして今までオフコンで成されていた機能が，Unix などへ移行していくことは明らか！！

3.3 メタな開発情報の情報管理（作成，更新，検索，転送）の一元化／簡素化

これからの情報処理は総合力である．それを一番望まれているのが，開発環境だろうし，また実際にそれをめざしてさまざまなアプローチが試みられているわけだが，とくにメインフレームではここの具体的な情報はなんとか管理できるものの，総合的にメタな情報を扱えないので....

4. ではどうする？　－運用環境の現実的な改革しかない！－

開発環境を論じることにより，運用環境の変貌を期待する．そのための，情報をフィードバックをするための技術が，コンセプトとしてあらわになっているものを望む．ＭＭＬという観点からは，現実問題としてプログラムの機能の見直しだろうし，それに見合う小回りのきく運用環境の選択だろう．それをなしうる開発環境は，目標が具体的であるがゆえに，技術的には低レベルのもので十分である．ただそれを正しく認識するための情報を纏めえていないのではない．そのような意味でワークショップへの参加を希望したい．

蛇足であるが，開発環境ということがいわれるようになった背景には，開発するための機能として独自に必要な物を実現する必要性とともに，運用環境のなかでは，無理のきかないことが多々あり，それに振り回されていては効率的な開発ができないということにあったと思う．そういう意味で，開発環境は「進んでいなければいけない！」訳だけれども，「本当に現実はそうなのかなぁ？」という疑問がない訳ではない．

MMLの確立を目指して

セッション・サマリ

中野 秀男
(大阪大学)

1. はじめに

釧路の環境ワークショップのMMLの座長として，セッション・サマリー風にまとめてみたいと思います．ワークショップ終了後，すぐにまとめるべきだったのですが，タイミングを逸したというか，ズボラをしたというか，締切の12月15日を過ぎて正月に書き始めることになりました．手元には，その時のOHPのコピーと録音テープがあるので，それをもとにMMLにおける問題とその解決のための提案という形で，まとめたいと思っています．途中で文体が変わりますが気にしないで下さい．

議論は，予想されていたように，メインフレームを捨てる「ドラスティック」派と，現状のメインフレーム環境を徐々に改革しようという穏健派に分かれました．提案も，その２つの方向から出ていると考えて下さい．チェアマン自身は「ドラスティック」派ですが，おおらかなO型ですので，両論並記の形での報告になります．

最初，セッション・チェアマンの中野(阪大)から，セッションを始めるにあたっての１つの方向づけがありました．

このMMLのセッションは，メインフレームとWSやPCを正しく連携して使うにはどのようにすればよいか，はたまたメインフレームをどのように捨てるのかを討論する場ですが，前回までの環境ワークショップの流れから，自然にメインフレームの有効利用の話になりました．

メインフレームの長所としては:

- 速い
- ディスクやメモリ等の資源が豊富
- セキュリティが良い
- 今までのソフトがメインフレームにある

などがあげられると思います．最後の項目は，長所というわけではありませんが，メインフレームを考えるときには特に考慮すべき問題だと思います．

解決法としては，メインフレームを捨てて，スーパー・コンピュータやWSやPCですべてを行なえばよいと私自身は考えていますが，それでは討論にならないので，問題を整理したいと思います．

開発環境のワークショップなので，実機と開発機の話になりますが，

1. 実機がメインフレームで，開発機がWSやPCの人はどうすればよいか？
2. 実機もスーパーWSクラスのマシンにするにはどうすればよいか？

というあたりから考えてみたいと思います．

2. 久保さん（富士通）の発表

久保さんの発表は，前回のワークショップのMMLの話から始まり，メインフレームとその端末ネットワークとWS群からなる分散環境の比較になり，後者の問題点が，メインフレーム・ユーザドがWSになかなか移れない理由としてあげられた．

ネットワーク環境の問題点は次のとおりである:

1. 管理負担が大きい．
2. システム・インテグレーション能力が不足している．
3. トランスペアレンシーとポータビリティのための標準化努力が不足している．

メインフレーム・サイドが分散環境のネットワーク環境の優位性を理解しても，MMLにとどまらざるをえない理由がある．それは，

1. 管理負担の大きさ
2. 芸者意識　環境が実機の環境に媚びている
3. 巨大な資産

である．最後のソフト資産の話は，問題が大きすぎるとして，最初の２つの問題について，次のような解決策の提案があった．

［提案］

1. 管理負担の軽減
   ARPA Net/Junet からの技術の移転
   技術移転のための努力
2. 芸者意識からの解放
   ネットワークを使う体験をさせる

この発表に対して，フロアからは，次のような意見が出た：

［**佐原**］ 久保さんのあげたネットワーク環境の問題点は，実はメインフレームの問題点ではないか？メインフレームでは，管理が楽なのではなく，低いレベルの管理しかできていないのではないか？ネットワークに接続するような（端末ネットではない）レベルの管理をやる努力を怠っているように見える．

［**深瀬**］ 今回の MML の話は，メインフレームの於ける端末ネットワークの話ではない．プロセス間通信でいかに仕事を分業させるべきかを考えるべきである．

3. 三上さん（日電）の発表

三上さんの話は，メインフレームが実機で，開発環境を WS にした１つのシステムを作ったさいの経験報告であった．開発環境はみんな WS を使いたがるので，WS をメインフレームの TSS 環境に結合し，WS からは shell を使う感覚で作業させようというものである．特に，Unix からメインフレームを感じさせないでメインフレームを使うための１つのアイデアが報告されたと思う．このような環境は，Unix もメインフレームも使わないとだめな環境（例えば大学で最後の計算は，スーパーコンピュータでやりたい）では，Unix の環境だけ覚えればよいので便利だと思われる．

討論の中で，メインフレームがリアルタイム・トランザクション処理に向いていて，Unix は向いていないと言う意見が出たが，すでにアメリカでは座席予約システム等で Unix が使われており，セキュリティ・レベルでも Unix が負けていることはないという話が出た．使い勝手やネットワーク環境で WS が優れていることは確かなので，ターゲット（例えばリアルタイムトランザクション処理）には何が必要でどの様にすれば（現時点で）最適かが分かれば，自然に，よいものが悪いものを淘汰して行くであろう．

フロアからの提案は以下の通り：

［**深瀬**］ メインフレームをユーザが使うのは恥ずかしいことだという意識を広めるべきである．上の方の頭の硬い人は WS を買ってくれないという状況が現在はあるが，．端末ならいくらでも買ってくれるのだから，端末と同じぐらいに WS の値段が下がればよい．

このシステムは，日電のメインフレームの上に TCP/IP を載せ，更に Telnet を使うことにより実現されているので，更に結合レベルの高い NFS を登載すれば，より高いレベルの使い方が期待できる．その意味では，望ましい（メインフレームを残したい人にとって）MML の構築にいま一歩の所まで来ていると思う．

4. 林さん（SRA）の発表

アプリケーションの土台をメインフレームから剥したい．Unix もよいが，Mac ならもっとよい，MMI がよいから．それで本当にメインフレームが欲しいときだけ，メインフレームを使う．メインフレームで残るのは，データベースだけだろう．

過去のソフトウェアの遺産でも，まったく捨てるのではなく，こうすればもっとよくなりますよといって，ジワジワとメインフレームを捨てていく．Mac で Wingz や 4th Dimension などを使って，必要な時だけ SQL 等でアクセスに行く．

この発表をきっかけに，いままでフロアで黙っていた人たちが次々に口を開いて，次のような討論になった：

[篠田] メインフレーマは端末を作ってはダメという法律を作る（笑い）.

[林] Mac なんかは優れているのだけれど，ビジネス・パワーが足りないから MML をやっている．MML の発想をしているかぎりダメ．

[深瀬] Unix は事務用のパワーは少ないと言われるが，1990 年の買物としては，100GB 程度のデータベースを 200MIPS ぐらいのコンピューティング・パワーでサーバとして動かすことは可能で，しかも 2 億から 3 億円の投資ですむ．それを今のメインフレームで安く買っても，もう計算力が足りない．唯一のメリットはソフトウェア遺産だけです．

[篠田] データベースも専用データベース・マシンにかなわなくなるでしょう．何でもかんでも端末からメインフレームにデータを送るのはおかしい．セブン・イレブンで中年のおじさんがおにぎりを一つ買ったなんてデータを送る必要はない．もっと分散処理すればよい．MML で考えるべき唯一の問題は今までのソフトをどうするか．すべてリライトしてしまえばよいが.... 一つの解決策は COBOL を殺すこと．

[熊谷] MML で巷で問題になっているのは，高いメインフレームと安い PC やワープロがあって，高い金をだしてメインフレームを買ったのに，A4 版 1 枚の文書も綺麗にでない．ホストマシンには新しい利用には向いていない．例えば，DTP や CG やプログラミング環境等．改善策としては，限界のあるメインフレームでやる範囲を明確にする．MML または WML をまじめにやるには，それぞれの機能分担をはっきりし全体でないと動かないようにする．

[岸田] メインフレームのソフトウェア資産は本当にプラスの資産なのだろうか？それともマイナスの負債なのか？メインフレームのソフトウェアの文化とワークステーションのソフトウェア文化を繋ぐことは，もともと無理ではないのか？メインフレーム環境の中で王様のような権力を持っている人が率先してカルチャーを打破しないとダメだろう．カルチャーとカルチャーのリンクはナンセンス．お互いにどちらかがどちらかを打倒する戦いしかない！？これから 5 年から 10 年先を考えた時には，今の資産を捨てて新しくソフトウェアを作り上げた方がよい．

5. まとめ

（その後，とっても危ない話が続いたので，オフレコにしますが，面白かったのは，メインフレームの人たちの中からワークステーションや Mac に共感する人を引き抜いて行けばよいという提案．その人たちは，結構できる人だから，後に残ったメインフレームはボロボロになっていくだろう.）

チェアマンの一方的なまとめというか提案は次の通り：

メインフレーム上の巨大な資産は，技術移転をして行く考え方もあるが，メインフレームの機能と同じものを，よく，速く，安く作ってドラスティックに変えてしまおう．

管理負担の大きさ，いわゆる分散環境の管理の大変さの問題．これは，わざわい転じて福となす努力が必要．そのような福をためる仕掛を作りたい．管理に関しては，マジシャンが必要．みんながマジシャンでは困るが，マジシャンを優遇する処置を積極的にやるべきである．

話が途中から過激な方向に向かいましたが，ソフトウェア開発環境としては端末ネットワークよりは，ワークステーションをばらまいた分散環境が優れているのは明らかなので，来年の環境ワークショップでこの関係をするのなら，どのように異機種ネットワークでの開発環境を作るかを議論したい．恥ずかしながらのメインフレームも含めてですが.....

MMLの確立を目指して

セッションの討論を聞いて

佐原: メインフレームを使わないのが一番よい．どうしてもメインフレームを使わねばならないなら，ベル研で15年位前に実現していたPWBを真似すればよい．そのためには，(1) Ethernetをメインフレームがサポートすること，(2) UNIXのi/o redirection的機能をメインフレーム上に実現すること，(3) NFSのようなNetwork File Systemをメインフレーム上で実現すること，が実現されればよい．

濱田: メインフレームにはトップダウン方式の管理図式が確立していて，底辺レベルでは自ら考えようとしなくても，ソフトウェア技術者として飯が食えるのではないか？

↓

1人1台WSの世界は，1人称で考えられる．人材がそろわないとプロジェクトは成功しない．

高橋（晃）: CL/1的なアプローチは結構アナウンスされていると思ったが，まだなんですね．スキルを溜める方法，拡げる方法．ネットワーク以前の環境もある ??

青木: 昨年，Adele Goldberg女史にたずねたことがあります．Smalltalk-80はメインフレーム，またはミニコンなどに移植されないのですか？という質問です．女史はネットワーク上の一つのノードとしか考えていないと断言していました．

現在のSmalltalk-80に不足しているのは，RDBMとの結合であり大量オブジェクトの処理であると考えているようです．私は，メインフレームはDBマシンとしか考えていません．その意味では林香氏の意見に同感です．やはりMMLという言葉はなくしませう!「MainFrame In Network!」

魚田: 機能分散型結合のUNIXライクにメインフレームを使うという方法は非常に面白い．社内環境でも是非実現したい．

メインフレームにしがみついている人はどうなるのだろう．本気でメインフレームが一番よいと考えているソフトウェア技術者は，少ないように思う．多分，お金の話がかなり大きなウェートをしめるのではないだろうか？ WSでシステムを開発する技術者への評価（お金）を高くすれば，多くの技術者はWS (UNIX)の信奉者になるのではないだろうか？

高橋（肇）: MMLに関しては，今まであまり考えていなかった．ドラスティックに新しいものに変えるという方向には基本的に同意できるが，システム管理の問題は，一般ユーザとしてはやはり苦しいと感じる．

野村（敏）: 個人単位ではなく，会社単位で考えれば次のようにいえる：

- カルチャの変革には急激には行なえない（無用な混乱が生じる）．
- 現実のソフトウェア開発環境が分散化の方向に進んでいることは事実であり，緩やかに変化してきている．焦ることはない．
- 久保さんのいう「Mainframe in Network」の概念の実現がMMLであり，熊谷さんのいう問題解決方法が現実的であろう．
- 最近ではメインフレームの方が少し謙虚で，UNIX利用者の方がむしろ狂信的なような気もするが，いいすぎだとうか？

三上: MMLには2つの目標があるような気がする．

- 大きくメインフレーム側を変え，ユーザ全体をまきこんで，メインフレームをなくす方向．
- 今あるものをどうするかといった小さな問題を逐次解決して行く方向．

メインフレームは結局なくなるか，データベース専用マシンにしかならないような気がする．

** 私はメインフレーマではありません **

落水: SEAのアクティブな方々のものの考え方の基本がよくわかったという意味において，非常に有意義なセッションであった．

「技術的に優位なものにスムーズに移行できない巨大な体質がある．この体質を改善しようとする若人がスポイルされないように我々は頑張る必要がある」(深瀬)

「昔の遺産をそのまま使えますよ．もちろん新しいものも使えますよ．例: データベースと Hyper Card」(林)

SEAはなかなかすごい組織である!!

新田: 最近メインフレーム関連の仕事をしていないので，その現場はあまり知らない．やはりメインフレームの仕事をすべてWSでやってしまうには，言語とかDBなどの問題がある．それらについてメインフレームよりよいものをWS側で用意ししてやればメインフレームは消えていくと思う．

私にとっては，今の環境は

メインフレーム= SONY/NEWS

マイクロ= Apple/Macintosh

です．

岸田： MML といっても

1) MM Hardware Link なのか?

2) MM Software Link なのか?

3) MM Culture Link なのか?

1) と 2) の問題は何とか解決できる．しかし，3) はもともとナンセンスだろうと思う．Culture vs Culture の Battle しかありえないのではないか？

深瀬： MML は結局，メインフレームの端末ネットの中でいかに，少しでも使い易い環境を実現することでしかないのではないか？水平分散システムとは全く異質のものである．この意識改革は少し時間がかかると思われるが，かなりの努力をそそぐ必要がある．MML の議論を続けることがかえって意識変革を妨げることになる．

酒匂： MML は最早死語として扱わなければならない．世界は Computation，Storage，User Interface で構成されているので，その抽象的なコンポーネントをいかにうまく統合できるかが今後の課題．

岡本： 久保さんの「目標は " Mainframe in Network" で "Micro Mainframe Link" ではありません」には賛成です．これに以下をさらに付け加えたい．
「TPO に合わせてメガネを着替えるのは常識です....」のコマーシャルがあるように，「TPO に合わせてコンピュータを着替えるのは常識です....」といいたい．

平尾： 結局,"闘いしかない" というのがみんなの本音に聞こえた．メインフレームはよ知らないいが，変化としては DB マシーンになるしかないのか？

加藤： MML の抱えている問題については，認識できたような気がします．ただ，WS のレベルが上がってきて，メインフレームとの機能的な区別がなくなっていく，ということを考えると MML そのものの意味がよくわからなくなってしまいました．とするとメインフレームの今までの資産をどう活かしていくか，が重要になってくるのでは？

熊谷： メインフレーム文化と WS/UNIX 文化の対立の様相を呈したが，MML System は，そんなものではない．メインフレーム，UNIX，PC/ワープロを取り囲むレイヤの議論が必要だ．

文体の主語として，メインフレームとマイクロコンピュータが使用される時代はもう終わっている．ハード／ソフトの両面を含めた生活環境としてのコンピュータ環境を考える時期である．具体例を示すと，ワープロとスーパコンを連携させて使用する仕組みは何か，などである．さまざまな形態の情報を解体したり，融合したりすることが自由にやれるコンピュータ環境が重要である．そのためには，ネットワークも必要だが，それよりも情報をどのような形で表現するかが，本質的な問題になりそうだ．このような観点から考えると，Macintosh と IBM 汎用機の連携は，情報の表現とその取扱い方という面から，新しい局面を生み出している．

野村（行）： MML -> Micro Mainframe Network という久保さんの提案，考え方がよいと思う．ネットワークという概念は，メインフレームが登場した後にできたため，メインフレームの OS には，基本的にネットワークの概念が入っていなかった．「初めにネットワークありき」というアプローチで，新しい OS でメインフレームを動かした場合，メインフレームの長所は何が残るのだろうか？

久保： 中野さんのまとめた動向の実践が SEA の中で起きることを望む．

林： もはや MML は死語か？メインフレーム中心のアプリケーション開発は先ぼそり．メーカ自身が WS ベースに心変わりしはじめている．マイクロはマイクロでなくなってきている．マイクロだけでアプリケーションは開発できるようになるだろう（もうマイクロとは呼べないワークステーション）

↓

MML は不用．なぜならメインフレームがいらなくなるから．

↓

MML はやめてネットワークとして議論しよう．

↓

同じことになるかもしれない．

中野： MML のチェアマンです．今までいろいろ話が出てきたことが整理された感じがします．WS のパワーが益々上がってきて，メインフレームの出る幕が少なくなってきています．なぜ MML なのかの問題点も，メインフレームのもつ資産（または負債）をどう整理するかだけの問題になった感があります．

メインフレームはデータベース・マシンとスーパーコンピュータに変身するしかないし，1 台ですべてをこなす時代はもう終っているし，データベースも専用マシンが出ればそれが取ってかわるでしょう．これからは，ヘテロジニアスな計算機ネットワークで効率よく開発，運用することと，その管理に向かうことになりそうです．

Seesion 1: MML の確立を目指して (Post-Workshop Position Paper)

ワークショップを終えて

久保　宏志（富士通）

ワークショップがあってからかなり日がたつ．今日は 1 月 10 日である．議論の内容を思い出せない．だから以下に書くのはワークショップの MML セションにおいて発言したこととほとんど関わりがない．

1. MML が話題になるのは理由がある

前回の環境ワークショップの MML セションのサマリは私が書いた．改めて読み返してみた．悪くないと思う．このサマリを出発点にして，先に進むことを考える．

サマリが示した議論の到達点から次の認識を切り出す．

目標にすべきはネットワーク環境であって MML ではない．ところが Mainframe の側から MML ということがいわれる．Mainframe だけが特殊であってはならないのに，現実の話として MML が話題になる．

その事情を Mainframe の延命策であるとして無視したり，関心がないとして議論の対象にしないのは得策ではない．やはり Mainframe は特殊な存在であり，正面から取り組まないとネットワーク環境の実現は難しい．

なおここでいう「ネットワーク環境」とは，地球上のすべてのコンピュータが 1 つのネットワークにつながっていて，ネットワークにつながるどのコンピュータからも，すべてのコンピュータを使うことができるような環境を想定している．どのコンピュータを使っているかを意識することなく，つまりネットワークに向かって情報処理要求を述べるだけでコンピュータが使えるような状態が理想である．ネットワーク環境がこの状態に達すると，ネットワークがユーザ要求を満たすのに最適なコンピュータを選択して，ユーザのコンピュータをそれに接続してくれるのである．

まだ想像の世界でしかないが，この状態になると，コンピュータを A 社から B 社にリプレースするというようなことはなくなるのではないか？　ネットワークからあるコンピュータが消えるのは，それが使命をおえて廃棄されるときのみである．したがって，いわゆるコンバージョンのような非生産的な仕事は，地球上から消滅する．代わって重要性が高まるのは，現在メンテナンスという名前で呼ばれている仕事であろう．ネットワーク環境が，使われながら進化を続けるためにはこの仕事が欠かせないはずである．

2. MML 対ネットワーク環境の比較は生産的でない

ネットワーク環境が MML より優れていることは自明である．少なくとも次はいえる．ソフトウエア技術者はだれしも，ネットワーク環境の優位性を理解している．だから，これからはネットワークの時代である，と主張する．顧客の一般アプリケーションのためにネットワーク環境を提案して，その実現に手を貸している．自分たちだけが例外であっていいはずはないのだけれども，自分たちの環境をネットワーク環境にすることには熱心ではない．この一見矛盾しているように思われる状況を打開しないと，ソフトウエア技術者がネットワーク環境を持つことへの展望は開けない．

3. ネットワーク環境は新しい技術ゆえの問題を抱えている

目標にすべきがネットワーク環境であることは分かっていながら，実現しないのには理由がある．解決ないしは緩和を必要としている問題があるのである．そのさい，ネットワーク環境が新しい技術であるが故に持っている問題と，旧秩序から新秩序に移行することが引き起こす葛藤や摩擦とかに類する問題とを，分けて考える必要がありそうである．

