SEAMAIL

Newsletter from Software Engineers Association

Volume 3, Number 5-6-7-8 1988

目　次

ソフトウェア技術者協会（ＳＥＡ）は，ソフトウェア・エンジニアの，ソフトウェア・エンジニアによる，ソフトウェア・エンジニアのための団体であり，これまでに日本になかった新しいタイプのプロフェッショナル・ソサイエティたることを目指して，１９８５年１２月２０日に設立されました．

現在のソフトウェア技術が抱える最大の課題は，ソフトウェア・エンジニアリング研究の最前線（ステイト・オブ・アート）と，その実践状況（ステイト・オブ・プラクティス）との間に横たわる大きなギャップを埋めることだといわれています．ソフトウェア技術の特徴は，他の工学諸分野の技術にくらべて属人性がきわめて強い点にあります．したがって，そうしたテクノロジー・トランスファの成否の鍵は，研究者や技術者が，既存の社会組織の壁を越えて，相互の交流を効果的に行うためのメカニズムが確立できるか否かにかかっています．ＳＥＡは，ソフトウェア・ハウス，計算センタ，システム・ハウス，コンピュータ・メーカ，一般ユーザ，大学，研究所など，さまざまな職場で働く人々が，技術的・人間的交流を行うための自由な＜場＞であることをを目指しています．

ＳＥＡの具体的な活動としては，特定のテーマに関する研究分科会（ＳＩＧ）や地方支部の運営，月刊機関誌（ＳＥＡＭＡＩＬ）の発行，各種のセミナー，ワークショップ，シンポジウムなどのイベントの開催，既存の学会や業界団体の活動への協力，また，さなざまな国際交流の促進等があげられます．

なおＳＥＡは，個人参加を原則とする専門家団体です．その運営は，つねに中立かつ技術オリエンテッドな視点に立って行われ，特定の企業や組織あるいは業界の利益を代表することはありません．

ＳＥＡＭＡＩＬ　Ｖｏｌ．３，Ｎｏ．５～８　　昭和６３年１０月１日発行

編集人　岸田孝一

発行人　ソフトウェア技術者協会（ＳＥＡ）

〒１０２　東京都千代田区隼町２－１２　藤和半蔵門コーブビル５０５

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

特価　２，０００円　（禁転載）

編集部ではなく事務局から

1. 長らくの御無沙汰でした

前号が出て以来，あっというまに時間が経過し，もうオリンピックも終わろうとしています．昨日の陸上４００メートル・リレーで，アメリカ・チームがバトン・タッチの乱れで失格しましたが，このＳＥＡＭＡＩＬも，岸田・藤野両氏のあいだのひきつぎが，まだもたついています（だれですか，２人ともＢ型人間だからなんて悪口をいうのは？）．

何人かの編集ボランティアの方々に御協力をいただいて，いくつかの会合（シンポジウムやフォーラム）あるいはセミナーの記録が，生原稿の形にまではなっているのですが，なかなか，最終的な整理に手がつきません．この夏，新編集長の藤野さんは「若手の会」の実行委員長として，また，前編集長の岸田さんは「信頼性ワークショップ」の世話人として，それぞれ目がまわるように忙しく，まだ，新編集部の第１回ミーティングも開かれていないというのが実情です．

事務局としては，会員のみなさん同様，かなりいらいらしています．しかし，こちらも，ワークショップや中国ツアー，さらには秋のセミナーの準備まで，なにせ１人でかけもちしていますので，とても編集部の尻たたきまでは，手が回りません．

とはいえ，もう我慢にも限度がありますので，とりあえず，つい先だって完成したばかりの「第３回環境ワークショップ（長野）」の報告書を，特別合併号としてお届けします．ワークショップ参加者のみなさんには，すでに報告書が送られていて，まったく内容的にはダブってしまいますが，なにぶんお許しください．

2. さて，これからの見通しは？

とりあえず，２～３号分の生原稿はそろっていますし，新旧両編集長が「明日から２週間の中国旅行中にじっくり打ち合わせをする」と申しておりますので，なんとか軌道に乗るのではないかと，いささか楽観的に考えています．