まず前者に属する問題を考えてみる．次を挙げることができる．

問題の 1 つは，計算機システムの管理面にある．Mainframe の世界には，使う人と管理する人との分業体制ができている．管理が集中されているから使う方は楽である．多少は不便でも楽なのはありがたい．ネットワーク環境になると，便利さと引換に自分のマシンは自分で管理しないといけない．他人のことなら管理体制の重要性を説いて，それを作るように働きかけることができても，自分のこととなると，プロのプライドにかけても自分でやるしかないと感ずるのではないか？　しかし，そうはいっても，この負担に耐えてまでネットワーク環境を使おうとする人は少ない．ネットワークとつきあうには，相応の技術が必要であることも知っている．そ

れぞれの人が使ってもいいと感ずる程度に負担を軽減し，技術の壁を低くすることを考えることが糸口になる．

第２の問題は，ネットワーク環境にさまざまの素材をインテグレートする能力にある．これが社会全体に不足している．メインフレームの場合には，すでに多くのシステムをインテグレートしてきた実績があり，それに携わってきた技術者の数も多い．この問題は一挙には解決できない．地道に人を増やしていくしかあるまい．

第３の問題は，標準化の面にある．ネットワークを意識させないトランスペアレンシーは標準化を必要とする．資源の配置をネットワーク・ワイドに自在に行えるようにするポータビリティもまた，標準化を必要とする．幸いこの問題は社会から，その解決を強く要求されていることもあって，解決努力が続けられてはいる．しかし，ともすると誰の案を標準にするかのヘゲモニー争いの方に熱心になる傾向にあり，社会的なロスを生み出している．使う側がもっと発言を強めて，このヘゲモニー争いに終止符を打たせる必要があろう．

4. ネットワーク環境は保守主義を乗り越えて行かなければならない

ネットワーク環境の実現は，それを実現するのに必要な条件を具備するだけでは達成しえない．パラダイム・シフトを社会に求める要素を持っているからである．２種類の問題のうちの後者の問題である．

いわば発想の転換を求めていかなければならない．従来ソフトウエア技術者は，自分を殺して他人に奉仕することに努めてきた．この姿勢を逆転させて，自分のことを優先し他人へのサービスは二の次にすることである．考えてみればもっともよく理解している自分の要求を満足しえないで，他人の要求を満足させることができるはずがないではないか？　ソフトウエア技術を自分のために役立てることが先決である．そこで役立てた技術の一部を他人の役にもたてる．そう考えると，なすべきことの優先順位が自然に決まってきて，ソフトウエア開発環境のネットワーク化を加速することになるだろう．

シグマは「ターゲット・マシン接続」を最初からテーマに掲げて，ネットワークの方をターゲット・マシンに寄せる努力を行っている．ターゲット・マシンをネットワークの方に寄せさせるのが正しい道ではないか？「ターゲット・マシンのネットワーク接続」というべきであった．この関係を逆転させなければならない．

将来のネットワーク秩序としての OSI の実現にことの他，熱心である．このことは大事なことであるが，その一方で，現在すでに可能な IP リンクの果実を自分のために役立てることにあまり熱心ではない．逆転というほど大げさではないが，資源の配分を，現実的に可能なことを実行する方に変えなければならない．

古い秩序がその長い歴史の中で蓄積してきた巨大な資産について，ネットワーク環境はそれを当面は生かす道を用意する必要がある．過去の資産をすべて捨てることがネットワーク環境に移行するための必要条件であれば，旧秩序はネットワーク環境への移行に頑強に逆らうであろう．やはり平和共存的な移行シナリオを用意して，社会的なあつれきをちいさくすることが必要であろう．

5. 打開の処方箋を考えてみる

管理の問題の解決のためには，いままでネットワークの構築と運用に苦労を重ねてきた人たちが，仲間を増やすことにもっと熱心になることを期待したい．蓄積してきた技術を広く移転するために工夫を凝らすことも期待したい．そのことにより，技術の収得に必要な資格要件を引き下げることを期待したい．

junet がネットワークの普及に果たしてきた役割は大きい．その普及範囲を拡大したい．しかし，junet そのものはいろいろと制約があって誰でもが入れるとはいかないかも知れないので，junet の構築と運用で得た技術をより多くの人が享受できるように，別のネットワークを考えるべきかもしれない．シグマ・ネットワークに，この別のネットワークを期待した人は多かろう．

ネットワーク環境のインテグレータを増やす問題，標準化を急がせる問題は，社会にその問題解決を要求する声を大きくしていくことによってしか，解決できないであろう．

自分たちの権利を主張する声を大きくするためには，ソフトウエア技術者が IP リンクの素晴らしさを体験する機会を増やすことが先決であろう．ソフトウエア技術者にとっての IP リンクの素晴らしさを社会に向かってデモンストレーションすることも，必要であろう．シグマの「ターゲット・マシン接続」のような問題を逆の構図に書き換えることも必要である．Mainframe でも IP protocol をサポートする情勢にあることは，問題を易しくしている．旧体制の巨大な資産との共存にも道が開かれつつある．この有利な情勢を自分たちのために活用すべきである．

Seesion 1: MML の確立を目指して (Post-Workshop Position Paper)

マイクロメインフレームリンク (MML) の将来

三上　理（日本電気）

1. 現状

現在の MML はマイクロ側にメインフレームの「幽霊」アプリケーションをのせ，単にメインフレームの召使として利用しているにすぎない(多少の例外はあるが)．マイクロ側の性能が向上するにつれて，メインフレームでやっていた仕事が，マイクロ側でも十分にこなせるようになってきた．こういった，状況のなかで MML というものが今後どうなっていくか，どうすべきかを考えたい．

2. MML の今後

2.1 メインフレームの今後

現在の資産をまったく考慮しないと，これからのメインフレームの役割にはどういったものが考えられるだろうか？　集中的に利用するコンピュータとメインフレームを定義すれば，データベースしかも集中的に管理したいデータベースの役割，あとは自分のコンピュータの CPU を使いたくないような大きなジョブ（特に計算）を実行させるコンピュータといった役割しか思いつかない．したがって，そういった役割を担う専用コンピュータとして位置づけられるであろう．

しかし，たとえ長い目で見てそうなったとしても，現実的にはそうもいかないだろう．現在の資産の運用がユーザからとってみれば，大事な問題である．したがって，開発者側から見れば現在の資産や機能をうまく利用しながら，いろいろなコンピュータとの機能分担を考えて行くのが現状では得策であると考える．

2.2 マイクロの今後

マイクロ側の能力が今後ますます上昇していくことは，だれにでもわかることである．こういった，環境内でもやっぱり大きな計算はしたくない．こういった，作業はどこかで専用マシンを作りそれをみんなで共有する，といった形態で利用したい．

2.3 MML の今後（メインフレームは機能サーバである）

現在の資産を有効に利用しながら，「幽霊」マイクロをやめる．こういったことを考えなければならない．それには，マイクロ側とメインフレーム側の機能間のインタフェースを相手にし，連係をとりながら独立に使えるようにするのがよいと思う．そうすると，メインフレームは特定の機能だけを提供してくれるサーバみたいになり，いまのような使いづらい環境はなくなるだろう．機能間インタフェースには次のようなものが考えられる．

(1) 通信インタフェース

　メインフレームは早く専用回線だけというのを脱皮し，種々の通信に対応しなければならない．これは，既存のものはほとんどすべて標準化の項目中に入っているので，あんまり心配していない．

(2) AP 間インタフェース

　こちらの方がむずかしいだろう．たとえ何らかの形式をきめても，新規 AP の場合にはよいが，今あるやつをどうするかという問題が残る．そこで，ここの形式を決めつつ，いまあるやつを救うためインタフェースの変換（標準的なもの，世の中で通用するもの）をするような仕組みも必要である．

2.4 そこで，どうする？

そこで，こうやって欲しい，こうやったらよいという提案的を述べてレポートを終了したいと思う．

(1) メインフレームは早く種々の便利な通信に対応する．

(2) 通信でつながるようになれば，次に AP 間のインタフェースの整備をする．たとえば，データベースアクセスのインタフェースなど．

(3) あとは，既存の AP をうまく利用できるように AP 間インタフェースの変換機構を用意する．

というわけで，現在の MML のイメージはまったくなくなり，能力の高いマイクロと集中処理コンピュータであるメインフレームの世界ができる．

Seesion 1: MML の確立を目指して (Post-Workshop Position Paper)

第 5 回 SEA 環境ワークショップに参加しての感想

野村 行憲（岩手電子計算センター）

1. MML の確立を目指して

特に興味を引かれたのが，メインフレーム(大型機)に Unix ワークステーションを接続し，ワークステーション側に独自のツールを置くことによって，大型機の操作を知らない人であっても，ワークステーションを使っている方法で大型機を使えるようにした例であった．

ソフトウエア開発の次世代を担うと言われている Unix ワークステーションは，その優れた環境のため，プログラマのメインフレーム離れを誘発する．しかし，大型機のパワーを有効に活用することも，ソフトウエア開発の効率を考える上で重要なことである．

この２つの矛盾を解決する１つの方向として，このような試みは重要であろう．

振り返って，我が社の状況を見ると，2050 という Unix ワークステーションが，日立のメインフレームに接続されてはいるが，この接続は単に，TSS 端末をエミュレートしているに過ぎない．つまり，大型機（M-260）を使うには，大型機の環境をかなり学習しなければ使えないのが現状である．このため，効率的な開発環境であるとは言えないのが現実である．

一方こうしたことから脱却できない決定的な理由はメーカー側にもある．現在日本の３大大型機メーカーのうち，Unix とインタラクティブに会話するための通信手順（プロトコル）である TCP/IP をサポートしているのは，現状では日本電気しかない．

しかし，ソフトウエア開発環境の Unix へのシフトは，まだしばらくは続きそうな見通しであることから，この分野の動向には目を離せないものがある．

もう１つ今回のワークショップで感じたのは，メインフレームを必要としなくなってきつつあるということである．ワークステーションのパフォーマンスが飛躍的に向上し，メインフレームでなければできないというものが，なくなってきている．

2. Hyper Text の光と影

情報の新しい表現手段として最近脚光を集めている，ハイパーテキスト（またはハイパーメディア）の実現の例とその技術，問題点を議論した．

容量の問題，速度の問題は，エンジン（ハードウエア）の進歩によって解決されるであろう．しかし，情報間のリンクの張り方や，ブラウジングの方法は，統一的な指針が確立されていないこともあり，開発者のセンスと利用者の感覚にギャップが生じている．

また，ハイパーメディアは所詮大規模なデーターベースには向いていない技術であろう．しかし，その技術は小中規模のデーターベースの検索手段としてもっとも有効な手段になると思う．

3. ソフトウエア工学は現場に生かせるか？

ソフトウエア工学の研究者であり，教育者である大学の先生と，現場で仕事をしている企業人との立場の違いを反映した，白熱の議論は，大変興味深く，時間が短く感じられた．

しかし，残念だったことは，「ソフトウエア工学のよい教科書がない」ということであった．たしかにそれは現実であろうが，日本の大学でそれを作ろうとする動きもなさそうである．

国外の文献を紹介したり，解説したりすることが，ソフトウエア工学のであると思っている研究者が多いような気がしている．

何を研究のテーマとしたらよいのかを模索しているというのが，日本でのソフトウエア工学研究の現状だと言ったら言い過ぎだろうか？

4. 企業間ネットワークの構築と活用

今回は，ある架空のプロジェクト（第４次金融オンラインの機能設計）を請け負った会社が，いくつかの会社と連係して企業体を組織し，ネットワーク環境で仕事を分担して進めるという想定で，それぞれの局面でどのような問題があるのかを議論した．

独占禁止法や，利益の配分，異なるネットワーク環境の統合と行ったはばの広い問題が明らかになったセッションであった．しかし，この考え方は，大規模なシステムを効率的かつ確実に開発する方法として，最も有効な手段であるという気がしてならない．

企業間ネットワークとまでは行かなくても，分散ネットワーク型の環境は，あちこちでその有用性が報告されてはいるが，まだ普及段階にある．これが定着して，さらに業界が統一的に力が結集されれば，必ず企業間ネットワークが必要になるであろう．いや，業界を結集して力を持つために必要なことかもしれない．

5. ツール・インテグレーションの理想と現実

この中で特に興味を引いたのは，FXIS の青木さんが発表した言語（あるいは環境と呼ぶべきかも知れない）: Small Talk-80 の新しいリリースである．Smalltalk-80 は以前から注目していたが，速度の遅さと，メモリー消費の多さなどの問題があった．しかし，新しいリリースではこの２つの点が改良され，一段と実用化が進んだ．

しかも，個人的に気になっていた Macintosh 用のものが，10 万円を切る値段になったことには，盛大な拍手を送りたい．

Unix の次に来るもの，それは SmallTalk という印象が強い．

6. その他

このワークショップに参加して，自分の勉強不足が強烈に認識させられた．議論に付いて行くのに，かなりの努力を要求され，辛い思いであったが，その分だけ得るものも多かった．

今後，何をどのように学んでいくべきかが，おぼろげながら見えてきたようなワークショップであった．

Session 2

HyperText の光と影

(1989 年 11 月 30 日 09:30 − 12:30)

Chair 熊谷 章（PFU & 富士通）

Speaker 加藤 康人（PFU）
高橋 肇（日本電子計算）
野村 行憲（岩手電子計算センター）
高橋 晃（釧路高専）
篠田 陽一（東京工業大学）
佐原 伸（SRA）
新田 稔（SRA）

討論の狙い

HyperCard on Mac の華やかな登場をきっかけとして，ハイパーテキスト技術の応用が，いま，要求定義や設計，ドキュメンテーション，さらにはメインテナンスといったソフトウェア開発のさまざまな側面で脚光を浴びているようです．ある意味で，それは，オブジェクト指向プログラミングの正しい発展型であり，知識ベース技術とうまく組み合わせれば，現在のソフトウェア開発が抱えている多くの問題点が解決できるだろうという楽観的な予想もあります．一方，批判的な目で見れば，スパゲティ風のハイパーテキスト・ネットワークは，GoTo 分を乱用した昔のプログラムと同じものであり，もう一度構造化革命を発動する必要がありそうです．

このセッションでは，具体的なケースを対象に，ソフトウェア開発においてハイパーテキスト技術をどのように利用すべきか，それによってもたらされるメリットや付随する問題点は何かを，さまざまな角度から検討してみたいと考えています．

Session 2: HyperText の光と影

目次

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

Hypermedia の今後の展開

熊谷 章（PFU & 富士通）

1. Hypermedia の定義

情報はあるユニットの束である．ユニットの情報はウィンドウに表示される．ユニットはリンクによって関係づけられる．ユニットの創成，編集，関係づけによって，さまざまな情報が表現される．Hypermediaは共有され，マルチアクセスが可能である．

- Key Issues: Data Model, Direct Manipuration.

2. データモデルの問題

1) 各ノードの表現にどんなデータモデルが適切か？
2) どんなサイズが必要か？
3) どんなタイプのノードがあるべきか？
4) 関係のリンクのソースとしてどんな種類のオブジェクトを用いるべきか？
5) リンクのデスティネーションにはどんな種類のオブジェクトを用いるべきか？
6) どんな種類のリンクが必要か？
7) リンクは内部構造を持つべきか？
8) ノードはどのようにしてより大きな構造にまとめられるか？
9) バージョンサポートはどのようにするのか？

3. ユーザインタフェースの問題

1) どんなスタイルのユーザインタフェースを使用すべきか？（Direct Manipuration Paradigm）
2) ノードはディスプレイ上でどのように表示されるべきか？
3) リンクをディスプレイ上でどのように表示するか？
4) リンクを辿った時，システムはどれくらいの速度でレスポンスすべきか？（＜ 0.25 秒／ノード）
5) システムはブラウジングをどのようにサポートすべきか？
6) データベースのグラフィカルなビューは必要か？
7) いかにして "Lost Space Problem" を解決するか？
8) ブラウジング以外の検索方法として何が必要か？
9) ユーザインタフェースをどのようにしてカスタマイズするか？

4. 共同作業における Hypermedia の問題

1) 情報はいかに集中化され共有化されるか？
2) マルチユーザのオーバヘッドをいかにして少なくするか？
3) クリティカルなデータへのアクセスの禁止をどうやって実現するか？
4) コミュニケーションをどのような形でサポートするか？
5) アノテーションのサポートの仕方をどうするか？

5. Hypermedia の今後の課題

- システムの拡張性： Programmng Language が必要（例：Hypertalk）
- 巨大なデータベース作成が難しい．
- 電子情報から紙情報へのコンバート
- Hypermedia の自由度の大きさは利用者には両刃の剣である．
- 採用したデータモデルがそのシステムのすべてを決める．
- 人間とコンピュータの交信の問題： ソフトウェアシステムの複雑さは半永久的に増大する．Hypermedia ではデータモデルの標準を研究開発することができる．Desktop Metaphor の革新が可能である．

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

HyperBook と Prototyper's Work Bench

加藤 康人（PFU）

1. HyperBook

HyperBookは、大量のイメージデータを格納し、それらをより実際の本を扱うのに近いユーザインタフェースで扱え、さらにはそれらのイメージデータ間にユーザが任意の関係付けを行うことができ、その関連を辿ることにより必要なデータを容易に引き出せる機能を備えています。

HyperBook特有の環境として、アイコンベースの作り出す世界があります。 HyperBookの中には、ユーザが独自のアイコンを作ることのできる道具箱(ToolBox)があり、そのなかの道具を使って自由にアイコンを定義することができます。アイコンには包含関係にあるものを階層的に整理して格納できる容器のようなロッカー型のものと、アイコン自身がある機能を持っている (ウインドを閉じたり、移動したりといったもの)機能型のものがあります。 アイコンを作り出すのも1つの機能型のアイコンといえます。

ロッカー型のアイコンの特殊なものが"本"であり、本は複数のシートから構成されます。本は実際の本のように厚みがあるように表示され、 ユーザはだいたいこのあたりのページ、といったように感覚的にシートを探し出すことができます。また、任意のシートにタグを付けて情報を整理することもできます。本をめくる、という操作のために"バンドラ"とよばれるアイコンがあり、本をパラパラめくっていくこともできます。

本の中にあるシートには、さらにエリアと呼ばれる四角の領域を幾つも定義することができ、それらの間をいろいろな種類のリンクで結ぶことができます。このようなリンクは、同一シート上のエリア間、同一の本の異なるシート上のエリア間、そして異なる本のシート上のエリア間でも結ぶことができます。

HyperBookでは、このようにしてまず大量のイメージデータを本にして格納してしまい、それらをいろいろなロッカー型のアイコンの中にいくつかの本にしたりして整理し、関連のあるイメージデータ間に自由にリンクを結び、このリンクを辿って視覚的にデータの検索が行えます。 リンクは方向性を持っており、どのリンクを辿ってきたかということもわかるようになっています。また、ロッカーにしまわれていたり、閉じられている本であっても、その本のシートにリンクが張られていれば自動的にその本を開いて必要なページを開いてくれます。

2. PWB(Prototyper's WorkBench)

私はここ数年間、 ソフトウェア開発の上流工程である要求定義・分析のための支援ツールPWB(Prototyper's WorkBench) の開発を行っています。 PWBは、システムのソフトウェアに対する要求を6つのモデルを用いてモデル化し、要求の漏れや矛盾を検出する機能と、それらのモデルを連動させて動かしてシステムの動的な振る舞いを検証する機能を持っています。

システムのモデル化は、ある特定の規則に従ったERA(Entity-Relation-Attribute) ダイアグラムでシステムを記述していきますが、ここではシステムを段階的にある部分をより詳細化したり、あるいは抽象化したりといった階層化の機能が必要となります。これは、ダイアグラムのあるEntityを別のダイアグラムにより詳細に記述したり、ダイアグラムを別のダイアグラム中のあるEntityに抽象化したりすることで、言い替えればEntityとダイアグラム間にある種の関係付け(この場合は、階層関係)をすることになります。 このような階層の関係付けが容易に行え、それらの階層構造を表示した

りする機能が必要でした。

またシステムの動的な振る舞いを検証するために、ダイアグラムの記述に従ってモデルを疑似実行するわけですが、複数のモデルを連動させる(ダイアグラム中のあるEntityが実行されたら別のダイアグラムを駆動したり、Entity 間で実行の同期をとったりできる)ためには、関連のあるダイアグラムのEntity 間にもある種の関連付け(駆動ポイント、同期ポイントなど)を行える機能が必要でした。

そこで、PWBでもダイアグラムを記述するためのツールやそれらを動かすための機構だけでなく、各種のダイアグラム群を格納したり、それらのダイアグラム間の関連付け、その関係を格納、あるいは必要なダイアグラムを容易に探し出せる機構が重要になってきました。

3. HyperBookとPWB

現在、これらのHyperBookとPWBを統合しようとしています。 つまり、HyperBookの本の中にPWBで扱うダイアグラムをシートとして格納できるようにし、PWBのモデルの階層関係や疑似実行のためのEntity間の関係を HyperBookのリンクと使って実現しようというものです。

ここでのポイントは次の2つでした。

(1)HyperBookでは本のシートの中身はすべてイメージデータを考えているが、 PWBで扱うダイアグラムは、ERAモデルという構造を持っている。HyperBook のシートとシート内のエリアという概念をどうダイアグラムに取り込むか？

(2)HyperBookではユーザがリンクのタイプを設定したり、また、そのリンクのタイプを使って自由に関連付けを行うことができるが、PWBではモデルとして意味のあるリンクしか結べないよう制限しているがこれをどうするか？

そこで、ダイアグラムを中身とするシートにはそれを表示するためのイメージデータを持つようにし、ダイアグラム中のEntityの存在する場所に自動的にエリアを定義し、かつそのエリアにはEntityの持つ各種情報を保持することによって、本の中でもダイアグラムのシートをイメージデータのシートと同じ操作で扱え、Entityとダイアグラムをリンクで結んだり、 Entity間にリンクを張ったりという操作も、それぞれエリアとシートをリンクで結んだり、エリアとエリアを結ぶ(これらの場合には、リンクを結ぶときに結べるリンクのタイプを制限する)ということで実現してきました。

ここまではほぼ実現の見通しができ、HyperBookでイメージデータ以外に文章のようなテキスト情報、あるいはダイアグラムなどのモデル情報が本の中に格納でき、それらの各種情報間でリンクを結び、視覚的に検索したり、PWBの疑似実行機能を使っていくつかのモデルを動かしてみることができるようになりました。

これからは、HyperBookをソフトウェアの要求定義･分析を支援するための環境として適用していく必要があると思います。具体的には次のようなことがあげられます。

(1)複合シート

現在シートの中身にはイメージデータ、テキスト、ダイアグラムといったある1つの種類の情報が入っています。 これをあるエリアにはテキスト、あるエリアにはダイアグラムというように1つのシートの中にいろいろな情報を混在できるようにする必要があります。

(2)Version Control

HyperBookでどこまでの情報をVersionとして管理するのかを検討する必要があります。つまりシートや本のような情報のかたまりだけでよいのか、 情報間のリンクまで管理する必要があるのか、あるいはそのリンク先の情報は、などなどです。

(3)いろいろなシート

これはPWBのモデルの疑似実行の機能の1つに、シミュレーションなどの測定を行うものがあるのですが、その測定結果のグラフなどをシートとして本に入れて残しておこうというものです。 このように何らかの分析結果を自動的にシートとし、ある分析からその結果が得られた、というようなリンクを自動的に結ぶことにより、後でそれを参照することができます。

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

ソフトウェア開発において ハイパーテキスト技術をどのように利用すべきか？

高橋 肇（日本電子計算）

私の仕事は，エキスパートシステム開発であるので，その観点から意見を述べさせていただきます．

１．エキスパートシステムの開発

エキスパートシステム開発において重要なのは，知識獲得のための技術である．例えば，ワークシートを作成したり，インタビューしたり，作業現場を見学したりとかなり泥臭い仕事である．

これは，対象問題の専門家の知識はきれいに整理されているわけではなく，断片的な経験知識等を組み合わせることによって，各種の判断を行っていることが多かったり，専門家が常識的に判断しているような知識も記述しなければならないことがあるからである．

また，専門家から獲得した知識を整理してモデル化を行わなければならない．そこから先の作業は，エキスパートツール等の機能でカバーできる範囲である．

２．ハイパーテキストの利用

ハイパーテキストシステムにおいては，カードに情報を記録し，カードとカードをリンクして情報を記録している．書き方に特に制限は無く，自由に記述できるので，情報を記録しやすい．その点で，知識をまとめる道具としては有効と考える．

知識獲得という点では，書き方が自由すぎると知識が出しにくいと思われる．例えば，真っ白いカードをたくさん渡されただけで，知識を書いていけと言われてもそうたくさんの知識を出せるわけではない．これは，無意識のうちに判断している知識は，簡単には記述しにくいからである．

では，どのようにしたら良いか？

例えばカードのリンクの階層化を視覚的に行っていくことによって知識獲得の支援をする．これは考えなければならない範囲をしぼることで知識を出しやすくし，全体の構成を確認しながら進めていくことで，不足している点などがわかりやすい．但し，この場合リンク方法を自由に行わせるのではなく，ある程度制限を設けて行う方が，問題をモデル化するという点で必要であると思う．

３．結論

ハイパーテキストは以上のような試行錯誤を必要とする局面で有意義な道具であると考える．

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

HyperCard はソフトウェア開発ツールになりうるか？

野村 行憲（岩手電子計算センター）

はじめに

HyperCard が Macintosh(Apple) の上で動き始め、私の手元にも日本語版が '89.8.31 に届いた。

何時の場合でも新しい言語や環境に触れるときには、好奇心と億却さとの葛藤があり HyperCard の場合でも例外ではなかった。しかし、最初のハードルを乗り越えると、その容易さに引き込まれ、他の言語を使うことが億却になってしまっていて、現在は自分のためのアプリケーションは、Hyper Talk だけで開発している。

このアプリケーション開発の容易性（とくにユーザーインターフェイス部分）を実際のアプリケーション開発に利用できるか検討してみたい。

適用例

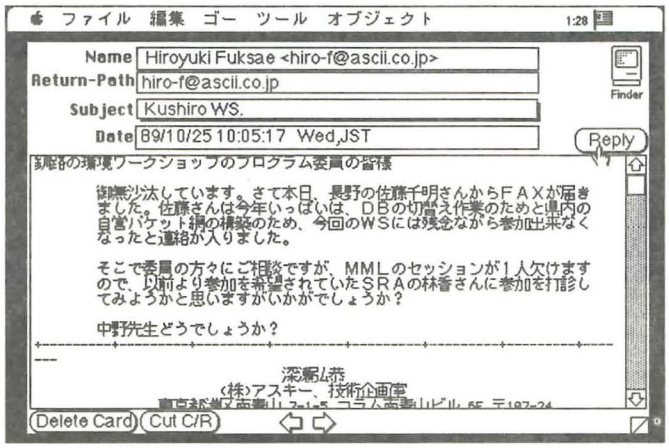
HyperCard (Hyper Talk) を使用して、作成したアプリケーションの例を上げる。

自分用

junet の mail を整理するスタック

上の画面は最初の画面で、今まで到着した mail の subject と発信者が一覧出来るようになっている。

右下のアイコン（書き込み）は通信ログを読み込んで mail を整理する機能ボタンである。取り込まれた mail は一覧表に付加され、一覧表の中から読みたい mail の行をマウスでクリックすると、次の図のように mail 1件1枚のカードが表示される。



この mail に対して返事を書きたい場合は、右上の Reply ボタンを押すと、次のように返信を編集するカードがあらわれる。

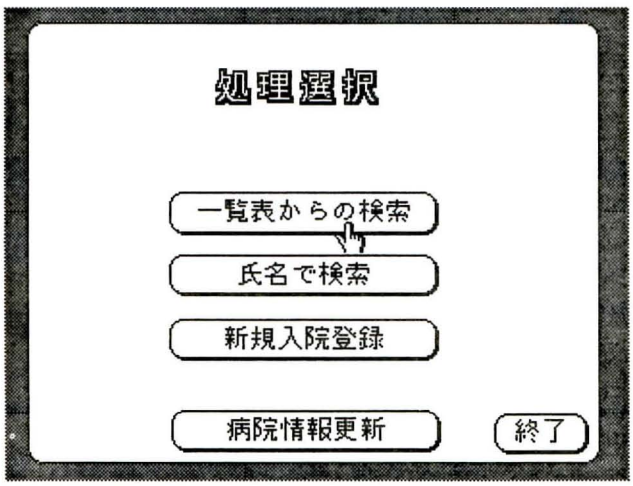
勿論、ここで編集したテキストは unix の mail コマンドの体裁で file に書き込むことが出来る。

このスタックウエアを開発するのに要した時間は、およそ8時間程度であった。

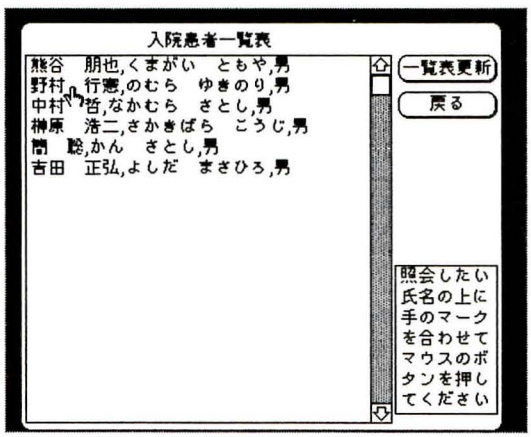
同様のアプリケーションを、従来の Pascal or C などで開発したらどれぐらいかかるのだろうか?

一般用

一般用の例として、病院の受付で使用する患者案内のシステムを作成した。



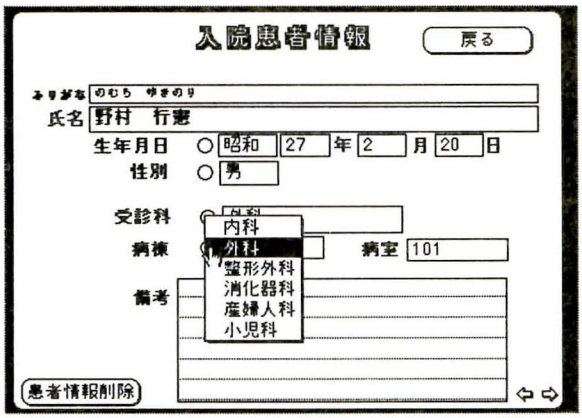
最初に上の処理選択画面が表示され、一覧表からの検索をクリックすると、次の画面が表示される。



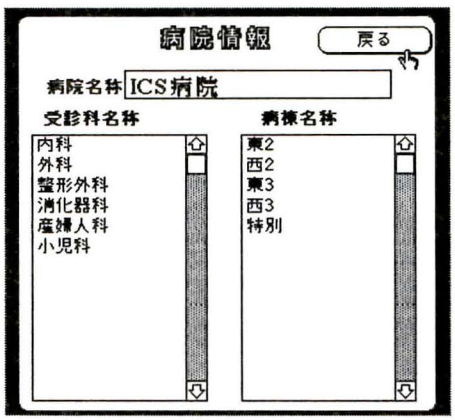
ここで、名前をクリックすると、その患者の情報が次のように表示される。

この患者情報のカードは何時でも内容を書き換えることができる。

新規入院患者などの入力は、図のようにポップアップメニューによって入力できるようになっていて、入力を簡単にしている。



このポップアップメニューに出ている内容は、導入された病院によって違うので、次の図のような病院情報を登録するカードを作っている。このカードは処理選択で病院情報更新をクリックすると表示される。



このスタックと同じ目的で、MS-DOS の16bitマシン上で動くアプリケーションを Level-II COBOL で開発するのには、1週間を要したが、HyperCard では、半日で完成した。

また、HyperCard 版では、マウスによる優れたユーザーインターフェイスを実現しているが、MS-DOS 版のそれは、キャラクタ表示とキィボードインターフェイスのものになっている。

問題点

このようにHyperCard(HyperTalk)では、アプリケーションが簡単に、短時間で作成できるが、大量のデータを扱ったり、文字列の操作などに速度上の問題がある。

これを解決するのには、XCMD（拡張コマンド）やXFCN（拡張関数）をPascal or Cで作成するしか方法が無い。

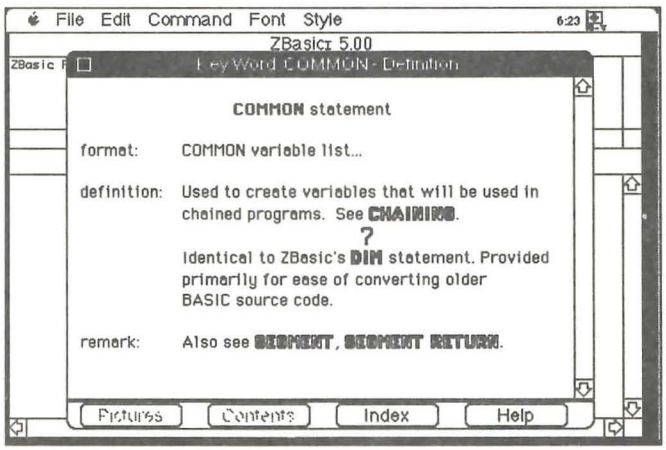
また、現在はインタプリタとなっているので、作成されたスタックウエアは、HyperCard上でしか動作しない。

この2番目の問題は、HyperTalkを開発環境として使う場合、OSの下で単独に動作するアプリケーションが作れないということにほかならない。

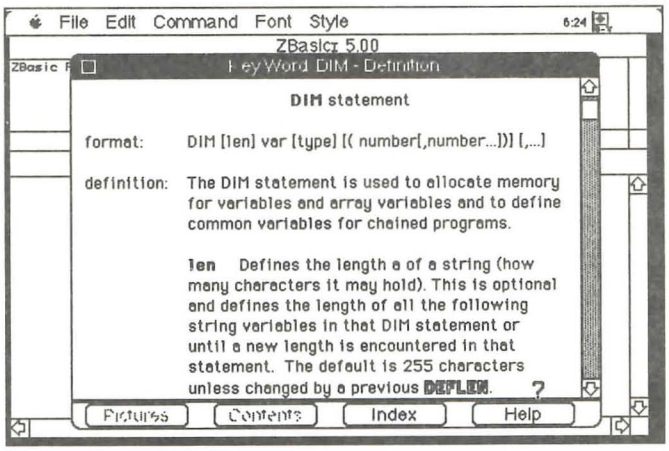
（現時点ではコンパイラ機能はまだできていない）

HyperTextの応用例

HyperCard以外で、HyperTextを応用した例として、マニュアルのHyperText化があろう。Macintosh版のZBASICは、HelpがHyperText化されているので、例に引用した。(ZBasic Ver. 5.0 Macintosh版)



この図で、白抜きで表示されている単語は他に説明があることを示している。?マークはマウスカーソルで、DIMをクリックすると次の画面が表示される。



このDIMの説明の中には、更にDEFLENなどの説明がリンクしていることが分かる。

このようにマニュアルへのHyperTextの応用はすでに実現されている。

ちなみに、このHelp Systemを構築するためのシステムは、開発者向けに発売されている。

おわりに

優れたユーザーインターフェイスを持ったアプリケーションを開発するための、使いやすい環境として現在リリースされているものとして、他にどのようなものがあるのか分からないが、HyperCardの上で実現されたものが、例えばX-Windowのソースで出力されるとか、HyperCard自身が性能や機能を向上して、優れた開発環境になるのか今後の方向性を模索できるような話しが、釧路で聞けることを期待している。

また、これ以外の開発環境の話しも聞けそうなので、今、私が気に入っているHyperCardよりも可能性を秘めたものがあるであろうことも期待している。

（SmallTalkなどに興味あり。）

junet mail の script

```
Script = *** Mail.Junet1 ***
on deleteAllCards
  set cursor to 4
  set lockScreen to true
  put (the number of cards) -2 into countCard
  go to card 3
  repeat with i = 1 to countCard
    put "delete card.... rest  " & countCard  -i
    domenu "Delete Card"
  end repeat
  go to card 1
  put empty into field "mail_list"
  set lockScreen to false
  hide message window
  set cursor to 1
end deleteAllCards

-- convert to date from unix-std to jpn-std
function jDate dt
  put char 1 to (offset(",",dt)-1) of dt into youbi
  delete char 1 to offset(",",dt) of dt
  delete char 1 to offset(" ",dt) of dt
  put word 1 of dt into hi
  delete char 1 to offset(" ",dt) of dt
  put word 1 of dt into mon
  delete char 1 to offset(" ",dt) of dt
  put word 1 of dt into nen
  delete char 1 to offset(" ",dt) of dt
  put char 1 to offset(" ",dt) of dt into jikoku
  delete char 1 to offset(" ",dt) of dt
  put word 1 of dt into flg
  get offset(mon,"JanFebMarAprMayJunJulAugSepOctNovDec")
  put trunc((it-1)/3)+1 into tuki
  return nen&"/"&tuki&"/"&hi&" "&jikoku&" "&youbi&","&flg
end  jDate



*** Background = bkgnd id 3655 ***
*** bkgnd button "Prev" ***
*** Location: 220,326 ***
*** Rect: 207,316,234,336 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  visual effect scroll right
  go to prev card
end mouseUp
*** bkgnd button "Prev" ***
*** Location: 220,326 ***
*** Rect: 207,316,234,336 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  visual effect scroll right
  go to prev card
end mouseUp
*** bkgnd button "Delete Card" ***
*** Location: 60,326 ***
*** Rect: 18,317,102,335 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  Answer  "Delete this card ?" with   "Delete" or "Cancel"
  if it = "Cancel"  then
    exit mouseUp
  else
    domenu "Delete Card"
  end if
end mouseUp
*** bkgnd button "Cut C/R" ***
*** Location: 135,326 ***
*** Rect: 104,317,166,335 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  global packID, undoBuff
  answer "Want you delete C/R of line-end ?" with "Undo" or  "Cancel"
¬
  or "Cut"
  set the cursor to 4
  if it is "Cancel" then exit mouseUp
  if it is "Undo" then
    if ID of this card <> packID then exit mouseUp
    put field "MsgData" into buff
    put undoBuff into field "MailText"
    put buff into undoBuff
    exit mouseUp
  end if
  put ID of this card into packID
  put field "MailText" into undoBuff
  put number of lines of field "MailText" into nL
  put empty into buff
  repeat with i = 1 to nL
    put line i of field "MailText" into theLine
    put empty into lineEnd
    put length of theLine into len
    if len < 2 then put return into lineEnd
    else
      put char len of theLine into last1
      put char (len - 1) of theLine & last1 into last2
      if offset(last1,".?!") > 0 then put return into  lineEnd
      if offset(last2,"。．：？！") > 0 then put return into lineEnd
    end if
    put theLine & lineEnd after buff
  end repeat
  put buff into field "MailText"
end mouseUp
*** bkgnd button "新規ボタン" ***
*** Location: 491,328 ***
*** Rect: 480,318,503,339 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  visual effect scroll down
  go to first card
end mouseUp
*** bkgnd button "Finder" ***
*** Location: 482,51 ***
*** Rect: 465,28,499,75 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  doMenu "Quit HyperCard"
end mouseUp
*** bkgnd button "Reply" ***
*** Location: 470,104 ***
*** Rect: 441,94,499,114 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  global FromStack, Nm,Rto, Subj, Cc, MailText
  put field "From" into Nm
  put field "ReturnPath" into Rto
  put field "Subject" into Subj
  put Empty into Cc
  put field "MailText" into MailText
```

```
  put the name of this card into FromStack
  put " of " after FromStack
  put the name of this stack after FromStack
  visual effect dissolve
  go to stack "Reply Edit"
end mouseUp
*** Background = bkgnd id 2738 ***
*** bkgnd field "mail_list" ***
*** Location: 256,177 ***
*** Rect: 26,59,487,295 ***
*** Visible: true ***
*** Style: scrolling ***
*** Script: ***
on mouseUp
  put item 2 of the clickLoc into vPos
  put item 2 of the rect of field "mail_list" into wPos
  put the textHeight of field "mail_list" into tHeight
  put the scroll of field "mail_list" into sPos
  put trunc((vPos-wPos+sPos)/tHeight)+1 into nLine
  go to card nLine+2
end mouseUp
*** bkgnd button "Prev" ***
*** Location: 220,318 ***
*** Rect: 207,308,234,328 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  visual effect scroll Down
  go to prev card
end mouseUp
*** bkgnd button "ImportMail" ***
*** Location: 476,313 ***
*** Rect: 457,299,496,327 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  global loopStart
  global indx_data
  global makeCard
  get fileName("TEXT")
  if it is empty then  ask "Input file name?"
  if it is empty then exit mouseUp
  put empty into msg_work
  put false into makeCard
  put it into fName
  open file fName
  put "We Make new cards now...Please wait."
  set the cursor to 4
  set lockScreen to true
  go to last card
  --  Main Loop  -----
  repeat forever
    read from file fName until return
    if it is empty then
      put "File" && fName && "close."
      close file fName
      exit repeat
    end if
    put it
    if word 1 of it = "From:" then
      cardMake
      delete char 1 to offset(":",it) of it
      put offset("(",it) into pos
      if pos>0 then delete char 1 to pos of it
      put offset(")",it) into pos
      if pos>0 then delete char  pos of it
      put it into field "From"
    else
      if char 1 to 5 of it = "From " then
        delete char 1 to offset(" ",it) of it
        put offset(" ",it) into pos
      if pos>0 then
        cardMake
        put char 1 to pos of it into field "ReturnPath"
      end if
    else
      if word 1 of it = "Return-Path:" then
        cardMake
        delete char 1 to offset(":",it) of it
        delete char 1 to offset("<",it) of it
        delete char offset(">",it) of it
        put it into field "ReturnPath"
      else
        if word 1 of it = "Subject:" then
          cardMake
          delete char 1 to offset(":",it) of it
          put it into field "Subject"
        else
          if word 1 of it = "Reply-To:" then
            cardMake
            delete char 1 to offset(":",it) of it
            put it into field "ReturnPath"
          else
            if word 1 of it = "Date:" then
              cardMake
              delete char 1 to offset(":",it) of it
              put jDate(it) into field "Date"
            else
              if  it = return and makeCard then
                --    cardMake
                put empty into msg_work
                --- mail text setUp Loop ---
                repeat forever
                  read from file fName until return
                  if it is empty then
                    put "File" && fName && "close."
                    close file fName
                    exit repeat
                  end if
                 if char 1 of it ="&" or char 1 of it = "%" then
                    put msg_work into field "MailText"
                    put false into makeCard
                    exit repeat
                  end if
                  put it
                  put it after msg_work
                end repeat
              end if
            end if
          end if
        end if
      end if
    end if
  end if
end repeat

-- Makeup Title table
go to first card
put "We add new title now...Please wait few minits."
put field "Mail_list" into indx_data
put  field "numbCard" + 3 into loopStart
put the number of cards - 2 into field "numbCard"
--
makeTable
-- ***Compress the stack
--  put "We compress the stack...Please wait"
--  domenu "Compact Stack"
set lockScreen to false
--
--*** set scrool to last **
--
put item 2 of the rect of field "Mail_list" into uPos
get item 4 of the rect of field "Mail_list"
put it - uPos into wHeight
put the textHeight into tHeight
```

```
  put the number of lines of field "Mail_list" into nLine
  put trunc(wHeight / tHeight) into nView
  if nLine > nView
  then
    get  tHeight*nLine-wHeight+4
    set the scroll of field "Mail_list" to it
  end if
  --
  hide message window
  set the cursor to 1
end mouseUp

on cardMake
  global makeCard
  if not makeCard then
    doMenu "New Card"
    put true into makeCard
  end if
end cardMake

*** bkgnd button "sort" ***
*** Location: 405,48 ***
*** Rect: 382,40,428,57 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  global loopStart
  global indx_data
  set cursor to 4
  put "Sorting cards by date....Please wait few minits"
  sort text by field "Date"
  set cursor to 4
  set lockScreen to true
  put "We make up the table of contents...Please wait few minits."
  put empty into field mail_list"
  put empty into indx_data
  put 3 into loopStart
  -- Make Up Table
  makeTable
  set cursor to 1
  set lockScreen to false
  hide message window
end mouseUp
*** bkgnd button "makeup" ***
*** Location: 462,49 ***
*** Rect: 432,40,493,58 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  global loopStart
  global indx_data
  answer "Do you make a Mail-List?" with "Ok" or "Cancel"
  if it is "Cancel" then exit mouseUp
  set cursor to 4
  put "We make up the table of contents...Please wait few minits."
  set lockScreen to true
  put empty into indx_data
  put empty into field "mail_list"
  put 3 into loopStart
  -- Make Up Table
  makeTable
  get the number of cards
  put it -2 into field "numbCard"
  set the scroll of field "mail_list" to 0
  set cursor to 1
  set lockScreen to false
  hide message window
end mouseUp
*** bkgnd button "delete all" ***
*** Location: 62,320 ***
*** Rect: 25,313,100,327 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  Answer  "delete all ?" with "Delete"or "Cancel"
  if it = "Cancel"  then
    exit mouseUp
  else
    deleteAllCards
    put 0 into field "numbCard"
  end if
end mouseUp
*** bkgnd button "Prev" ***
*** Location: 220,318 ***
*** Rect: 207,308,234,328 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  visual effect scroll Down
  go to prev card
end mouseUp
*** Card =card "TitleCard" ***
on makeTable
  global loopStart
  global indx_data
  put the number of cards into last_count
  repeat with i=loopStart to  last_count
    put "Rest..... "&last_count -i+1
    put i-2 after indx_data
    get field "Subject" of card i
    delete last character of it
    put " "&it&" (" after indx_data
    get field "From" of card i
    if last char of it = return then delete last char of it
    put it & ")" after indx_data
    --    get field "Date" of card i
    --    if last char of it  = return then delete last character
of it
    --    put " <"& it & ">"  after indx_data
    put return after indx_data
  end repeat
  put indx_data into field "Mail_list"
end makeTable
*** card field "About" ***
*** Location: 243,169 ***
*** Rect: 117,95,370,243 ***
*** Visible: false ***
*** Style: rectangle ***
*** Script: ***
on mouseUp
  set the visible of me to not the visible of me
end mouseUp
*** card button "Mail of junet" ***
*** Location: 241,48 ***
*** Rect: 160,38,322,58 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  set the visible of card field "About" to ¬
  not the visible of card field "About"
end mouseUp
*** card button "Finder" ***
*** Location: 427,311 ***
*** Rect: 410,295,445,328 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  doMenu "Quit HyperCard"
end mouseUp
*** card button "汎用ボタン" ***
*** Location: 311,313 ***
*** Rect: 271,303,352,324 ***
```

```
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  -- set the cursor to 4
  -- repeat with i=3 to the number of cards
  --   get field "Date" of card i
  --   if it is not empty then
  --     put jDate(it) into field "Date" of card i
  --   end if
  -- end repeat
  -- set the cursor to 1
end mouseUp
```