しかし，まだ，原稿の形にならずに録音テープのままの材料がたくさんあります（たとえば，一昨日終わったばかりの秋のセミナーのテープが２４セッション，各３時間）．どなたか，ボランティアでテープ起こしをしてくださるという方がおられましたら，事務局まで御連絡ください（手紙，電話，ＦＡＸいずれでも結構です）．テキストとテープをお渡しします．

もちろん，その他の原稿もどんどん送ってください．いまの編集部の実態からすると，そのまま写真製版できるようなワープロ原稿（またはレーザ・プリンタ出力）のほうが望ましいと思います．

3. イベントのご案内

全体のページ数が厚くなったついでに，巻末に，こらからのイベント案内のパンフレットをまとめてつけておきました．とりあえず，今月中に募集の締切りが来る（または開催される）のは，

１０月フォーラム
信頼性シンポジウム
SDE ツアー
環境ワークショップ

の４つです．いずれもまだ余裕がありますので，ふるってお申込みください．

4. 会員１５００突破

先日，ちょうど９月の幹事会の当日に，ＳＥＡの会員数が１５００人を越えました．次の目標は２０００人です．ひところに比べて会員数の伸びはやや鈍りがちですが，みなさん，それぞれ身近の方々への入会の勧誘を心がけてください．よろしくお願いします．

第3回 SEA 環境ワークショップ@長野
報告書

1988 年 1 月 27 日（水）～ 29 日（金）

ホテル長野国際会館

ソフトウェア技術者協会

(SEA)

ご挨拶

実行委員長 熊谷章（株）ＰＦＵ

第三回を迎えた環境ワークショップは旅情豊かな信濃の国で開催されました．昨年に続いて今年も真冬というのに雪が見られない長野市でしたが約６０余名の方々が参加されました．まずは，ご参加されご協力頂いた諸兄諸姉の皆様に心からお礼申し上げます．

今年も昨年同様に話題の多いワークショップになりました．回を重ねる毎にワークショップの枠組が大きくなり内容も充実してきたと考えられます．その具体例を幾つかご紹介します．

第一はお隣の中国の上海ソフトウェアセンターから二名の参加者があったことです．このことはＳＥＡは企業を超えた組織ですが，国を超えた活動への第一歩です．特にアジアの技術者との交流は意味深いものがあり感慨もひとしおです．

第二は会場でのツールの実演と説明が実現したことです．これは，出展された企業の方々の熱意とご協力のお陰です．関係された皆様に改めて感謝致します．

第三はポジションペーパの内容が充実していたことです．今年も昨年同様に連続パネルの直列セッションにしたのですが，各セッションの発表者はすべてポジッションペーパの投稿者の中から選びました．質と量の何れもが実践的で一線級になったことを証明しています．

第四は官学民の協同と連帯の定着化です．ＳＥＡを媒介としたこの動きはコンピュータ産業界の動きに一つの大きな影響を与えています．

第五は長野支部の活躍です．ワークショップ運営に関して強力なサポートをして頂ました．今後も各支部との連携でこのような催しを実現したいものです．

次に内容について簡単にご紹介します．今年のテーマは，ＭＭＩの改善，ソフトウェアＤＢ化の試みＷＳ＋ネットワークによる分散環境，開発環境管理の５テーマとツールデモと中国セッションでした．

何れのテーマも少し討論し内容を深めると他のテーマと深く関連していることがよく分かりました．これは開発環境の基礎的なアーキテクチャが今後の研究開発の本質的なテーマであること示しているようです．アメリカでは，開発環境をそのアーキテクチャから次のように分類しています言語指向環境，構造指向環境，ツールキット環境それに方法論立脚環境の四つです．我々も目の前にある開発環境をよりよく使用するだけではなく，ソフトウェアやコンピュータシステムを開発するために必要な開発環境はどのようなアーキテクチャがよいかを考え，そしてそれを開発するる時機に直面しているように思えます．

昨年の目標はアメリカのＰＳＤＥに優るとも劣らない日本版ＰＳＤＥを実現することでした．そのためには，実践的な開発環境の具体例，理論的なアプローチ，会場での実演発表，国際的な交流が懸案事項になっていました．今年は実演と国際交流を実現できました．来年はきっとアメリカＰＳＤＥと比肩できるワークショップになると確信しています．最後に皆様の益々のご活躍と来年のこのワークショップで再会できることを祈願しています．