患者照会の script

```
Script = ***入院患者照会***

on ListUp
  answer "一覧表を更新しますか？" with "いいえ" or "はい"
  if it is not "はい" then exit ListUp
  put empty into listData
  put the number of cards into cCount
  repeat with i = 6 to the number of cards
    put field "kanjaSimei" of card i after listData
    put ","& field "furigana" of card i after listData
    put ","& field "sex" of card i after listData
    put return after listData
    put "更新処理中.... 残り："&cCount-i
  end repeat
  put listData into field "pList" of card 4
  hide message box
end ListUp


*** Background = bkgnd id 2649 ***
*** Background = bkgnd id 3515 ***
*** bkgnd button "一覧表からの検索" ***
*** Location: 256,168 ***
*** Rect: 186,157,326,180 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  visual effect iris open slowly
  go to card id 4438
end mouseUp

*** bkgnd button "氏名で検索" ***
*** Location: 256,201 ***
*** Rect: 186,190,326,212 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  ask "検索する名前（部分）を入力してください。" with "なまえ"
  go to card 4
  put "find "&quote&it&quote
  hide message box
  type return
end mouseUp
*** bkgnd button "終了" ***
*** Location: 379,284 ***
*** Rect: 355,274,404,295 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  domenu "Quit HyperCard"
end mouseUp
*** bkgnd button "新規入院登録" ***
*** Location: 256,235 ***
*** Rect: 186,224,326,246 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  global upToDate
  put false into upToDate
  answer "新しく入院患者の情報を登録しますか？" with¬
  "いいえ" or  "はい"
  if it is not "はい" then exit mouseUp
  set lockScreen to true
  go to last card
  domenu "New Card"
  set lockScreen to false
  put true into upToDate
  get the rect of field "furigana"
  click at item 1 of it+1, item 2 of it+1
end mouseUp
*** bkgnd button "病院情報更新" ***
*** Location: 257,283 ***
*** Rect: 187,272,327,294 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  visual effect iris open
  go to card id 6197
end mouseUp

*** Background = bkgnd id 5940 ***
*** bkgnd button "戻る" ***
*** Location: 373,48 ***
*** Rect: 329,37,418,59 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  visual effect iris close slowly
  go to card id 3638
end mouseUp

*** Background = bkgnd id 4238 ***
*** bkgnd field "pList" ***
*** Location: 228,191 ***
*** Rect: 85,48,371,335 ***
*** Visible: true ***
*** Style: scrolling ***
*** Script: ***
on mouseUp
  put "pList" into name_fld
  put item 2 of the clickLoc into vPos
  put item 2 of the rect of field name_fld into wPos
  put the textHeight of field name_fld into tHeight
  put the scroll of field name_fld into sPos
  put trunc((vPos-wPos+sPos)/tHeight)+1 into nLine
  get  line nLine of field name_fld
  if it is empty then exit mouseUp
  put character 1 to (offset(",",it)- 1) of it into it
  put it&"...検索中"
  set lockScreen to true
  go to card 6
  find it
  visual effect iris open
  set lockScreen to false
  hide message window

end mouseUp
*** bkgnd button "一覧表更新" ***
```

```
*** Location: 414,59 ***
*** Rect: 376,48,453,70 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  send "ListUp"
end mouseUp
*** bkgnd button "戻る" ***
*** Location: 414,87 ***
*** Rect: 376,77,453,98 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  visual effect iris close slowly
  go to card id 3638
end mouseUp

*** Background = bkgnd id 4926 ***
on closeCard
  global upToDate
  if upToDate is true then
    send "ListUp"
    put false into upToDate
  end if

  pass closeCard
end closeCard
*** bkgnd button "nengoSet" ***
*** Location: 192,133 ***
*** Rect: 183,124,202,143 ***
*** Visible: true ***
*** Style: radioButton ***
*** Script: ***
on mouseDown
  put PopUp("平成;昭和;大正;明治",2,the mouseh,the mousev)¬
  into selNo
  if selNo>0 then
    put word selNo of "平成 昭和 大正 明治" into field "nengo"
  end if
end mouseDown
*** bkgnd button "sexSet" ***
*** Location: 192,154 ***
*** Rect: 183,145,202,164 ***
*** Visible: true ***
*** Style: radioButton ***
*** Script: ***
on mouseDown
  put PopUp("男;女",1,the mouseh,the mousev) into selNo
  if selNo>0 then
    put word selNo of "男 女" into field "sex"
  end if
end mouseDown
*** bkgnd button "kaSet" ***
*** Location: 194,189 ***
*** Rect: 185,180,204,199 ***
*** Visible: true ***
*** Style: radioButton ***
*** Script: ***
on mouseDown
  put empty into popList
  put field "jusinkaList" of card 3 into jusinkaList
  repeat with i = 1 to the number of lines of jusinkaList
    put line i of jusinkaList after popList
    put ";" after popList
  end repeat

  put PopUp(popList,1,the mouseh,the mousev) into selNo
  if selNo>0 then
    put line selNo of jusinkaList into field "jusinka"
  end if
end mouseDown
*** bkgnd button "byoutouSet" ***
*** Location: 194,212 ***
*** Rect: 185,203,204,222 ***
*** Visible: true ***
*** Style: radioButton ***
*** Script: ***
on mouseDown
  put empty into popList
  put field "byoutouList" of card 3 into byoutouList
  repeat with i = 1 to the number of lines of byoutouList
    put line i of byoutouList after popList
    put ";" after popList
  end repeat

  put PopUp(popList,1,the mouseh,the mousev) into selNo
  if selNo>0 then
    put line selNo of byoutouList into field "byoutou"
  end if
end mouseDown
*** bkgnd button "患者情報削除" ***
*** Location: 95,325 ***
*** Rect: 49,314,142,336 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  answer "この患者の情報を削除しますか？" with "はい" or "いいえ"
  if it is "はい" then domenu "Delete Card"
  go to card "menuCard"
  send "ListUp"
end mouseUp
*** bkgnd button "戻る" ***
*** Location: 404,52 ***
*** Rect: 361,41,447,63 ***
*** Visible: true ***
*** Style: roundRect ***
*** Script: ***
on mouseUp
  visual effect iris close slowly
  go to card "menuCard"
end mouseUp
*** bkgnd button "前へ" ***
*** Location: 440,326 ***
*** Rect: 431,318,449,335 ***
*** Visible: true ***
*** Style: transparent ***
*** Script: ***
on mouseUp
  if the number of this card <6 then
    visual effect scroll left
    go to last card
  else
    visual effect scroll right
    go to preview card
  end if
end mouseUp
*** Card -card "titleCard" ***
on openStack
  wait 2 seconds
  visual effect dissolve very slowly
  go to card "menuCard"
  pass openStack
end openStack

on mouseUp
  visual effect dissolve very slowly
  go to next card
end mouseUp
```

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

ポジション・ペーパ

篠田 陽一（東京工業大学）

■ 自己紹介

浪人中に出会った Apple][から計算機に入り、学部の時代にはカード入力の M160 で情報処理の基礎教育を受ける。この間、出入りしていた研究室で、ACOS,NEAC3100 等も触り、NEAC のイニシャルオーダをパチパチで入れたり、ACOS のマイクロプログラムを組んだりする。修士課程に進学し、地道 (?) にプログラマとしての道を歩きはじめ、機器制御マイクロプロセッサのアセンブラプログラミング、オフコンのユーティリティー、プロセス制御、メインフレームのデータベースシステムなどを手掛けて今日に至る。

■ 一言

ここ一年程度、気にかけているキーワードがある。それは、"Product Oriented Integration" あるいは "Artifact Oriented Integration" であり、要するに「生成物」にもっとそれなりの地位を与え、それに従って環境を構築していこうという考えかたである。

もちろん、「生成物」といっても、最終的なコードやなんちゃら仕様書といった「表物」に限らず、アイデアや設計メモ、障害のデータなどの「裏物」も含んでいる。表物、裏物に限らず、これらの生成物の重要さはみんなが認識しているにも関わらず、どうも扱いが「軽い」ような印象を受ける。

今回のワークショップでは、「生成物」の扱いを念頭においた議論をして行きたいと思っている。

一応、簡単に各セッションにおける立場を記しておくと:

□ MML の確立を目指して

> *'It's a poor sort of memory that only works backwards,' the Queen remarked.* Lewis Carrol 1832-1898

前回の神戸以降、Big Blue からもイーサネットの native ハードウェアが発表になったりして、なんとなく甲板はできたという感じだが、デッキクルーの充実がまだまだ問題である。妙なアナロジーはさておき、mainframe-in-network 記述 (確かこう呼ぶことにしたはず) の確立は、

- データトランスペアレンシーの確立 (ヘテロ環境でのデータ互換。アップロード・ダウンロードは、もはや差別用語。)
- メインフレーム側からの concession (実は某国家プロジェクトの失敗の原因は、ここにもあると思っているのだった。)
- RPC の活用による開発環境の構築

がキーになると思う。まあ、これは「良い子」の意見であって、過激派の青二才の書生談義の立場からいうと、いまだに「メインフレーム捨ててしまえ」である。Big Blue が現在のアーキテクチャを捨てる決意をした今、ミイラ化した計算機と心中するメインフレーマの姿と、訳もわからずに路頭に迷うプログラマ達の哀愁あふれる後姿が早く見たくてたまらない今日この頃である。

Keyword:「ちょっとばっかビビらせてやんねえとな。」

□ HyperText の光と影

'Solution may be Hypertext + Groupware.', R.B. said.
I thought *'Does it mean anything ?'*

HyperText には

- 発想を補助する
- 物の整理のしかた、参照のしかたの手段を与える

の二つの機能があることは、誰もが同意するだろう。ところが、前者は混沌を表し、後者は秩序を表す。まさに両刀の剣、一つのメタフォアがこれほど両極端の顔をしていることは少ない。それだけに、使い方を間違えると悲惨だろう。必要なのは混沌と秩序を見分ける明せきな頭脳と混沌から秩序を生み出すことによってマネージャを喜ばせる神の手である。

□ ソフトウェア工学は現場に生かせるか?

A man cannot be too careful in the choice of his enemies
Oscar Wilde 1854-1900

こればっかしはわからん。

Keyword:「甘い汁ばかり吸おうとしてもだめさ。」

□ 企業間ネットワークの構築と活用

Let's your communication be Yea, yea; Nay, nay.
New Testament, St. Matthew.
Let's your communication be yea Yea YEa YEA!
Yoichi Shinoda 1959-

使ったことのない人にいくら言ってもだめさ。
禁断の果実の味を知っている人だけで楽しみましょう。

Keyword:「どんどん良く鳴る法華の太鼓」

□ ツール・インテグレーションの理想と現実

ツール・インテグレーションを妨げている要因の大きなものに、

- オブジェクト・トランスペアレンシーの欠如
- ユーザインターフェース・トランスペアレンシーの欠如

を考えている。前者は、ソフトウェアオブジェクトのインターフェースの、後者は、ツール間でのユーザインターフェースの不統一によるものである。ツール・インテグレーションは、ユーザが全く気がつかない内に行なわれ、その効力を発揮することが望ましい。

Keyword:「見のがしてくれよ」

Seesion 2: HyperText の光と影 (Pre-Workshop Position Paper)

HyperText System の goto はなぜ自然か？

佐原 伸（SRA）

ハイパーメディアはなぜ必要か？

従来の情報システムは、「金融システム」のように紙と鉛筆をいかに効率よく模倣するか、あるいはミサイル制御システムのような巨大システムをいかに制御するかに重きをおいていた。紙と鉛筆で育った古い世代や、大きな組織を支援するためのものであった。

しかし、コンピュータの小型化・個人化・ネットワーク化・マルチメディア化でにより、この流れが変わりつつある。ビデオで育った若い人々が、ネットワークでつながりはじめた。この世界では、テキストだけでなく、絵や音楽や映像が情報システムの対象になってくる。

もともと、「情報」は紙と鉛筆では十分に表現することは出来ない。例えば、原住アメリカ人(アメリカ=インディアン)や原住日本人(アイヌ)の言葉を表す文字がないのと同様、多くの言葉は元々文字がなかった。日本語が中国文字を借りてきていることでも分かる。音楽を「楽譜」で鑑賞できる人も少ないだろう。

しかし、現在のコンピュータのアーキテクチャ、特にソフトウェアのアーキテクチャはマルチ=メディアを考慮していなかったため、このまま現在の流れを発展させていっても、本当の意味のマルチメディア化はできないだろう。例えば、今のコンピュータで「音」をサンプリングして格納すると、あっというまに数メガバイトの空間を占拠してしまう。

そこで、ハイパーメディアの研究・開発が必要になる。特にソフトウェアからのアプローチが重要になると思う。Apple社が発売したHyperCardは、素晴らしいソフトウェアであるが、まだまだハイパーメディア=ワールドの入口あたりに位置しているに過ぎないだろう。

ハイパーメディアで重要なこと

ハイパーメディアの世界では、ユーザーが「プログラミング」することが重要であると思う。分かりやすいユーザーインタフェースのためには「ユーザーにプログラミングさせるな」という説が一般的だが、どうもこれはおかしい。もちろん、従来のようなテキストでプログラムを作らせるのは論外であるが、ロボットに命令するといった形式で「プログラム」を作るのはさほど不自然ではない。現実に、我家の子供達にソフトウェアを使わせてみると、全てを自動的にやってしまうソフトウェアではなく、自分達で自由に操れるソフトウェアを夢中になる。LogoやSmalltalkを子供達に使わせて実験したMITやPARCの実験結果も、同じだったと聞いている。

次に重要なことは「非構造化」だろう。人間の頭はランダムに乱雑に並行的に物事を考えている(私だけそうだったらどうしよう:-)。テキストに書く場合は、これを構造化して紙の上に表現しなければならない。そこでKJ法などといった手法が出現することになる。しかし、考えてみれば「構造的でない」考えを「構造化」した表現にマップすると、重要な「何か」が漏れて記録されてしまうことになる。手書きの汚いメモの方が、清書された議事録より役立つといった経験は誰にでもある。それならば、いっそのこと「非構造化」された情報をそのまま蓄積したほうが良いのではないか？

ハイパーテキストはなぜ必要か？

そこでハイパーテキストの登場となる。ハイパーテキストは、「非構造化」された考えを次々と格納し、それを整理して探しだすのに適している。最初から、情報を格納する「構造化された」枠組みを考えるのではなく、「入れてから考える」方式であると言ってもよいだろう

ハイパーテキストの特徴

ハイパーテキストの特徴をテキスト:-)で示すと以下のようになる。

・テキストまたは図形あるいはその他の情報の頂点(ノード) のネットワーク
・頂点の視点（ビュー）としていくつかのウィンドウが、画面上に表示される。
・ウィンドウは移動したり、サイズを変更したり、閉じてアイコンにすることができる。
・ウィンドウのアイコンを指定して、ウィンドウを開きその中の情報を操作することができる。この操作によって頂点の中の情報を変えることができる。
・ウィンドウはいくつかのリンクアイコンを持つことができ、それらはデータベース中の他の頂点へのポインタを表す。
・リンクアイコンは何を指し示しているかを表すテキストフィールドを持ち、このアイコンをクリックすることにより、データベース中の他の頂点へ移動し、その内容をウィンドウに表示することができる。
・ユーザーは新しいリンクや頂点を付け加えたり、内容を更新したり、削除することができる。
・リンクをたどって情報を検索できる。
・文字列やキーワードあるいは属性による検索が行なえる。
・図形的に表現したブラウザによる操作が行なえる。
・階層構造を持つことも可能である。
・頂点やリンクに属性や値を持つことができる。
・頂点やリンクが複数の版を持つことができる。
・頂点やリンクに手続きを付加することができる。
・頂点の内容を変更するエディタがある。
・複数ユーザーが同時に頂点の内容を編集できる。