目　次

第3回 SEA 環境ワークショップ@長野

1988 年 1 月 27 日（水）～ 29 日（金）

ホテル長野国際会館　弥生および桂の間

長野県長野市県町 576

TEL.0262(34)1111　FAX.0262(34)2365

1/27	12:30-13:30	受付	4F 弥生の間入り口にて
	13:30-14:00	オープニング	
弥生 4F	14:00-17:00	セッション1 チェアマン プレゼンテータ	マン・マシン・インタフェースの改善 坂下秀（アステック） 青木淳（富士ゼロックス情報システム） 吉村智香子（ヒラタソフトウェアテクノロジー） 新田稔（ソフトウェア・リサーチ・アソシエイツ）
	17:00-18:00	休憩	
	18:00-19:00	中国セッション チェアマン プレゼンテータ	 熊谷章（ＰＦＵ） 張　然（復旦大学）
藤 2F	19:00-21:00	情報交換パーティ	司会　深瀬弘恭（アスキー）
1/28 桂 B1F	9:00-12:00	セッション2 チェアマン プレゼンテータ	ソフトウェア・データベース化の試み 佐藤千明（長野県協同電算） 岡本隆一（神戸コンピューターサービス） 福田充利（日本電気ソフトウエァ） 堀川博史（三菱電機）
	12:00-15:00	昼食およびツールデモ	昼食を各自で済ませていただいた後，MAC, XEROX1161, Sony/NEWS, Tektronix4406 を会場に設置し，参加者によるツールのデモを行ないます．
	15:00-18:00	セッション3 チェアマン プレゼンテータ	ワークステーション＋ネットワークによる分散環境の構築 中野秀男（大阪大学） 加藤朗（東京工業大学） 酒匂寛（ソフトウェア・リサーチ・アソシエイツ） 西岡健自（横河電機）
	18:00-深夜	BOF	下記のテーマで趣味の合う人が集い，好きな場所で話し合っていただきます．もちろんワリカン．場所等のローカル・アレンジメントは佐藤千明さんをはじめとする SEA 長野支部の方々にお任せしてあります． 中国ソフトウェア事情（中国の方を中心に） MAC:HYPER CARD（佐原さんを中心に） Theoretical Computer Science 夜話（中野先生を中心に） 分散環境の資本主義（加藤さんを中心に） ソフト工学と超能力（岸田，臼井さんを中心に）
1/29 桂 B1F	9:00-12:00	セッション4 チェアマン プレゼンテータ	開発環境管理の実際 臼井義美（日本電子計算） 山浦恒央（日立ソフトウェアエンジニアリング） 田中一夫（山一コンピュータ・センター） 桜井麻里（ソフトウェア・リサーチ・アソシエイツ）