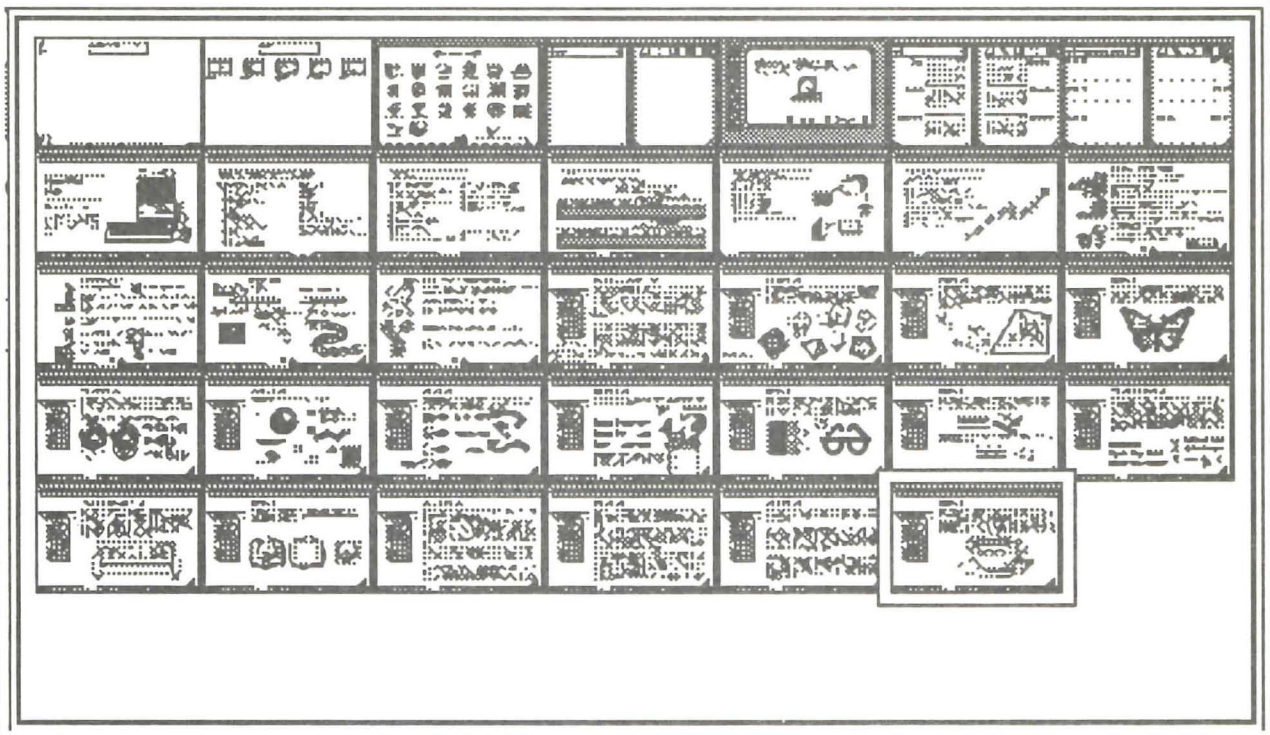
go to はなぜ自然か？

ハイパーテキストシステムで、go to が復活したことを嘆く声がある。しかし、上に示したハイパーテキストの特徴でも分かるように、ハイパーテキスト＝システムではgo toでジャンプしても「自動的に」戻ってくる手段が提供されている。「行ったきり」のプログラム言語の場合とは異なるのだ。

また、人間がある作業をしているとき、それを連続して行なうことは少ない。「ながら族」の様に、テレビを見ながら電話し、電話が終わるとテレビを聞きながら新聞を読み、かつポジション＝ペーパーの内容を考えるといった具合である。この間の作業の切り替えは、いってみれば go toに当たる。

つまり、人間の作業ではgo toを伴うことの方が自然なのだ。問題はgo toではなく、「行ったきり」で戻って来ないことなのだ。また、go to を使う必要がないところでgo to を使うことが問題なのだ。HyperCardで作業しているとこのあたりのことが実感できる。以下に、HyperCardで「戻る」ための仕掛けの一つであるミニ=ピクチャー画面を示す。

図1　ミニ=ピクチャー



ハイパーメディアにはオブジェクト指向がよく似合う。

ハイパーメディアを実現しようとすると、テキストや音や絵や映像を同じ立場で扱わなければならない。このためには、オブジェクト指向の考え方が良く似合う。「ロボットに命令する」という前に述べたパラダイムにぴったりするのだ。オフィスでコピーやFAXを人に頼むのも、これと同じだろう。決してコピーやFAXが面倒なのではなく、コピーの仕方やFAXの仕方が分からないから人に頼むのである:-)まさに、オブジェクト指向の真髄である。

おわりに

と、まあ、以上のような議論の曖昧なところを討論したく、参加する次第です。

HyperText の光と影

セッション・サマリ

熊谷 章
(PFU & 富士通)

1. はじめに

本レポートは，HyperText セッションをまとめたものである．司会は熊谷がつとめ，話題提供を次の方々にお願いした：

加藤 康人 (PFU): PWB と HyperBook
高橋 肇 (JIP): ソフトウェア開発において HyperText 技術をどのように利用すべきか？
野村 行憲 (ICS): HyperCard はソフトウェア開発ツールになりうるか？
高橋 晃 (釧路高専): HyperText のキー技術
篠田 陽一 (東工大): HyperText の Human-Interaction-Model 限界説
佐原 伸 (SRA): Chaos は Chaos のまま表現した方がよい
新田 稔 (SRA): あなたは Hyper Realism に蝕まれていないか？

2. 発　表

各自の発表を司会者の立場から勝手にパラフレイズすれば次のようになる．

加藤康人氏の説明は，Hypermedia 技術をベースとして開発した HyperBook と，その上で動作するプロトタイパー向けのワークベンチ，PWB に関するものであった．要求定義や要求分析に HyperBook を使用するためには，(1) 複合シート，(2) バージョン・コントロール，(3) いろいろなメディアのシート，が必要であるという主張であった．

高橋肇氏の発表は，HyperCard をソフトウェア開発においてどの部分に使用できるかがテーマであった．かれの主張は，エキスパート・システム開発時の知識獲得のような試行錯誤を必要とする場面では，HyperText 技術が威力を発揮するという．具体的には，カードのリンク構造を階層的にビジュアルに表現できるので，思考対象の制約，全体と部分の関係の理解などが容易にできるということであった．

野村行憲氏は，HyperCard と HyperTalk を使って mail を整理するスタックウェアの開発事例を紹介した．僅か８時間でこのスタックウェアができたらしい．かれの主張では，HyperCard と HyperTalk で作成したスタックウェアには２つの欠点がある．１つは，大量データ操作や文字列操作における処理速度の遅さである．そして第２は，作成したスタックウェアが HyperCard 上でしか動作できないことである．

高橋晃氏は，御当地釧路からの唯一の参加者である．かれは，HyperText は User Interface と Object System の問題である，という主張を提起した．かれ自身は，いま Object Oriented な環境でのブラウザ等のツール群や並列分散環境でのデバッギング・ツールをどう構築するかといった問題を Hyper Text で解決しようとしている，という．それで，今後は並行プログラミングの新しい計算モデルや認知科学からの成果を採り入れることが必要だ，と説明した．

篠田陽一氏は，HyperText には，(1) 発想を補助する，(2) 物の整理の仕方，参照の仕方の手段を与える，といった機能を持っているが，しかし，(1) は混沌を表わし，(2) は秩序を表わすので，正に両刀の剣としての危険性を秘めていると指摘した．必要なことは，混沌と秩序を見分ける明晰な頭脳と混沌から秩序を生み出す神の手である．絵と文字が一緒に表現できるといった余り他愛のないことではしゃいではいけない，というのもかれの持論である．

次は，佐原伸氏．かれは，元来，情報は紙と鉛筆では十分に表現できないという．たとえば，音楽と音譜，原住アメリカ人の言葉を表わす言語の不在などが考えられる．マルチメディア・コンピュータと HyperMedia によって，この紙と鉛筆の限界が解決される．HyperMedia で重要なことは，ユーザが自分でプログラミングすることと非構造的なアイデアをそのまま表現できることである．人間の頭は(少なくとも佐原氏の頭の構造は)，ランダムで乱雑で並行的に物事を考えている．それを無理矢理構造化された形にマッピングして記録すると，重要

な何かが漏れてしまう．最初から情報として構造化された枠組として考えるのでなく，入れてから考えるといったHyperMediaが重要である．HyperTextの出現でgoto文が復活したと嘆く向きが多いが，そもそも人間の作業では連続したものが少ないのでgotoが自然である．問題はgotoではなく，行ったきりで戻ってこないことなのだ．戻ってくる機構がHyperTextにあればよい．また，HyperMediaではテキスト，音，絵，映像などのメディアを同じレベルと立場で扱う必要がある．これは，まさにオブジェクト指向の真骨頂だというのが，かれの主張であった．

新田稔氏の主張は，スーパーリアリズムに対する警告である．われわれは，コンピュータの中で起きる非現実的な事柄を，現実世界の何かにたとえ手理解してきた．ところが，コンピュータを使ったHyper*では，

距離 ---> 1にできる
重さ ---> 一定（ゼロ？）
速さ ---> 一定（瞬時）

となる．すると，現実界に対応するものがなくなるので，人間のとる態度は次の4種類に分かれる．

A) よーく考えて理解する型： 超次元時空間
B) 何も考えずに受け入れる型： なんじゃらほい
O) そんなものは認めない型： 現実に合うように変える
AB) ちょっと離れて批評する型： ？

HyperMediaで重要なことは，一つのノードから出るリンクの数を制限する，必ず帰りのリンクをもつ，ものはそれ自身の内容よりどことつながっているかで価値が判断されるようになったことである，と説いた．

司会である熊谷は，HyperMediaの定義，そこにおけるデータモデルの問題，ユーザインタフェースの問題，協調作業におけるHyperMediaの問題，HyperMediaの今後の課題について説明した．その中で，特にデータモデルとDirect Manipulationが重要であると主張した．そして，今後ソフトウェアシステムの複雑さは半永久的に増進する．しかし，HyperMediaを使ってその複雑なシステムのための標準を研究開発することができる．そして，そのことによってDesktop Metaphorを革新できると主張した．

3. 討　論

まず，Desktop Metaphorに関して議論した．マルチメディアのコンピュータ・システムを使用することによってわれわれのDesktop Metaphorは確実に変革するという意見とHyperMediaのようなChaoticなものを持ち込んでもそれは混沌を増すばかりでDesktop Metaphorにはならない，という意見に分かれた．Cosmosが光であり，Chaosが闇である，という主張があった．実際は，これらの両者の間にChaotic CosmosまたはCosmic Chaosがありこれが中庸ではないか，ということになった．

次にHyperMedia systemの作成の問題に入り，オブジェクト指向プログラミング環境であるSmalltalkの話題になった．ここで，青木淳氏からこれから先のSmalltalkやObjectworksの説明があった．その話によれば，Objectworksはほとんどの WSと PCで動作するようになる．また，これからの方向としては，分散ネットワークシステムになり，いろいろなノードからオブジェクトを共有して使用できるようになるという．それの方向の第一歩として，データベースとの結合という観点からObjectworksとデータベースシステムOracleとの結合をやっているという．圧倒的多数の意見で，これからはSmalltalkを使ってこれを広範に広めようということになった．

HyperText の光と影

セッションの討論を聞いて

魚田: HyperText では，一度 Link 関係をつけたものを捨てる(別の発想で作り直す)ことが容易であると一般にはいわれているが，ほんとうにそうだろうか？ 新たに作る場合はたしかにやさしいと思うが..... HyperText について知識がないため，このあたりがよくわからない．

久保: Object Space を対象化するためには Data Model と計算モデルがいる．モデルを視覚的に表現する手段の１つとして HyperText があると理解すればよいのだろうか．すっきりはしないが，HyperText 派のちゃらちゃらした話を冷やかに聞くと，自信が持てそうである．篠田さんに感謝．

野村(敏): Hyper-XX は単なるツールの一種であり，過剰な期待をしてはならないだろう．篠田氏のいう「浮かれてはいけない」という発言は理解できる(表面的議論を避けるべきということ)．

HyperText が特殊なものであり，一部の好き者たちがオモチャにする対象だと思われてしまっているようだ(？)．このことからすると，Hyper-XX のブームは一過性ではかなく消えていくのではないかという予感がする．少々趣味の世界に走りすぎてしまったのではないだろうか？ 趣味と有用性（あるいは実用性)が一致することが幸せではあるが，現実的には後者を重視した議論が必要ではないか？

酒匂: HyperText の参照システムは数多く存在するが，Public から Private を作る仕掛けが欲しい．

新田: Hyper-XX はデバッグもテストも困難である．したがって，Hyper-XX でアプリケーションは作らない方がよい．Hyper-XX は（データ，情報，知識）の蓄積に使えばよい．

青木: 難しいセッションでした．なんだか私は Smalltalk-80 の説明をさせられたという感じです（Smalltalk-80 はあくまでも１つの手段にすぎません！）．Hyper-XX は OODB のプロジェクションでよいのでは？（Hyper-XX はあくまでも OODB の投影の１つの手段です！）

濱田: 発想の補助について一言．商売になるアイデアは大衆を喜ばすものであり，決して高級なものでなくてもよい．HyperText の中でブレーンストーミングすると面白いのではないか？

中野: PFU の加藤さんの話は，いま研究室でも電子本を考えているので，非常に興味があった．やはりこれはデモが必要でしょう．Data Model の話は興味がないのでパスします．SmallTalk-80 についての議論は，現在この線で HyperText/Media の研究をし始めているので賛成である．現在ある優れた環境の１なので，これからも利用(活用)していきたい．Hyper-XX はこれから実際のボリュームのあるシステムが出てきてから，本当に議論できるのではと思っています．

林: Hyper-XX はまだ議論のテーブルに乗せるには早すぎる(？)

1: 経験者（利用者）が少なすぎる．

2: 具体化されたものがチャチすぎる．

ということで，普及に努めようとする熊谷さんの提案程度のところでしょう（論点がぐちゃぐちゃ)．なかなかに B 型的進行でした（何かを産むかも知れないが，他には何もないかな？)．

野村（行): HyperText とは直接関係ないが，最近の SmallTalk は魅力がある（HyperText のアプリケーション構築の道具として....)．

加藤: HyperText が情報の表現手段･参照･格納の手段であるということはわかった．こういった手段を使ったプロセス(手順)，たとえばどうやってその情報にアクセスしたか，なども HyperText として捉えるべきか！

深瀬: HyperText + AI Flame + GroupWare が今後の方向でしょう．Object Base については，今後 2 ～ 3 年で OODB が商用化されるので，むしろ Authoring System についての問題がクローズアップされるでしょう．

佐原: Smalltalk で自分なりのソフトウェア開発支援 HyperText システムを作る決意をした．

熊谷: 実に楽しいセッションであった．SmallTalk-80 の夜明けを作る思いであった．ここ 5 年間の私の集中が，今ここに青木淳氏の説明で実現できる途ができてきた．これからだ．全てはこれから始まる．この記念すべき第一歩を釧路で歩めたことは記念すべきである．「かの国の涯にて，オブジェクト指向の思いを遂げたり．皆の顔に喜びの涙流れけり」

岸田: 光: OOP の大衆版としての HyperText の未来は明るいか？ 影: Data Modeling は，ただの Metaphor でよいのか？

岡本: Hyper-XX を話題として取り上げて，Hyper-XX をやっている人たちの話に耳を傾けると，ついつい Hyper-XX って万能薬であるかのように話を聞いてしまう．ところが，実際に Hyper-XX が活用されて「ソフトウェアの開発がこう変わった」とか，「ソフトウェアの開発をこう支援しています」といった実例にまだ出会ったことがありません．悲しいですね．だから，Hyper-XX って好きになれないのかな？ たしかに使ってみて面白いことは事実なんですけど....

高橋: 情報を情報としてだけ捉えるならば，Hyper-XX の機能は興味深いが，それ以上のことはあまり現状は追求したくない．これからの展開は別であるが....

Seesion 2: HyperText の光と影 (Post-Workshop Position Paper)

ワークショップを終えて

加藤 康人 (PFU)

今回が初めての SEA 環境ワークショップへの参加であったため議論の流れが見えないところも多々あったが，次のようなことを感じた．

革命 (Revolution) か，進化 (Evolution) か？

MML のセッションでも話しがあったが，ワークステーションのパワーが増大するにつれて，メインフレームの得意とする高速演算，データベースといった使われ方の有難味が減少し，ワークステーションがそれにとって代わることが可能になる．そうすると MML ということ自体があまり意味をなさなくなり，メインフレームではこれまでの資産（と呼べるかどうか？）をどう生かして行くか，という話題につきることになる．

また，Tool Integration セッションでも，やはりどういった Tool の Integration を考えて行くのか？ つまり，これまでの Tool を生かす形での Integration なのか？ それとも，これから現われる Tool なのか？ ということが話題になった．

現存するものを捨てて新しい環境を創り出していくことを革命とし，現存するものをうまく生かすような環境を構築していくことを進化とするならば，どちらの道を選択するのか？

私としては，革命を起こすことのほうがより現実的であるように思う．ソフトウェアが膨大になり，多様化してきた今日，なるべく過去のしがらみから離れたところ（これを見極めるのも重要か？）で，いままで研究されてきたソフトウェアの開発プロセスに対する方法論やツール，オブジェクト管理といったソフトウェア工学の成果を現場に生かすよう，充分な体力（必要なツールはどんどん作ってしまうだけのパワー）を養うことが重要になる．幸いにして Smalltalk という環境がこれらのことを支援してくれそうな感触を得ることもできた．

また，これからのソフトウェアの開発においては，チーム共同作業やプロジェクト管理がますます重要な位置を占めてくると思う．そこで必要になるのが，オブジェクト管理とコミュニケーションのための Hypermedia ではないだろうか？ 私個人としては Smalltalk という環境でソフトウェアの開発を行う場合に，そのプロセスはどうなっていてそれを支援するためのツールにはどんなもが必要(有用)か？ それに Hypermedia をうまく使うことができるか？ といった問題に興味が湧いてきた．

いったん革命が起こると，そのインパクトが強ければ強いほど，現状のしがらみから抜け出たいという動機づけは増えるだろう．この時に，現存の資産からの歩み寄りを可能にするための流れをつけておく必要があるかも知れない(これが何らかの標準化ということか？)．最初は細く遅い流れでも，一旦流れ出せば怒涛のような流れとなるであろう．

Seesion 2: HyperText の光と影 (Post-Workshop Position Paper)

ワークショップを終えて

高橋　晃（釧路工業高等専門学校）

(1) 現状

Hyper-XX というのは１つの流行語というイメージが強い．

実用的なハイパーテキストシステムは LISP マシンや Smalltalk 等のブラウジングシステムに，あるいは Mathmatica 等のアプリケーションに埋めこまれている．ただし，これらは，テキストのリンクの種類や数をダイナミックに変えられないものが多い．

ハイパーテキスト自体を売り物にしているソフトで最も成功しているのは，Macintosh のハイパーカードであろう．オブジェクトをカード，フィールド，ボタンに制限し，さらに，HyperTalk という言語によってユーザがカスタマイス可能になった．しかし，カードのサイズに代表されるように制限は多い．

いずれにしても現存するハイパーテキストシステムの機能はまだ低いと考えられる．これらの最低限の共通点をあげれば，

見かけは，ボタン（アイコン）を押すことでアクションを起こす．
インプリメントは，既存の（ウィンドウ）システム上にまちまちに実現．

ということであろう．

(2) 問題点

ユーザインタフェイスの人間工学的な見地からの考察が十分ではない．つまり，アイコンの妥当性（よいアイコン，悪いアイコンの評価）の基準がない．また，アクションをアイコンから予測することは難しい（悪用も出来る）．

ハイパーテキストシステムは，システム依存性が強い．

(3) 提案

アイコンデザインやボタンアクションのガイドラインが必要である： ウィンドウシステムの標準の GUI のガイドラインを制定しようという動きがあるが，ハイパーテキストのためには，さらにアイコンやボタンについてのガイドラインを作成する必要がある．

ネットワークワイドのハイパーテキストシステム構築には並列オブジェクト指向の基礎的な研究か必要である： Macintosh でハイパーカードが成功したのは Macintosh オブジェクト指向プログラミングの下地が十分であったからである．ネットワークワイドのハイパーテキストの実現のためには，並列オブジェクト指向プログラミングのメカニズムを OS レベルから洗い直す必要であろう．

「必要であると」いうばかりでは無責任なので，自分でかかわれる範囲の話をすると....

並列オブジェクト指向モデルはいくつか提案されているが，北海道大学で研究されている Kamui モデルもその一つである．Kamui は場とイベントに基づく計算モデルで，その詳細については，ここでは述べないが，プロジェクトの一環として，ユーザやアプリケーションレベルで，ネットワークトランスペアレントな計算環境の構築をめざしている．Kamui-Kotan は Kamui-C の視覚的実行環境として開発中であるが，Kamui-Protocol 上の GUI として位置づけることができる．その上でのハイパーテキストシステムの試作も行いたいと考えている．

Session 3

ソフトウェア工学は現場に生かせるか？

(1989 年 11 月 30 日 14:00 − 17:00)

Chair 岸田 孝一 **(SRA)**

Speaker 落水 浩一郎 **(静岡大)**
佐原 伸 **(SRA)**
新田 稔 **(SRA)**
野村 敏次 **(JIP)**
野村 行憲（岩手電子計算センター）
濱田 勉 **(NTT 九州)**
平尾 一浩 **(HST)**

討論の狙い

ソフトウェア工学が生まれてからすでに２０年たち，その間にいろいろな技法やツールが提案されてきましたが，いまだにアプリケーション開発の現場には普及していません．最近では，ワークステーションの高性能化とともに，これらのツールが“CASE”という新包装を身にまとって再登場しつつありますが，はたして，その効用はどうでしょうか？「再利用」や「プロトタイピング」,「オブジェクト指向」などのキャッチフレーズも，やたらかけ声ばかり大きくて，実践がともなっていない感じがあります．こうした状況は，環境のハードウェア的側面が整備されれば，自然に改善されるのでしょうか？ それとも，もっとドラスティックな打開策が必要なのでしょうか？

このセッションでは，開発環境と人間（あるいは組織）との関係を話題の中心にすえて，ソフトウェア工学をアプリケーション開発の現場で実践することの今後を占ってみたいと考えます．

Session 3: ソフトウェア工学は現場に生かせるか？

目次

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ソフトウェア工学は小康を目指す？

岸田 孝一 (SRA)

1. 大同と小康のメタフォア

(1-1) かつて孔子がある国の祭礼に賓客として出席した．式典が終わったのち，城門上の楼観を散策しつつ，かれは何度か溜息をついた．かたわらにいた弟子が「何を嘆くのか？」とたずねた．孔子は次のようにいった：

(1-2)「むかし，大道が実践されていたとき，天下（世界）は公共のものであった．人々は賢者と能力あるものとを選び，かれらの言説は誠実で行為は親睦的であった．ゆえに人は自分の親のみを愛することなく，自分の子のみを慈しむことがなかった．老人は安楽に生きて死を迎え，壮年には仕事があり，幼いものは無事に成長し，妻を失ったものと夫を失ったもの，孤児と子なきもの，廃疾者もすべて十分に保護されたのである．男たちには仕事があり，女たちには家庭があった．財貨を粗末に扱ったり地に棄てたりすることを憎み，といって必ずしも自分のためだけに労働するのではなかった．したがって，謀略が閉ざされて行われることなく，窃盗や乱賊が起こることもなかった．ゆえに外の戸も閉じられることはなかった．これを＜大同＞という．

(1-3)「しかし，いまや大道は崩れ去った．天下（世界）は家族のものだと考えられ，人々はそれぞれ自分の親だけを愛し，自分の子だけを慈しんだ．財貨は自分のために蓄積され，労働は自分自身のために行われた．統治者（大人）はその地位を子孫に伝えるのが規則となった．城郭と溝池とが堅固に構築された．礼儀が社会の綱紀となって，君主と臣下との関係を正しくし，父と子との間柄を篤く，兄と弟を睦まじく，夫婦を和合させた．制度が設けられて，耕作地と集落が形成された．人々は勇気と知力とを尊び，功績はすべて自分のために挙げられたのである．ゆえに謀略が起こり，それによって戦争が起こった．夏の禹王，商の湯王，周の文王，武王，成王，周公はこうした社会において卓越していた．これら６人の君子は，礼を非常に尊重したのであって，正義を明らかにし，信実を行い，過誤を明白にし，仁愛に拠って謙譲を論じ，かくて民衆に不変の道徳律が存することを示した．もしこの道徳律に従わずに権勢を有するものがあれば，それを処罰してその地位から退けた．民衆がそれを禍としていたからである．これを＜小康＞という．

(1-4)「こうしたことは，すべて記録に残っている．しかし，私自身はまだこの目で＜大同＞も＜小康＞も見たことがない．それが悲しいのだ」と．

(1-5) そもそも Programming-in-the-Large への社会的需要がもたらした混乱状況の鎮静策として提唱されたソフトウェア工学が，孔子のいう＜小康＞的プロジェクト運営を目指していることはいうまでもない．すなわち，ソフトウェア・エンジニアリングの主たるドライビング・フォースは，われら自身の内なる＜小康＞への願望である．

(1-6) しかしまた一方で，われわれの心のすみのどこかに，かつて Programming-in-the-Small の世界のどこかで実現されていたにちがいない＜大同＞的コミュニティへのあこがれ（コンプレックス）が存在していることも，また否定できない．たとえば，その顕著なあらわれとして，いわゆる「チーフ・プログラマ・チーム」の概念などが挙げられよう．

(1-7)「人間は本来集団を作らなければ生きて行けない動物である．たとえ集団を作っても，そこに一定の秩序がなければ乱れ，乱れればお互いに争い，争えば分裂し，分裂すれば弱くなり，弱くなれば物に勝つことができない」（荀子）．

(1-8) 人間集団としてのソフトウェア開発組織に安定をもたらすためには，なんらか原理（荀子のいう＜分＞）によって，そこに整然とした階層的秩序を生みだし，＜小康＞を確立しなければならない．ソフトウェア工学に期待されたのは，そうした原理としての役割をはたすことであった．

(1-9)「名　正しからざれば，言　順（したが）わず，言　順わざれば，事　成らず，事　成らざれば，礼楽　興らず」．

(1-10) 論語・子路篇における孔子のこのことばは，開発環境をめぐる諸概念の階層関係を示したものだとして，

解釈することができよう．すなわち，階層の最上部には，正しいプロセス･モデル（名）があり，それにもとづいた開発方法論（言）が下層のレベルで確立されている．でなければ，プロジェクト（事）はうまく行かないし，そのための開発環境（礼楽）も構築できない．

(1-11) こうした概念の階層構造は，また，プロジェクトの組織構造にほぼそのままマッピングできる．したがって，われわれがもし，適切な礼楽（開発環境）の導入によって何らかの現状変革を成そうと思うならば，まず，最上階の概念レベル（つまり，対応する組織ピラミッドの最上層）において，「名を正さなければ（正しいプロセス･モデルを確立しなければ）」ならないのである．そして，そのさい注意すべきは，ふつう，開発組織自身はそうしたプロセス認識を改めるべき自浄力を持っていないということである．

(1-12) 紀元前数世紀，乱立する都市国家群において，新しい政治思想（プロセス･モデル）や統治メカニズム（環境）は，つねに城壁の外から，すなわち，おのれの舌だけを頼りにそれらの都市の間をさまよい歩いていたがゆえに「東西南北の人」と呼ばれた思想家たちによってもたらされた．ソフトウェア工学の世界においても，同様に，新しい技術革新はふつう，日常の仕事に追いまくられ，自己改革能力をうしないかけている組織の外部から，パラダイム革命のかたちでもたらされるのである．

(1-13) 上述の階層構造は，常識的な（体制側の）の立場からすればきわめて**快適**であるが，しかし，現状変革の立場に立つといささか**不快**である．

それはなぜか？

現状のプロセスを肯定したままで，たとえば CASE ツール等を導入したとしても，結果は，かなり小幅な改善しかもたらされない．根本的な事態の改善には，プロセスの大幅な変更を必要とする．真の意味での New Technology とは，その導入によって現状プロセスを**破壊（または革新）**するようなものである．

2. ソフトウェア･プロセスの本質

(2-1) 考察すべきいくつかのプロセスがある．

(a) ユーザ･プロセス
(b) ソフトウェア進化プロセス
(c) 開発プロセス

(2-2) われわれが開発しているソフトウェアは，ユーザの仕事を助けるツールまたは作業環境にすぎない．ソフトウェアは，ユーザのニーズになるべくあてはまるように，一連の学習およびコミュニケーションの**プロセス**を通じて，設計され具体化されて行く．ソフトウェアの適切さは，ある期間の試用および改訂の**プロセス**を踏んで，ようやく確立される．

(2-3) ところで，ユーザ･プロセスは，現実社会のなかに生きているプロセスであり，たえず変化しつづけている．したがって，そこに組み込まれた（Embed された）ソフトウェアは，宿命的にたえず進化しつづけなければならない．

(2-4) そうしたソフトウェア進化へのニーズは，ソフトウェアの開発が完了し，それがユーザ･システムに組み込まれてから初めて起こるというわけではない．むしろ，開発途中の時点でいつでも起こりうる．すなわち，開発プロセスはソフトウェア進化プロセスのなかに組み込まれており，そしてまた，後者はすでに述べたように，ユーザ･プロセスのなかに組み込まれている．

(2-5) この複雑な入れ子構造が，ソフトウェア･プロセスの宿命である．

(2-6) 古典的なウォータフォール･モデルの致命的な欠陥は，こうしたプロセスの本質を見誤り，ハードウェア製造プロセスの場合と同様，開発と保守（進化）を分離し，仕様化（設計）と製造（具体化）を分離できると考えたことにある．

(2-7) システムとは，１人あるいは１群の人間が「ある期間，何らかの理由によって，一定の関心を持って」切り出した世界の１部分である．システムは，いくつかの構成要素から成り，各構成要素はそれぞれの特性によってお互いに区別され，またこれらの特性を操作するための動作が定義されている．

(2-8) ＜対象システム＞とは，われわれがプログラムを開発するために切り出した現実世界の１部分を指す．そして，＜ソフトウェア･システム＞とは，われわれが開発したプログラムの集合とそのインタフェイスである．

(2-9) コンピュータ応用の初期の時代には，機械の処理能力が貧弱だったために，対象システムは，「ソフトウェア・システムの開発プロセスが円滑に進むように」という観点に立って，選択された．対象システムは，本質的に静的なものとしてとらえられ，ソフトウェア・システムの導入「以前」および「以後」という２つの状態だけを考えて，その導入の効果をあらかじめ予測することが一般的であった．

(2-10) ソフトウェア・システムは，閉じた存在（対象システムに関する完全で矛盾のない理解にもとづいて作られたプログラムの集合）であると考えられており，ソフトウェアとそれを取り巻く周囲の環境との相互作用は，開発のさいにあらかじめ定義されている．

(2-11) ソフトウェア開発者は，対象システムの外に立っており，システムの開発と利用は，それぞれ別のものと考えられていた．

(2-12) こうした古典的な＜プロダクト指向＞の考え方が，現在われわれが開発しているほとんどのソフトウェアにあてはまらないことは明かである．

(2-13) 現代的な＜プロセス指向＞の視点では，対象システムは，「ソフトウェア・システムによって支援されるユーザの作業プロセスの設計がうまく行くように」という観点に立って，選択されなければならない．しかも，それはは，動的であり時間とともに進化するものとしてとらえられる．ソフトウェア・システムの導入によりユーザ・プロセスは大きく変化し，それによって対象システム自体に予測しがたい影響がもたらされる．

(2-14) したがって，ソフトウェア・システムは，開いた存在である．任意の時点において現存するプログラムの集合は単なる１バージョンにすぎず，対象システムのさまざまな部分に関する限られた（そしておそらくは相互に矛盾した）理解を実現したものであるがゆえに，いずれは改訂される運命にある．ソフトウェアとそれを取り巻く周囲の環境との相互作用は，ユーザの手で，実際の作業プロセスにおいて生ずるニーズに合わせて，適当に調整される．

(2-15) この場合，ソフトウェア開発者は，対象システムの一部になる．システムの開発と利用は，お互いに影響を及ぼし合うものと考えられる．

(2-16) すでに 1970 年代後半に，M.M.Lehman は，そのソフトウェア進化論（Software Evolution Dynamics）のなかで，S-Program（仕様化可能なソフトウェア）と E-Program（アプリケーションのなかに組み込まれるソフトウェア）とを区別し，前者がごく少数の例外的な存在であることを指摘している．

(2-16) 昨年 11 月の京都国際シンポジウム (ISFSE-1) で L.A.Belady が指摘した Type-A Software と Type-B Software という分類もまったく同じ意味だと考えられる (SEAMAIL Vol.5, No.10-11-12).

3. ソフトウェア・プロセスにおける協調活動について

(3-1) ソフトウェア開発プロセスは，本質的に，開発者およびユーザのあいだの学習およびコミュニケーション・プロセスである．いかなるソフトウェアの開発も，（論理的には）最低２人の人間が含まれる．すなわち，それは１つの社会的活動である．

(3-2) したがって，あらゆるソフトウェア・プロジェクトは，それに関わる個々の人間にとって，１つの社会すなわち＜自己疎外＞の場でしかありえない．

(3-3) ところで，人は最初からソフトウェア・エンジニアとして生まれるわけではない．社会（いくつものプロジェクト）における自己疎外の経験を積み重ねたのちに，ようやく沈潜すべき自己を発見し，技術者として育って行くのである．しかし，そうした自己発見は決して容易なしごとではない．したがって，プロジェクト・マネジメントにおける最大のポイントは，「技術者たちに真の意味での自己沈潜の機会をどれだけ多く与えるか？」であろう．

(3-4) 俗流ソフトウェア・エンジニアリングの大きな誤りは，開発の生産性とかプロダクトの品質とかいったキーワード（それらは，もともと，社会活動としてのプロジェクトを成り立たせるための言い訳でしかありえない）を，プロジェクトの真の目的と置き換えてしまったあたりにある．こうして，大多数のプログラマたちは，硬直したプロジェクトの慣習（規律，標準，生産性指標，etc）に押しつぶされ，ただいたずらに自己疎外の年輪を重ねて行くだけに終わってしまう．特に，大きな組織や大きなプロジェクトには，その傾向

が強いように思われる．

(3-5) たとえば，いま，ソフトウェア·エンジニアリングの成果としての開発方法論について，次のような誤解が一般的である：

- 開発方法論はそれ自体１つのプロダクトである．方法論を正しく利用すれば，必ず同じ成果が得られる．
- 方法論は，開発者間のコミュニケイションを制約する働きをする．
- 方法論は，ソフトウェア開発から属人的な要素を排除する効果をもたらす．
- 方法論は，一般的に適用可能であることが望ましい．
- 方法論は，静的であり，よく定義され，状況に左右されないものでなければならない．
- 方法論は，プロダクトの特性をきちんと記述するのに役立つ．

上の＜方法論＞を＜プロセス·モデル＞と置き換えて読んでもよい．

(3-6) 方法論（またはプロセス·モデル）の正しい役割についての理解は，おそらく次のようになるであろう：

- 開発方法論などは存在しない．単に，方法論が開発され利用されるプロセスだけが存在する．これらのプロセスは，方法論を利用した成果に影響を与える．
- 方法論は，開発者間のコミュニケイションを支援する働きをする．
- 方法論は，それが人々によって使われてはじめて意味がある．
- 方法論は，対象とするアプリケーション·ドメインおよび開発工程と密接に関連している．
- 方法論は，経験によって進化する．また利用者のニーズや開発状況に応じてカストマイズされる必要がある．
- 方法論は，プロダクトが次第に明らかになってくる協同作業のプロセスを支援する働きをする．

(3-7) ソフトウェア·プロセスのモデル化にさいして，もっとも大きな問題の１つは，協調活動をどう取り扱うかであろう．それらはもとより非合理的な社会的必要から生まれるものであって，なんら内的な必然性をもってはいない．そして，そうしたモデル化のための手助けをしてくれるはずの社会学は，誕生いらいすでに１世紀いじょうになるというのに，まだ充分に成熟の水準に達していない．

(3-8) 一方，ソフトウェア開発のキー·ファクタである個々のエンジニアの自己沈潜活動（一時的に社会的活動から身を引いて，孤独なしかし創造的な思索に沈む）を，そのモデルのなかにどう位置付けるかも，もう１つの重要な課題である．

(3-9) これまでに，提起されたソフトウェア·プロセス·モデル化の方法論（記述言語，フォーマリズム，etc）は，プロセス·プログラミングの野心的アイデアを含めて，すべて不適切だといわざるをいない．

(3-10) 十分な社会学的ツールをわれわれがまだ持ちえていないという現実に照らして考えると，とりあえずいまのところは，アナロジカルまたはアレゴリカルな概念モデルによるプロセス記述と，実際のプロセス·ヒストリーとそれらのモデルとの対応をトレース（分析）するような支援環境だけが，わずかに有効であるだろうと想定される．

(3-11) 5th ISPW の Position Statement で私が提示した地図のアナロジーは，その１例である (SEAMAIL Vol.5, No.1-2)．しかし，まだそれはモデルというにはあまりに未熟である．

4. プロセスからの出力としてのプロダクトについて

(4-1) 現在，やはり次のような誤った考え方が支配的である：

- プログラムは，ドキュメントを読むことによってのみ理解されるべきである（ドキュメントはプロダクトの一部を構成する）．
- プログラムの意味は，その仕様書で与えられる形式的意味論によって定義され，ドキュメントを機械的に解釈することによって導かれる．
- 仕様書の重要な構成要素は，プログラム中に具体化されるべきモデルである．このモデルは，１つの形式的言語を用いて記述されなければならない．自然言語による説明はあまり好ましくない．
- プログラムに対するわれわれの知識は，形式的な規則を理解することによって得られる．ドキュ

メントは完全で，一貫性を持ち，あいまいさのないものでなければならない．でないと，解釈が一意的に定まらない．

- ドキュメントは，プログラムが何をするかを，はっきりと記述していなければならない．また，それがどうやってなされるか（開発者の視点）を抽象的に説明すべきである．
- ドキュメントは，設計（およびその他の）プロセスから得られた結果を記述すべきものである．

(4-2) しかし，正しい理解はおそらく次のようなものであろう：

- プログラムの理解には，個人的な議論や試用が不可欠である．ドキュメントは，これらの活動を容易にするものでなければならない．
- プログラムの意味は，作者の意図したところとの関連において考えられるべきものである．またそれは，プログラムがどのように使われるかと，密接に関連している．仕様書と現実世界との関連は，仕様書に関する議論を通じて明らかになる．
- あらゆるドキュメントの本質的な部分は，自然言語で書かれ，いろいろな概念を紹介し，モデルに関する定義や説明を与える．形式的表現は，もし必要なら使ってもよいが，何ら新しいことをつけくわえるものではない．
- プログラムに対するわれわれの知識は，ほとんど，背景となる既存の知識を用いた実例やたとえを通じて得られる．適切な例示やたとえは，ドキュメントのどこがポイントであるかをはっきりさせ，あいまいさに関する人間の側の許容度を拡大する．
- ドキュメントは，プログラムが何をどのようにするか，またそれがなぜであるかを，読者に理解できる言語を用いて，しっかりした観点から説明するものでなければならない．
- ドキュメントは，また，ソフトウェア開発の最終成果がいかにして得られたか（学習の経験，さまざまな意志決定やその理由，採用されなかった代替案など）を記述すべきものである．

5. 環境構築者（ツール開発者）へのメッセージ

(5-1) ツールの本質 － エリック・ホッファ（アメリカの哲学者）

- 人間が最初に飼いならした家畜－イヌ－は，まず遊び友達としての習性を買われたのである．狩猟用に使われるようになったのは，だいぶあとのことであった．また，弓は武器として利用される以前に，まず楽器として作られたのである．

(5-2) 分析とは何か？ － パウル・クレエ（スイスの画家）

- 化学における分析では，未知の物質を要素に分解しその特性を調べ，謎を解く．芸術活動においては，そうした分析のアプローチはとらない（それは，単なる模倣または贋作へと導くだけにすぎない）．芸術における分析は，作品と向かい合って，その作品が生まれた道程（プロセス）を辿ってみることである．それによって，初めて，自分の足で歩くことができるようになる．

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ポジション・ステートメント

落水 浩一郎 (静岡大学)

プロフィル：

大学の教官．「アルゴリズムとデータ構造」，「プログラミング方法論」，「ソフトウェア工学」，「計算機ネットワーク」等の講義を担当している．学科の計算機設備はｅｔｈｅｒｎｅｔｓ−ｗｏｒｋｓｔａｔｉｏｎｓの分散環境である．参加セッションは「ソフトウェア工学は現場にいかせるか」．

Ｐｏｓｉｔｉｏｎ　Ｓｔａｔｅｍｅｎｔ

「ソフトウェア工学は現場にいかせるか」という主題は刺激的ではあるが，ナンセンスである．ソフトウェア工学の成果には嚇々たるものがあり，現在も発展し続けている．ソフトウェア工学の研究者達は生産現場の実情にしっかりと根をはりつつ，その背景にある原理，原則を追求しつづけてきた．疑問に思われる方は添付した論文「ソフトウェア工学の成果と今後の課題」を一読頂きたい．

もちろん，計算機技術の発展や情報処理ニーズの拡大にともない，常に新しい課題に取り組まなければならないという不安定さはある．また，過去に開発された技術のすべてが現在でも有用なわけではない．

しかし，新しい言語を使いこなすには相応の努力（教育，訓練）が必要であり，種々のツールを使いこなすにはその背景理論の理解が必須であり，開発方法論を現場に定着させるには試用実験を進んで実施する必要があることを御記憶頂きたい．

もうかればよいという拝金主義，やればできるという肉体労働者的発想，人のやったことは説明できるという非創造性，新奇なものをいみきらう風土だけの世界からは新しい有用なものは生まれない．

現状を革新するためには良い評価手段が必要であり，革命のためには１０年後を読みきる哲学が必要である．また１０年間耐えうる体力が必要であることはいうまでもない．

ソフトウェアは他の工業製品とは異なり，複雑性，不可視性等の人間にとって厄介な特性を持つ．このような問題点の克服のために過去におこなわれた研究の成果のまとめつつ，今後の課題を展望する．

1．なぜソフトウェア開発は困難か

1．1　複雑性

1．2　非規律性

1．3　拡張性

1．4　不可視性

1．5　人間の弱さと集団制御の困難さ

2．ソフトウェア動作と開発過程の可視化

2．1　プログラム作成過程の可視化と複雑性の制御

2．2　ソフトウェア動作の可視化

2．3　開発過程の可視化と非規律性の克服

3．人間要因とプロトコル解析

3．1　プロジェクト管理における人間要因

3．2　設計における人間要因

4．まとめ

1．なぜソフトウェア開発は困難か

ソフトウェアは他の工業製品と異なり，作成や検査を困難にする基本的な特性を持つ．［1］によれば，複雑性，非規律性，拡張性，不可視性の４つがある．

1．1　複雑性

複雑性を生みだす原因は２つある．

一つは極端ないい方をすれば，「プログラム中には一つとして同じ要素はない」ことである．同じ要素があればサブルーチンの概念を利用して共通化し記述量を減らすことを，例えばプログラミングの初等教育で厳しく訓練される．

もちろん，やみくもに共通化すればよいわけではなく，変更容易性や汎用性とのトレードが存在し，偶然的な一致等は無視すべきことが指摘されている．

いずれにしても，サブルーチン化の徹底はプログラムの要素をすべて異なるものにする方向にあり，１０万ステップ，１００万ステップと規模が大きくなるにつれ複雑さは増大する．同じ部品を組み合わせて作る他の工業製品に比べその複雑さは比較にならない．

事態をさらに複雑にするもう一つの原因は「状態の数」である．
先に述べたプログラムの要素は，ただ慢然と集まってプログラムを構成しているわけではない．機能要素の実行順序やデータの設定参照の関係で相互に関連づけられている．