第3回 SEA 環境ワークショップ@長野　参加者名簿

No		氏　名	所　属	Tel
1	実行委員長	熊谷　章	（株）ＰＦＵ　研究開発部	0427(96)5211
2	プログラム委員長	臼井義美	日本電子計算（株）　大坂支店　金融一般営業部	06(448)6022
3	プログラム委員長	林　香	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
4	プログラム委員	岡本隆一	（株）神戸コンピューターサービス　ソフト開発部	078(391)8291
5	プログラム委員	坂下　秀	（株）アステック　社長室	03(477)1541
6	プログラム委員	佐藤千明	長野県協同電算（株）　開発課	0262(28)3215
7	プログラム委員	中野秀男	大阪大学　工学部 通信工学科	06(877)5111 ex.4773
8	プログラム委員	深瀬弘恭	（株）アスキー　システムソフトウェア事業部	03(486)1207
9	プログラム委員	藤野晃延	富士ゼロックス情報システム（株）　技術部	03(378)8010
10		葉　建敏	上海計算機ソフトウェア技術開発センター　国際合作部	377934
11		張　然	復旦大学　計算機科学学科	484852
12		久保宏志	富士通（株）　システム本部 パッケージ企画統括部	03(437)5111
13		岸田孝一	（株）ソフトウェア・リサーチ・アソシエイツ　専務取締役	03(234)2611
14		御喜家貴子	横河ヒューレット・パッカード（株）　マーケティング部門	03(335)8239
15		堀川博史	三菱電機（株）　情報電子研究所　システム・ソフトウェア開発部	0467(44)9084
16		伊野　誠	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
17		鶴見泰久	（株）東海クリエイト　VAR 推進部	03(456)4611
18		山浦恒央	日立ソフトウェアエンジニアリング（株）　第１技術本部 第５設計部 第４課	045(824)2111
19		森　幸一	（株）ＰＦＵ　開発企画部 ソフトウェア検査課	0427(96)5211
20		吉村智香子	ヒラタソフトウェアテクノロジー（株）	096(371)7300
21		村川貴広	ヒラタソフトウェアテクノロジー（株）	096(371)7300
22		西岡健自	横河電機（株）　研究開発 4 部 第 1 研究室	0422(54)1111
23		盛田政敏	（株）神戸コンピューターサービス　ソフト開発部長	078(391)8291
24		斎藤信男	慶應義塾大学　理工学部 数理工学科	044(63)1141 ex.3710
25		和田　勉	長野大学　産業社会学部	0268(38)2350
26		新田　稔	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
27		佐原　伸	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
28		福田充利	日本電気ソフトウェア（株）　OA システム事業部 第三開発部	03(444)3211
29		落水浩一郎	静岡大学　工学部情報工学科	0534(71)1171
30		三浦あさ子	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
31		歌代和正	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
32		酒匂　寛	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
33		桜井麻里	（株）ソフトウェア・リサーチ・アソシエイツ　環境開発部	03(234)2611
34		田中正則	（株）ソフトウェア・リサーチ・アソシエイツ　システム開発本部 開発第一部	03(234)2611
35		北野義明	（株）神戸コンピューターサービス　ソフト開発部技術第一課	078(391)8291
36		田中一夫	（株）山一コンピュータ・センター　開発第二部 基本システム課	0474(37)3183
37		田中慎一郎	（株）ソフトウェア・リサーチ・アソシエイツ　ソフトウェア工学研究所	03(234)2611
38		小池幸徳	日本システムサイエンス（株）　システム部 システム技術課	03(352)9551
39		青木　淳	富士ゼロックス情報システム（株）　技術推進室	03(378)8010
40		鐘　友良	富士ゼロックス情報システム（株）　企画課	03(378)8010
41		古田とおる	富士通ビーエスシー（株）　商品企画開発部 開発課	03(544)0506
42		荻生準一	日本システム（株）　府中事業所 第２システム技術部	0423(60)5811
43		野中　哲	日本電気（株）　マイクロ波衛星通信事業部	045(931)2503
44		市川　寛	（株）電算　企画部	0262(34)0151
45		堀内泰輔	長野県工業高等専門学校　機械工学科	0262(95)5529
46		井上尚司	ソフトバンク総合研究所　技術部	03(818)7531
47		小林明彦	東電ソフトウェア（株）　技術開発部	03(592)1311
48		中村　眞	シャープ（株）　技術本部　コンピュータシステム研究所	07436(5)1321
49		近藤康二	ソニー（株）　スーパーマイクロ事業本部ワークステーション事業部	03(448)4030
50		井川裕基	日本電子計算（株）　ファイプロジェクト	03(668)1334
51		加藤　朗	東京工業大学　工学部　情報工学科	03(726)1111 ex.2566
52		海尻賢二	信州大学　工学部　情報工学科	0262(26)4101
53		寺井　孝	（株）日立ビジネス機器　技術部	03(773)2111
54	レポータ	塚田道明	（株）電算　システム本部	0262(34)0151
55	レポータ	小林貞幸	（株）電算　企画本部	0262(34)0151
56	レポータ	石坂幸一	ケイケン長野データセンター　システム２部	0262(28)6644
57	レポータ	細野広水	エム・ケー・シー	0263(26)8220
58		吉村鐵太郎	（株）管理工学研究所　社長	03(405)1423
59	報道	下田博次	フリー・ジャーナリスト	0263(83)5284
60	事務局	杉田義明	（株）ソフトウェア・リサーチ・アソシエイツ　企画部	03(234)2611
61	事務局	中島千代子	SEA	03(234)9455