ある機能要素の実行が終了した「状態」で，計算結果の一部を，後に出現するある「状態」で参照するために共通データ領域に格納しておく．このような相互関係がプログラム中には網の目のようにはりめぐらされている．このような構成要素間の相互関係は要素数の増加につれて非線形に増加する．機械や電子機器等が持つ状態数に比べ，圧倒的に大きい状態数をソフトウェアは持つ．

1．2　非規律性

上記の複雑性に，もし一定のパターン（現象上の複雑性を説明しうる少数の原理）が期待できるなら，対象物は複雑でも，その開発のプロセスにおいては従来の科学とエンジニアリングの手法を適用できる．

ところが，ソフトウェアが対象としている人間社会の情報処理活動は，「千差万別であり，すべてに共通するルール」はありえないので，ソフトウェア内部における処理はそれを忠実に反映し，個々のユーザの情報処理方式と整合性を保たなければならない．ユーザ世界とソフトウェア・システムのインタフェスを遵守することは，通常一方的にソフトウェア側に要求される．

1．3 拡張性

ソフトウェア製品は，応用分野，利用者，法律，コンピュータ・システムなどの文化的基盤の上になりたっている．このような世界は常に変化しておりソフトウェアも共に変化しなければならない．ソフトウェアはその言葉が持つ意味あいが必要以上に過大評価されており，変更要求の出現頻度が他の工業製品に比べて極度に高い．しかし，実際には，変更または拡張という作業は，最初に述べた複雑性の特性の影響をまともにうけて非常に困難な作業である．

1．4 不可視性

最後の特性は不可視性である．人間は設計においてよく幾何学的抽象を道具として使用する．ビルの設計図，機械設計図，半導体チップの設計図しかりである．

ソフトウェアの動作は，この幾何学的抽象がやりにくい．部分的には可能である．プログラム図式，モジュール呼び出しの階層構造図，データ・フロー図等，ものごとの一面を記述する手段は色々ある．しかし，それらが全体としてソフトウェアの構造・動作を構成している．ソフトウェアの理解のためにはそれらの相互関連を把握しなければならない．このようなことは人間は通常不得手である．とくに時間関係の表現や理解は幾何学的イメージで表現しにくいという問題がある．

1．5 人間の弱さと，集団制御の困難さ

以上のような厄介な特性を持つソフトウェアの作成にあたって，以下に述べる肝要な部分のほとんどは人間が担当せざるのをえないのが現実である．個人のレベルでは

a．問題を分析しシステム・イメージを固める（要求分析・定義）

…対象分野の知識を統合しシステム設計へ写像する

b．問題を技術的解に変換する（設計）

…新規開発システム特有の制約を考慮しつつ，

設計問題解決の経験的知識を再利用する．

c．設計結果をコーディングする（コーディング）

…データ設計，アルゴリズム設計に関する経験的知識を有効に利用する

d．誤り検出の方針を定める（検査）

…エラー現象と原因のつながりを発見するための戦略を状況に応じて選択する

e．プログラムを理解する（保守）

…開発者による意思決定の過程を再現し，変更箇所とその波及効果をよみきる

集団のレベルでは，

ａ．秩序だったプロジェクト運営による量（生産性）と質（信頼性）の制御

ｂ．チーム構成員の快適な作業環境の保持

ソフトウェア開発とは，あいまいなユーザ要求を，不確定な要素を次第に減少させつつ，最終的には正確・高速ではあるが一切融通がきかないプログラムとデータ群に変換していく作業であり，個人のレベルでも集団のレベルでも単なる標準化による強制では対処しきれない面がある．単に管理的側面を強調しすぎるとソフトウェア作りを味気なく辛いものにする．部分の改善が全体の効果につながるという製造プロセスの原理がいかせつつ，かつ，個人の能力が開発・改善されていくようなシステム化が必要であり，そのためには人間や人間集団の行動特性についての十分な理解を持つ必要がある．

・個人が利用する便利な道具

・そのような人々が５，６人程度の小人数で協同作業を進めるとき，少数のルールによりチーム活動を調整すること

・さらに，チーム構成員の数が数十人以上になるプロジェクトにおけるプロジェクト構成員の個々の活動を一定の方向に持っていく協調方策

・組織レベルでは，そのような方策を生みだすためのメタ方策（資源の管理，共有，再利用を促進する方策）等がうまくかみあう必要がある［２］．

２．ソフトウェア動作と開発過程の可視化

大昔は「計算」は人間だけが可能な知性であった．電子計算機の発明以来，計算は石ころでもできるようになり，計算を組織化するプログラミングの仕事が一部のエリートにのみが保持する知性であった．高級言語の発達により，要求される知性の範囲はさらに拡大し，冒頭に述べた複雑性や拡張性を制御できる知識の集積と利用法の開発が課題になりつつある．このように知性の前線は除々に拡大しつつある．

以下に挙げるソフトウェア工学の成果のうち，人間や人間集団の持つ弱さを補完するために開発された諸技術に焦点をあてることにより，現状を整理してみよう．

（１）計算機動作のマクロ記述による記述量の削減，制御構造やデータ構造等の記述法の改善，それらを処理するツール群の整備

（２）テストや変更が容易になる構造をシステムに組込むための設計方法論やプログラミング方法論を開発する．

（３）ユーザ要求の表現手段（要求定義）を開発する．

（4）要員見積り，危険要因検出，検査打ち切り時期の判断等のプロジェクト管理技法を開発する．

（5）標準化により再利用を促進する．

（6）システム動作を早期に確認するためのプロトタイピング技術を確立する．

（7）必ずしも専門家とは限らない利用者，開発者の増加に対応するための技術を開発する．

2．1 プログラム作成過程の可視化と複雑性の制御

プログラミングの世界では，

・プログラミングにおける「記述単位」と「プログラマの思考単位」の対応のよさ

・全体の記述量の削減

が研究されてきた．

（1）構造化プログラミングにおいては，段階的詳細化法「問題を解決する理想的な仮想機械をまず想定し，仮想機械の命令とデータ型の分解を排他的に実施しつつプログラム動作の具体化を進める」の導入により，一連の決定の連鎖によりプログラムを完成する方法を定式化した．技術的には，データ型の分解にともなう一群の決定がソース・コード上には散在するという問題点が残された［3］．

詳細化過程の図式表現技術も開発が試みられた．

図式表現の原則として採用されたのは「右出し型」と「ネスト型」である．

ネスト型の代表例としてはＳｃｈｎｅｉｄｅｒｍａｎｎ－Ｎａｓｓｉチャートがあげられる．一入力一出力の様々な制御構造に対して用意された図式単位を順次埋め込み展開する．埋め込みのための図式はネストが深くなるにつれて新しい紙に切り替える必要があり，紙の切り替えと設計上合理的な切り替えのタイミング（例えば，データ型の詳細化が必要であるとの認識）があわない問題があり，あまり普及しなかった．

右出し型，すなわち分解毎に詳細化の単位を右に出していく方法は，多くの生産の場で開発が試みられた．ＨＣＰチャート，ＰＡＤ等がある．このような，図式言語を使いこなすためのグラフィック・エディタ，コード生成機能を具備した環境の開発も進められた．

共通する問題点は以下の通りである．「仮想機械命令の分解」の側面のみを支援し，「データ型の抽象化のレベルに違いにより層を切り分ける」，「データ型の分解の陽な表現」の２点が技術的に克服されていない．また，技術的にはさらに進んだ「データ抽象」の

概念もとりこまれていないため，その効果にはおのずから限界がある．

（2） データ抽象と抽象データ型

構造化プログラミングの手法によれば，データ型の分解によって生じる仮想機械命令の分解は，ソースコード上では散在した箇所に出現することになるが，この問題を解決したのがデータ抽象の概念であり，さらに進んで抽象データ型の考え方を生み出した．

データ構造の定義とそれに直接関係する操作の定義がソーステキスト上で一箇所にまとまって記述され，情報の局所性が高まるような記法をデータ抽象という．データ抽象の考え方によりデータの定義場所とその利用場所が分離され，データ構造の実現の詳細を変更したり，それに伴なうアクセス法を変更する作業を利用箇所を変更することなく実施できることになる．また，関数によってのみデータ実体にアクセスするので，複雑性や拡張性を情報の局所化という面から制御可能にした．

データ抽象は２次入口を持つ手続きや関数を与える言語では容易に実現できるが，同じデータ型を記録する領域を複数個必要とするような問題では，変数毎にデータ抽象を実現した手続きを複数個容易しなければならない．これははばかげている．

この問題に答えるのが抽象データ型の概念である． 抽象データ型ではデータ抽象の概念にもう一つ概念をつけくわえる．データ型とその操作の表現を手続き表現とせずに型定義として表現する．型定義部においてただ一箇所，データ構造の詳細とその操作法を定義し，必要な数だけ変数宣言部で領域を確保して名前をつける（変数名またはインスタンス名）．インスタンスに対する呼び出しは，インスタンス名と操作を指定して実施される．

抽象データ型の概念は１９７４年に提案された［4］．このように技術的優位性を持った概念が，日本では長い間生産現場に浸透しなかったのは驚くべきことである．最近，オブジェクト指向言語が出現するにつれて除々に世に浸透しつつある．

（3） オブジェクト指向

オブジェクト指向プログラミングパラダイムとは，抽象データ型によるプログラミング・パラダイムの特徴に加えて多くの型定義の間に存在する類似性を整理して表現する機構が存在する（クラス間の遺伝継承）可能である．Ｓｍａｌｌｔａｌｋ８０の場合，「あるクラスは上位クラス（スーパクラスという）で定義された操作を定義なしに使用できる」という木構造の関係がクラス間に定義でき差分プログラミングを可能にする．クラス階層はまた部品化を可能にし再利用を促進する効果もあるが，このためには強力なブラウザが必須である［5］．オブジェクト指向プログラミングにおいては，現実世界の「もの」を

そのままオブジェクトとして認識しプログラミングを進めるので，問題世界と計算機における実現方式の対応のよさも大きな特徴となっている．

2．2 ソフトウェア動作の可視化

設計初期段階における設計対象の動作確認手段の開発は最近の重要な課題であり，種々のCASE（Computer-Aided-Software-Engineering）ツールが開発されつつある．CASEツールは，グラフィック・エディタを利用して設計対象システムの動作記述や解析を図式言語（状態遷移図，データ・フロー図，ペトリネット等）に基づいて支援し，設計初期段階におけるシステム動作の可視化をはかるものである．また，ADAやCなどのソースコードの一部を自動生成するものもある．各記法には以下の特徴がある．

データ・フロー図は，論理的なシステム機能と機能間のデータの流れ（各機能の入力と出力）を明確に表現する．機能の起動タイミングが問題になる場合は，別に状態遷移図を書きデータ・フロー上に制御の流れを書き込む．

ペトリネットは，幾つかの並行動作システム間の同期条件を明確にし解析を支援したい場合はペトリネットが使用できる．

またオブジェクト指向プログラミング・パラダイムと問題世界との対応のよさに注目し設計パラダイムとしても利用する試みもある［6］．

2．3 開発過程の可視化と非規律性の克服

ソフトウェア開発で生成される中間生成物（文書等）や，最終製品を記述する従来の言語の立場に加えて，最近そのようなものを生成していくプロセス自身を解明しようとする研究がおこり，注目を集めている．

プロセス・プログラミングの提唱者であるOsterweil によると，

「ソフトウェア開発とは数多くの小さなソフトウェア・オブジェクトを作る作業でありそのようなオブジェクトは，まったく予測不可能ではない．それゆえ，オブジェクトを作成する順番をあらかじめ用意されたプランに従って秩序正しく作成することも可能である．すなわち，さまざまなソフトウェア・オブジェクトが作られていくプロセスを記述（プログラミング）することが可能である」［7］としている．

用語の硬さのせいか，一般には実現困難な課題とうけとられているようだが，実はこの概念は，さまざまな場所ですでに指摘されつつある概念である．例えば，シナリオという概念はほぼ同じねらいを持っている．

例えば，社内の生産環境の体制作りの場面を想定して見よう．社内における標準開発工程を分析すると以下のような要素に分けられるはずである．

（1）オブジェクトとは

業務分野（事務計算，組み込みシステム，通信，科学技術計算）に依存するが，開発工程の中で必ず明確に記述して分析等をしなければならない情報があるはずである．例えば，事務計算の場合は客先の業務フロー，出力帳表の様式，データベースの設計書，システムフロー，プログラム仕様等が例としてあげられる．もちろん，プログラム仕様書の様式等は社内の標準とは別に客先からの指定があることが多く，形式も種々異なり実際の作業ではそんなにうまくことが運ばないかもしれないが，おもいきって，社内標準を客先標準に変換する文書自動化ールの問題としてとりのぞけるものとする．すると業務分野毎に，長い間の社内での開発経験の蓄積から生れた，開発過程で必ず作成されなければならないものが認識できるはずである．実際の開発で作成された文書類から，客先固有の情報を除去し，低水準の標準化規約のせいで実際の開発者が作成したり，参照している情報との間にずれがある場合にはそれを調整した上で，性質の同じものを分類すると，いくつかの集合にわけることができる．一つの集合（分類）に属するものすべてに共通する性質を洗い出してみると，それは記述されるべき情報に対する仕様を与えていることになる．これをむつかしくいうと型付きオブジェクトという．

（2）関係とは

型付きオブジェクトの間には，開発者や開発管理者が日常暗黙の内に利用している相互関係があるはずである．例えば，これを作成する場合には，これとこれを参照しなければいけないとか，これが決定されていない場合には，あれを作成することはできないとかいう関係（派生関係）があげられる．

また，一つの型付きオブジェクトのインスタンスの間には，これは，あれを変更することによってできたものだというような関係がある（版の関係）．

このように，情報の管理のために必要な関係を認識することで開発によって生成されるべきものに対する情報構造を設計することができる．

（3）プロセスとは

ここで再び，標準開発工程を吟味する．標準開発工程の中には，長い間の開発経験を通して整備されてきた手順的な情報もあるはずである．検査の手順，承認の手順，部分的な作成の手順等枚挙にいとまがない．もちろん，開発者の創造性に大きく依存する部分もあ

るので，そこは思いきってブラックボックス化し，その活動には名前をつける．ブラックボックス化の単位が大きすぎて，管理不可能と判断される活動に対しては熟練者の行動パターンをいくつかあつめ，熟練者間のコンセンサスをとった上で標準化してしまう．つぎにこのような開発手順を，先程整理した型付きオブジェクトを利用して表現を変えてみる．そのような表現がすなわちプロセス・プログラムである．

もちろん，プロセス記述言語は手順的なものがすべてではない．整理された手順は，開発作業全体を包含していないことが多いと思われる．いわば，サブルーチンの集合が得られていると考えた方が良い．ここで，サブルーチン群の起動制御が問題になる．メイン・プログラムを書くことが出来るという立場からは，Ｏｓｔｅｒｗｅｉｌのプロセスプログラミングの考え方が妥当であり，開発状況に依存して起動されるべきだという立場からは，前提条件，後件を附加したＧ．Ｋａｉｓｅｒのルールベース記述の考えかたも魅力がある［8］．どのような方式が妥当であるかについては，まだ結論が得られていない．ソフトウェア開発に特有の現象「先に進んでみなければわからない」「失敗した時どこまでもどればよいのか判断がむつかしい」「ソフトウェア開発の方針レベルでは同じだが，プロセスの詳細は事例毎に異る（プロセスの動的変更）」というような問題がある．

また，プロセス自体が決定論的ではなく，非決定性とナビゲーションの機能等も環境に組みこまなければいけない．

3．人間要因とプロトコル解析

繰り返しになるが，ソフトウェア開発の主たる担い手は人間である．そこで，ごくあたりまえのことではあるが，人間要因に関する理解不足が上記諸技術の直接または間接のボトルネックになっている．近年，このような理解のもとに人間要因に関する研究がさかんになり，プロジェクト管理技術やソフトウェア開発環境との関係がみなおされている．ここでは，ソフトウェア要求分析・基本設計の局面にしぼって，人間要因に関する最近の研究動向のあらましを紹介する．ここでいう人間要因とは観測と分析によって裏打ちされた，個人や集団の行動特性や人間機械系に対する洞察結果を意味する．知識獲得の一手段としてプロトコル解析がある．プロトコル解析とは問題解決活動に携さわる個人や集団の活動を，テープレコーダやビデオを利用して記録し解析することにより，人間の行動特性に関する種々の規則性を発見しようとする実験心理学の一手法である．手段や結果の信頼性については充分吟味する必要があるが，ソフトウェア・エンジニアリングに関する有用な知識を集積するにあたっては欠かせないものである．

3．1 プロジェクト管理における人間要因

品質と生産性向上をとらえる最も大きな変量はソフトウェア設計過程におけるコミュニケーションと意思決定のプロセスに存在するという仮定のもとに大規模システムの開発事例から収集されたデータを基におこなわれたプロトコル解析の報告がある［9］．

ソフトウェア開発パラダイムの意義はソフトウェア開発活動を生産的で信頼性の高いものにするガイドラインを与えることであり，開発モデルの構築にあたってはいくつかの視点がある．

（1）何がいつ得られるかという立場から管理する

生産性が上る変量として，「いつ」「だれが」おこなうかの2つを仮定しており，さらに，中間生成物を作りだす活動が信頼性高く遂行され，ある活動が不首尾におわったばあいはそこから影響を調べうるという仮定がある．このようなモデルはもし基本活動の担い手の変動要因が大きい場合には（例，人間），その出力によって失敗の原因を説明することが困難であり，管理者にとっての価値が減少する…ウォータフォール・モデル

（2）人間の関与する部分を少なくする

操作型仕様パラダイム，形式仕様と自動変換パラダイムがこれに属する． これは，製造工程における変動要因（人間）の関与する部分を少なくするという効果は望めるが，研究室レベルの成功からさらに一歩進めるにはまだまだ時間を要し，さらにもっと重要な問題点として，不確定要素（ユーザ要求の不確定度を減少させるある種の重要な要因）を制御できないという欠点を持つ．

これに対して，ソフトウェア開発において最もおおきな変動要因である人間要因を明らかにするという立場からフィールド研究がなされ，以下のような結果が報告されている．

（1）個人レベル…個人の能力に関する観測

成功したプロジェクトには以下のような能力を持つプロジェクト・リーダが存在した．

・対象分野知識の統合化能力とシステム設計への写像力・プロジェクト構成員への意志伝達能力
・プロジェクト性能の技術的評価力

（2）チームレベル…チーム内意思伝達機構の特徴

初期のシステム・イメージを形成するためにあらかじめ必要となる作業は，応用分野に関する知識の形式化，いくつかの分野の知識の統合，人間の内部に存在する知識イメージの外部表現（記述），それの他人に対する伝達であり，プロジェクトチームの時間資源の

大部分を消費する．このような知識統合の作業は，個人やチームの少数の人間によって達成される．複数の人間が関与する場合のコンセンサス形成のプロセスについても報告されている．最下層の意思決定は，各人によるシステム動作に対する内部イメージの形成であり，その上層でそれらが部分的につきあわされていくつかの連合モデルができる．最終的なチームコンセンサスは他の影響を受けつつも，連合モデルのどれか一つが採用されることで実現される．採用されるモデルは応用分野に関する知識豊富な少数の人によって提案されたものになる場合が多い．２つの競合するモデルがあらわれることは少ない．このときチームの他のメンバの意見が全く無視されるわけではなく，採用されたモデルに対する広い視野からの検討を考慮させる形で影響を与えている．このことは，エキスパートといえども，個人は狭く深く考える特性を有しており，チーム内メンバの協同によって広い視野からの吟味が可能になることをあらわしている．

（３）プロジェクトレベル…手段の検討

設計を首尾良く達成するために必要な手段を学ぶために設計の時間の大部分は消費される．これに要する時間の必要性は，一般に過小評価されすぎている．

手段の検討の具体的内容としては例えば次の２つがあげられる．

ａ．システム動作の中心部の抽出

ｂ．システム動作の表現媒体の選択

また，応用分野特有の知識を学習するプロセスは，通常のライフサイクモデルでは明示的に表現されず，かくされてしまう

このようなフィールド研究の結論として，ソフトウェア開発においておおきな比重をしめる活動は，ユーザ要求や開発手段に対する学習と，プロジェクトチーム内，対ユーザに対する意見交換，意思伝達の活動であると結んでいる．

３．２　設計における人間要因

設計の作業もまたおおきく人間要因に左右される．これに対処するためには，再び人間要因の観測と把握が必要である．このような観点から以下の２つの研究結果を紹介する．

（１）　ソフトウェア設計における，問題領域に関する経験の効果［１０］

制御関係の設計者を対象にしてプロトコル解析をおこなった結果が報告されている．観測結果を表１に示す．設計エキスパートや初心者の対象分野に関する知識や，問題解決の経験に依存する行動特性の分析例が興味深い．

表 1　設計者の行動特性

行動特性	設計コンテクスト			
	(既知の分野で未知のシステムを作る場合) 熟練者	初心者	(未知の分野で未知のシステムを作る場合) 熟練者	(既知の分野で既知のシステム 熟練者)
メンタル・モデルの形成	動作シミュレーションを支援するワーキングモデル（図表現）を書く．モデルを形成する能力は対象分野に関する知識に依存する	メンタル・モデルを作れない．代わりに擬似コードで表現する	全体モデルを形成する能力は，領域に関する知識に依存するので，この場合，利用者視点から記述がなされ，実現の視点（OS との関係）が欠ける	以前にやっていることで記憶の中にあることなので，詳しく書く必要がない．そこで抽象的かつ整理された絵のみを書く
メンタル・モデルの系統的詳細化	シミュレーションを行なうために入出力のレベルを少しだけ詳細化する（抽象→具体）	擬似コードの展開しかできない	誤りを犯すことがある． 2つのデータベースが必要であるのに一方しか展開しない	仕事を挑戦的にするために，自分にとって新しいことにのみ注意を払う
メンタル・モデルのシミュレーション	シミュレーションを実施し，全体の動作を確認する．この能力は対象分野に関する知識と無関係な一般的能力である	シミュレーションができるモデルをもっているようには思えない	シミュレーション能力は一般的能力であるが，全体モデルが不十分なので全体の相互作用についてシミュレーションしきれない	実施しない．どうすれば問題を解決できるかを知っているので，動作を確認する必要がない
制約の表現	メンタル・モデルには現われてこない問題の性質，実現上の制限の考慮	行なわれない． 知識が少ないので推測できない	実現法に関する考慮が少ない	一番不慣れな点で発生する
問題解決プラン片の想起とラベル付け	以前解決した問題を使用できると考えたとき，それにラベルを付けて覚えておく	プランは低レベル． 擬似コードで表現される	量が少ない．問題がどのような性質をもっているかではなく，それをどのように実現するかについての知識が少ないため	頻繁に行なう． 解法空間の知識が豊富
注意書きの作成	今，考えなくてもよいが，展開を進めていくうえで後で必ず必要になる点が見つかると，そのレベルに達したときに考えるべきであるということを書き出しておく	問題文を読んでいる間だけ作成される． しかも，すぐに没頭するか，結局忘れてしまう	経験が少ないので，どこで問題が発生するのか予測できず作成されない	作成しない． 自身の経験に頼り，動作解析や注意書きの作成をしないため

（2） 図形表現の効果［１１］

複数種類の制約下で，トレードを満足する解をみつけだすという設計問題を，空間的仕様（事務所のレイアウト）と時間的仕様（シフトの構成）それぞれの場合について被験者にあたえ設計に要する時間や設計結果の質を比較するという実験がおこなわれた．もちろん，双方の問題に対する制約はグラフで表現すると同じ構造をもつ．この結果後者の方が，誤解や誤りを生じやすく，また時間がかかるという結果がえられた．時間の概念がはいる問題はより解決困難であるということを意味する．

しかし，興味深いことには，時間的仕様の図式表現法を工夫し，別の被験者に，使いかたを訓練したうえで同じ実験をおこなうと，その差がなくなったと報告されている．

4．まとめ

ソフトウェアは他の工業製品に比べ，複雑性，非規律性，拡張性，不可視性等の特性を持つ．また，ソフトウェア開発の主たる担い手である人間に要求される知性のレベルは，個人レベル，集団レベルにわたって高度なものである．

・複雑性や拡張性を制御するために設計対象物の表現法や構築・解析法

・ソフトウェア・プロジェクトの運営を円滑にするための，人間集団の協調と調整法

・経験者と初心者の差の認識と対処法

・開発活動プロセスのモデル化と表現法

等に関する研究が現在進められている．

－参考文献－

[1]F.P.Brooks Jr.,"No Silver Bullet-Essence and Accidents of Software Engineering",IFIP,1986.
[2] D.Perry and G.E.Kaiser,"Models of Development Environments",10th ICSE,1988, pp.60-67.
[3] E.W.Dijkstra, C.A.R.Hoare, O.J.Dahl 著, 野下, 川合, 武市共訳：構造化ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ, ｻｲｴﾝｽ社,1975.
[4] B.H.Liskov, S.N.Zilles,"Programming with Abstract Data Type",Proc. ACM SIGPLAN Conf. on Very High Level Languages,SIGPLAN Notices 9,4,pp.50-59,1974.
[5] A.Goldberg,D.Robson 共著, 相磯秀夫監訳,"SMALLTALK-80言語詳細-",ｵｰﾑ社,1987.
[6] G.Booch,"Object-Oriented Development",IEEE Trans. on SE,Vol.SE-12,No.2,pp. 211-221,1986.
[7] L.Osterweil,"Software Process is a Software Too", 9th ICSE,1987,pp.2-16.
[8] G.E.Kaiser,P.H.Feiler,"An Aechitecture for Intelligent Assistance in Software Development", 9th ICSE,pp.180-187,1987.
[9] B.Curtis et al,"On Building Software Process Models Under the Lamppost",9th ICSE,1987,pp.96-103.
[10]B.Adelson,E.Soloway,"The Role of Domain Experience in Software Design"IEEE Tutorial,ORDER#577,pp.233-242.
[11]J.M.Carroll et al,"Presentation and representation in designing problem-solving",IEEE Tutorial,order# 577,pp.261-271.

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ソフトウェア工学は役に立つ

佐原 伸 (SRA)

はじめに

「ソフトウェア工学など役に立たない」という声をよく聞く。いわゆる現場で「仕事」をしている「本当のプログラマ」や「管理者」にこういうことを言う人が多い。

結論から言うと「ソフトウェア工学は役に立つ。しかし、COBOLやFORTRANやBasicの世界で仕事をしていると、その恩恵がまことに少ない」ということになる。

このようになった原因はいくつかあるが、一番大きな要因は「ソフトウェア工学をやっている人達で、これらの言語に興味を持っている人はほとんどいない」という点にある。

従って、COBOLやFORTRANを使いながら「ソフトウェア工学は役に立たない」と言うのは、本当は暗いところで財布を落としたのに、街灯の下で財布を探しているという逸話に似ている。

証拠を示すために、個人的に開発したプログラムの属性を時系列的に並べ、今と昔とで、何がどう変わっているかをまとめてみた。また第3者が開発したプログラムについても、いくつか調べてみた。

その結果、HyperCardやSmalltalkやObject Pascalなどのオブジェクト指向言語、UNIXなどの強力なツールキット、あるいはInformix 4GLなどの第4世代言語等を使った結果、大きな生産性の向上が得られていたことが分かった。これらの生産性向上の要因を調べたところ、生産性だけでなく信頼性の向上にも役立つことが分かった。

個人的生産性の向上

表-1は、筆者が作成した主なプログラムの履歴である。作成年度・OS・開発言語・開発時間・機能量・ステップ数・対象アプリケーション・開発方法論を列挙した。これらには、会社の仕事として作ったものも、個人的な趣味で作ったものもある。方法論の欄にOOとあるのは、オブジェクト指向のことである。また、言語欄にMacAppとあるのは、Object Pascalを使用し、MacAppというオブジェクト・ライブラリーを利用した場合を示す。

ここで、開発時間は純粋にプログラムを作成した時間だけでなく、会議などの時間も含まれている。また、ステップ数には注釈も含まれている。機能量は対象アプリケーションが住所録プログラムであるものについて、主要な機能の量を比率で表したものである。

13番目のグラフエディタの見積りでは、ソフトウェアのコスト見積ツールであるCOCOMOも使った。

表-1 個人の生産性

	A	B	C	D	E	F	G	H
1	作成年	OS	言語	時間	機能	行数	内容	方法論
2	1973	HIPAC-103	Assembler	20		50	Dice	フローチャート
3	1974	MELCOM	Fortran	10		100		フローチャート
4	1975	EXEC-8	Fortran	600	1	3000	住所録	フローチャート
5	1977	EXEC-8	Fortran	600		3400	有価証券評価	フローチャート
6	1977	PET-2001	Basic	500	1	1500	住所録	インタープリター
7	1980	EXEC-8	COBOL	120		3000	2次オンライン	構造化設計
8	1981	CMS	PL/I	600		11000	投資分析	構造化設計
9	1981	VOS3	PL/I	100		2000	電卓	コンパイラー理論
10	1982	VS-100	Basic	300		3000	COBOL解析	コンパイラー理論
11	1982	VS-100	Basic	600		12000	COBOL生成	ツールキット
12	1982	Unix	Lex, Yacc	100		1200	COBOL解析	コンパイラー理論
13	1983	Unix	C	150		3000	グラフエディタ	ツールキット
14	1983	Unix	awk	24	3	78	住所録	ツールキット
15	1984	MS-DOS	Yacc	3		100	電卓	コンパイラー理論
16	1985	MS-DOS	Informix	10	3	80	住所録	関係モデル
17	1988	Macintosh	HyperCard	3	6	322	住所録	OO+HyperText
18	1989	Macintosh	HyperCard	2		319	英語辞書	OO

表-2 他人の生産性

	A	B	C	D	E	F	G
1	作成年	作成者	OS	言語	機能	行数	内容
2	1986	K.J.Schmucker	Macintosh	MacApp	1.2	2551	グラフエディター1
3	1986	K.J.Schmucker	Smalltalk	Smalltalk	1	556	グラフエディター1
4	1983	佐原伸	Unix	C	2	3000	グラフエディター
5	1988	大久保清貴	Smalltalk	Smalltalk	1	400	グラフエディター2
6	1988	川辺治之	?	CLOS	1	400	グラフエディター2
7	1988	根本敬	Unix	Objective-C	1	300	グラフエディター2
8	1988	小島泰三	Unix	C++	1	1900	グラフエディター2
9	1988	Apple	Macintosh	C	1	1459	テキストエディター
10	1988	Apple	Macintosh	Pascal	1	1270	テキストエディター
11	1987	Apple	Macintosh	Object Pascal	1.3	396	テキストエディター

他人のケース

表-2は、第３者が作成した２種類のグラフエディターと１つのテキストエディターを比較した表である。

生産性向上の要因

住所録・COBOLパーザー・電卓プログラム・グラフエディター・テキストエディターによる生産性の差が、数十倍から数百倍、場合によっては1000倍を越えるのを見てきた。

何故そのような差が発生するのか、以下に見てみる。

言語とライブラリー

最大の要因は、プログラム言語の進化だろう。生産性の悪いプログラム言語 Fortran, Basic, COBOL等では、

(1)モジュール化
(2)情報隠蔽
(3)構造化プログラミング

が素直にできなかった。

その結果、既存のライブラリーがほとんどなく、自ら手作りしなければならなかった。例えば、COBOLパーザーのBasic版は、Unix上であれば標準関数として利用できる多数の関数を相当な工数をかけて開発しなければならなかった。それでも普通のパソコン上のBasicよりは相当強力だったのである。

一方で、生産性の高い言語は、モジュール化や情報隠蔽あるいは構造化プログラミング能力は当然備えているが、他にも住所管理やCOBOL解析や電卓に関する知識を、言語機能として持っていた。

例えば、

(1)文字列の扱い
(2)文字列パターンの一致
(3)検索
(4)対話機能
(5)リレーショナル・モデル操作
(6)OSとのインタフェース

などに、生産性の悪い言語よりも遥かに優れた機能を持ち、しかもその機能の実行効率も優れていた。

特に、オブジェクト指向言語の場合、HyperCardやSmalltalkやMacAppなどのように

(7)各種のデータ構造とそれへのアクセス
(8)標準的なユーザーインターフェースの実装

があらかじめライブラリー化されて用意されていると、他の言語に比べて特に顕著な生産性の向上を示した。

また、HyperTalkやSmalltalkでは、

(9)インタプリタ
(10)差分コンパイル

などの仕掛けにより、プログラムの一部を作り、全体の完成を待たずにすぐに動かすことができた点も大きい。

開発環境とツール

次に重要だったのが開発環境とツールである。

優れた言語が動く開発環境・ツールの方が、良くない言語が動く開発環境・ツールよりも遥かに優れていた。実際は良くない言語に、開発環境・ツールはほとんどないといって

良かった。

(1)エディター

(2)コマンド言語

(3)ツール群

などに優秀なものが多く、プログラムを段階的に少しずつ作り、それをすぐに確認することができた。

これらのツールは、構造化設計、ツールキット方式によるツールの再利用、コンパイラー理論の使用などを支援してくれた。というよりは、これらのツールを通して優れた方法論を吸収し、生産性が飛躍的に高まったともいえる。

おわりに

ここまでで、「ソフトウェア工学は役に立つ」ことを見てきたが、これだけでは問題は解決しない。相変らずソフトウェア工学はボチボチと発展し、相変らずCOBOLやFORTRAN世界は「ソフトウェア工学の世界」と遭遇せず、「巨大な負債を再生産していく」だけだろう。

では、どうすればよいのか？

筆者は、以下の3つが重要であると思っている。

(1) ソフトウェア工学・科学的手法を使って、身の回りのシステムを作っていく。

(2) ソフトウェア工学・科学界とCOBOL, FORTRAN界の交流を図る。

(3) ソフトウェア工学・科学的手法を駆使するプログラマの地位を向上させる。

何故参加するのか？

表-1に示すように、かってはFORTRAN, COBOL界に身を置き、今やソフトウェア工学・科学の恩恵に浴しているので、「解くべき問題がない」状況になっている。このワークショップに参加して、「何が問題か?」を改めて考えたい。

また夜の部?では、日頃考えている「日本でソフトウェア工学が普及しない」のと「日本でサッカーが普及しない」のは同じ原因であるという推論を酔論したい。

さらに、HyperTalkなどのハイパーメディアにも興味を持っているので、そちらの議論にも話題を提供できる。

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ソフトウェア工学はこのままでは現場に活かせない？？

新田 稔 (SRA)

ソフトウェア工学を現場に活かすのは，現状ではかなり難しいと思う．なぜなら，ソフトウェア工学が目指しているものと，ソフトウェア開発の現場が必要としているものが，くいちがっているからである．すべてがそうであるというわけではないが，一般に，ソフトウェア工学はソフトウェアを正確·精密に作ることを目指しているのに対し，現場が求めているのはソフトウェアを簡単に作る方法である．あるいはもっと俗ないい方をすれば，ソフトウェア工学は「バカにはプログラムを作らせるな」と警告しているのに，現場は「バカにプログラムを作らせる方法」を必要としている．

われわれは，ソフトウェア以外の分野では，いろいろな方法論や道具をきわめて有効に活用しているが，それは，それらの分野では人間の生 (なま) の能力がもともと低かったからかもしれない．しかしそのような分野でも「最後は人に頼っています」というフレーズがよく聞かれる．ソフトウェア開発については，人間の生の能力がもともとかなり高い．したがって，作業全般を人に頼ることになり，ツールを導入してもその運用のオーバーヘッドだけが目だってしまうことが多くなる．

ソフトウェア工学が理路整然と筋道だった方法論を研究しているのに対し，現場のやり方は臨機応変の無手勝流である．システマチックな考えがそのまま通用するようなものについては，われわれは充分強力な武器をすでに持っている．たとえば，コンパイラ，コンパイラ·コンパイラ，各種のトランスレータ，ジェネレータ，バージョン管理，データベース定義用のツール，エディタ，ワープロ，フォーマッタ,.... などである．しかし，そのような武器を持っていても，ソフトウェア開発全般に対しては，やはり無手勝流の方が優れているようだ．

とはいっても，現場に役立たない工学は価値がない．何とかして，ソフトウェア工学を現場で活かさなければならない．ひとつの方法は，現場で，多くの機械的に働く SE たちの上に少数の頼れるプログラマが立ち，下位をコントロールしながらソフトウェアを開発する．SE たちの仕事ぶりを観察し，単純なものから機械に置き換えていくのである (注 1)．今までのソフトウェア工学には，この観察という作業が不足していたように思える．

もうひとつの方法は，ソフトウェア工学で「工学的無手勝流」を研究することである．無手勝流の基本は，矛盾·不完全·曖昧さに対する寛容であり，問題にぶつかったときの創意·工夫である．たとえば，はじめから完全な記述や情報伝達を行なおうとすると，要求定義書や設計書の厚さでご存知の通り，われわれは膨大なデータを扱わなくてはならない．ところが，チョットいい加減でもよいことにすると，扱うべきデータ量はかなり減る．

どれぐらいのいい加減さを容認するとどれぐらいのデータが節約できるのかや，完全·正確を期すために膨大なデータを記述したとき，それが完成したときにすでにもとになるものが変更されていて修正が必要になるということと，最初はいい加減にしておいて，必要になったとき，必要になった事柄から正確に記述していくことを比較すると，どの程度のいい加減さがもっとも効率的であるのか，などを研究すべきである．曖昧なものを曖昧なまま取り扱うというファジー的な (注 2) 考えや，オリエント的オブジェクト指向 (注 3) などはどうだろうか？

[注 1] SE とプログラマとの位置関係に注意．SE の下働きであるコーダのことをプログラマと呼ぶ人が多いがそれは間違い．人間を「プログラム」という言葉をキーワードに，マシンにたとえて３種類に分類することができる．すなわち，

(1)Programming Machine，(2)Programable Machine，そして (3)Junk

である．ソフトウェア開発ラインにおいて，プログラマは (1)，SE とその下は (2) である．プログラマより上が (3) であるかどうかは不明．

[注 2] 今まで 0 と 1 で表していたものを 0 から 9 で表すというスケールを細かくしただけの似非ファジーではなく，真面目に曖昧さを扱うもの．

[注 3] Oriental Object Oriented = O3 = Ozone

Seesion 3: ソフトウェア工学は現場に生かせるか？ (Pre-Workshop Position Paper)

ポジション・ステートメント

野村 敏次 (日本電子計算)

１．はじめに

私の理解では、工学と名前がついているものは実践的な学問と位置づけており、現場に生きないものは工学ではないと思っている．「ソフトウェア工学は現場に生かせるか」というテーマは、今まで「現場で生きるはず」ということを前提に、何故現場で活用されないのか、如何にしたらソフトウェア開発現場に適応させられるかという態度でこの問題に接してきた者にとっては、ソフトウェア工学というものを否定する命題のようにも思える．現在、余りにも概念が先行することが多いため、こうした議論が起きてくるのであろう．

更に、大学や研究機関におけるソフトウェア工学の研究とソフトウェア開発の現状が余りにも掛け離れているためにこうした議論が必要になってきているものと思われる．

２．今まで本当に生かされなかったのか

少なくとも、１５年前位は、ソフトウェア開発の現場で何等かの形でソフトウェア工学は生かされていたのではないだろうか？現場に生かされていないと多くの人々が感じ始めたのは、ここ１０年位のことではないだろうか？ここでは、ソフトウェア開発現場において何故ソフトウェア工学が活用されていない（と感じている）かを考えたい．

（１）多様化

現実のソフトウェア開発環境は、非常にバラェティに富んでおり、開発内容も複雑にして多種多様である．端末やワークステーションの数は年々増加し、今や２人に１台にまで普及し、近い将来においては１人１台の端末／ワークステーションの利用は確実なものとなった．ここで言える事は、あらゆる開発環境に適応する方法論、技法、ツールといったものが、存在し得なくなってきているということであろう．現在においては、これらが様々な開発環境で共通的に利用されるということの方が、むしろ例外であると考えて良いであろう．

（２）開発者と利用者の背離

現在企業内においても、往々にして、ツール等の開発者と利用者は育った環境が異なり、コンピュータの利用環境も異なる．そこから、はたして利用者が必要とする技法やツールが生まれるのだろうか．

（３）概念の先行

最近は概念が先行するが実体が伴わない事が多い．更には概念すら未だ統一的な見解が得られていないものがある．その場合、わけの判らない事を言う人が持てはやされ（僻みか）、その概念は時としてブームを巻起こす．だが、それは５年と持たずに廃れていくことは、過去の事実が証明している．我々は、概念の先行（ある意味では流行）に惑わされることなく、地道に行動することが必要である．

（４）大学・研究機関と企業との背離

ソフトウェア工学の研究開発においては、特定の大企業を除いては、大学や研究機関の方が、はるかに先進的であり研究が進んでいると言って良いであろう．しかし、大学や研究機関で開発されたものが、どの程度現場に流れ、活用されているのだろうか？一つには、それらに関する技術移転のメカニズムが存在しないこと、

一つには、日本の大学の閉鎖性がある．更に挙げるなら、現場のニーズを理解した研究開発が為されていないということも言えよう．

３．生かすために必要なことは？

（１）あらゆる環境に適合するものは無いと認識すること

前述したように、ソフトウェア開発環境の多様化／複雑化はソフトウェア工学における研究開発分野の細分化を促進している．従って、利用者も共通的なものを求めるのではなく、分野（あるいは的）を絞って方法論／技法／ツールに対する要求を出していくようにしなければならない．

（２）産学共同の技術移転メカニズムの確立

昔から一部の分野では産学共同研究が当然のこととして行われてきているが、ソフトウェア工学分野においては、掛け声は高いが、なかなかうまくはいっていないようである．卵が先か、鶏が先かと同じであるが、企業が大学に期待していないのか、企業が出す金と人が充分でないので大学や研究機関でしっかりとした成果が挙げられないのか？ＳＤＡプロジェクト等は、やり方さえ間違わなければ、それなりのメカニズムを充分確立出来ると思うのだが．

（３）マネージメント（リーダーシップ）の重要性

おそらく、企業の各人の認識を変えることや実際の開発環境を個人で変えていくことは極めて難しい．ポイントは、一人でも良いから有能なマネジャを配することであり、それが体制をも含めた環境を変え、ソフトウェア工学を現場で生かす鍵となるであろう．

４．（結論とは言えない）結論

（１）ＡＩは概念であり、夢を追うことは許される．

（２）エキスパートシステムは、ソフトウェア工学の一部として位置付けたい．

（３）オブジェクト指向は正体不明である（と思っている）

（４）ソフトウェア工学は、実践である．ソフトウェア工学を現場で生かす方法について、的を絞った議論を行いたい．