セッション1
マン・マシン・インタフェースの改善

チェアマン　坂下秀（アステック）

プレゼンテータ　青木淳（富士ゼロックス情報システム）

吉村智香子（ヒラタソフトウェアテクノロジー）

新田稔（ソフトウェア・リサーチ・アソシエイツ）

－ 内容 －

- 事前アンケート
- プレゼンテータのポジション・ペーパ
- セッション・レポート
- 事後アンケート

今や開発環境をささえる土台は、ＴＳＳ＋キャラクター端末の時代から、ワークステーション＋ビットマップ・ディスプレイの時代へと移り変わりつつあります。ウインドウ・システムもいくつか登場し、どれがメジャーになるか興味が持たれています。このような変化に伴い、従来のツールのユーザ・インタフェースを改良したり、斬新なアイデアに基づいたマン・マンシン・インタフェースを基礎にして、開発環境を再構築するなどの活動が行なわれているものと思われます。このセッションでは、マン・マシン・インタフェースのアイデアの提示や、ユーザ・インタフェースの改良事例の発表などを通じて、人とマシンとの『親和性』の本質に迫れるような議論ができれば、と考えています。

熊谷　章／ＰＦＵ　研究開発センター第三開発室

◆マルチメディア情報の MMI

マンマシンインタフェースはマルチメディア情報を介してスムーズなコミュニケーションを実践している。マンマシンインタフェースもこの境地に近づくべきた。カラー、図形、絵、音はコミュニケーションの重要な部分を占めている。そして、今やこれらのマルチメディア情報は様々なアプリケーションで使用され始めている。マルチメディア情報のデータ特性、データ構造、データの意味等の考察を通して、新しい MMI の枠組とそれを作り上げるためのアーキテクチャを考えたい。

◆オブジェクト指向プログラミング環境

議題に対する一つの解がオブジェクト指向プログラミング環境である。Smalltalk を用いたアプリケーション開発やプロトタイピング用の PWB システム開発の例から、これからの計算機ソフトウェアにおけるオブジェクト指向プログラミングとユーザインタフェースに関して考えたい。

岡本隆一／神戸コンピューターサービス　ソフト開発部技術第一課

◆技術者に最適化するマン・マシン・インタフェースの改善

一言でソフトウェア技術者と言ってもデザイナからプログラマまで様々である。マン・マシン・インタフェース・セットアップ・ツールの必要性、具体化について議論したい。

◆コンピュータからのメッセージ…、フレンドリーと軽薄さの境目…？

コンピュータから利用者への要件伝達は、メッセージ＜文字、図形、イメージ、…、＞によることが多い。厳格過ぎて意味不明も困るが、フレンドリーの解釈をひとつ誤ると軽薄の世界。また、方言の増加で新たな問題も顕在化してくる＜予感＞。

中野秀男／大阪大学　工学部通信工学化

マルチウィンドゥのマルチタスクを学生に１台づつ与える環境づくりがまずやりたいことなので。なるべく楽しくなる MMI を考えています。営利団体でないので。でもそんな学生が普通の会社に就職したら問題だな。(ソフトウェアの開発環境とは関係ありませんが、計算機を触ったことのない事務屋さんにも、抵抗なく使ってもらえるのではないかと思います。)

深瀬弘恭／アスキー　システムソフトウェア事業部

◆現在の WS のインタフェースの問題点は？

ビットマップ＋マウスの組合わせのハード構成で当面いけるのか？ソフトウェア側の問題はしだいに改善されるし、選択の余地があるが、ハード構成はなかなか変更出来ないので。

藤野晃延／富士ゼロックス情報システム　技術部

◆ MMI のモデル化

MMI のモデルとしてどのようなものがあるのか。Standard としての MMI は、どうあるべきか、その基本的機能の階層を決定できるのか？

◆ UIMS の動向

UIMS(User Interface Management System)として、Application independent-plugable な U/I はどうあるべきか。

久保宏志／富士通　システム本部パッケージ企画統括部

◆ MMI のアプリケーション独立性(概要・整理)

◆ MMI のアプリケーション独立性(標準化の動きと、その展望)

岸田孝一／ＳＲＡ

◆ MMI 改良のアイデアをユーザが簡単に実現(または実験)できる仕掛けは可能か？

これからは、いろいろな環境やツールがどんどん作られる。

環境の開発者や管理者の立場からすれば、それらの

MMI をなるべく統一したい。一方、ユーザの立場からすれば、自分なりにチューニングしたい。

この２つの要求を同時に満足させることは可能だろうか？

◆ビットマップとウィンドウの将来？

自分ではまだ本格的に使っていないので、経験者の方々の意見を聞きたい。

御喜家貴子／横河ヒューレット・パッカード　マーケティング部門

◆グラフィックスなどを使って、わかりやすい（推測されやすい、イメージできる）インタフェースがよいのでは？

１人１台ワークステーションが供給されるようになれば、人間の思考スピードとコンピュータの対応スピードとのギャップはどんどん改善され、ごくごく自然なものになってくると思うが、使いがって、という面で工夫してゆくことが次の目標と思う。たとえば、各種アプリケーション間でのマン・マシン・インタフェースが一つに見えるような仕掛、コンピュータとユーザの会話の部分へのグラフィックスの導入などはどうでしょうか。

伊野　誠／ＳＲＡ

◆徹底したヘルプ機能を用意すべし

最近 UNIX 上の Emacs や Symbolics の Genera 環境を使ってみて Help 機能、使い方のオンライン説明／ガイドが良いのに感心している。UNIX にも man があるがあまり使い易いものではない。今後、ツールの開発者は初心者からプロまでの数段階の利用者に対し徹底したオンライン・ガイダンスおよび Help 機能を組み込むべきと思う。

鶴見泰久／東海クリエイト　VAR 推進室

◆マン・マシン・インタフェースの関点

マン・マシン・インタフェースについて、さけばれているが、それは基本的操作面のソフトが多い。現実に末端のエンドユーザの開発取組やすさの問題を考えなくては、ウインド、日本語処理、編集等だけでは、一見の商品価値だけしかなく、企業内の EDP 部門の悩みは解決しない。

山浦恒央／日立ソフトウェアエンジニアリング　第１技術本部第５設計部第４課

◆誰にも使いやすいマン・マシン・インタフェスは可能か？

UNIX は初心者にとって非常に不親切な OS であるが、上級者には使いやすい。また一方銀行の現金引出し端末は初心者にとっては親切なものだが、慣れた人々は非常にまわりくどい。このようにマン・マシン・インタフェスは使う人により大きく評価が異なるが、誰にとっても使いやすいインタフェスは可能か？使い手とともにまた使い手の学習レベルと共に進化するインタフェスは存在するものだろうか？

森　幸一／ＰＦＵ　開発企画部ソフトウェア検査課

◆ヘルプ、オンラインマニュアルのあり方

印刷されたマニュアルを読まなくても、とにかく使ってわからなければ、そのときシステムに聞いて先に進むといったマン・マシン・インターフェースの実現はヘルプ、オンラインマニュアルの工夫で充分可能ではないかと思う。

村川貴広／ HST

◆マン・マシン・インタフェースの基本

マン・マシン・インタフェースには、人及びマシンが変わればそれぞれのインタフェースが存在すると思います。しかし、どのような人、マシンであっても、基本的に一致するインタフェースはあるような気がします。いったいそれが何であるのか、そして、その基本的なインタフェースをベースに、人、マシンが変わった時にどのようなことを考えていかなくてはならないのかを討議していきたい。

西岡健自／横河電機　研究開発４部第１研究室

◆構造化マンマシン・インタフェース

マルチウィンド、マウス、アイコン、ポップアップメニュ…とマンマシン・インタフェースの動向は操作の自由度を追求しているように見える。

かつて、JIS フローチャートがその自由度の故に多くの悲劇を生んだと同様、近い将来マンマシン・インタフェースにおける新たなソフトウェアの危機が到来するのではないかという懸念を持つ。今のうちにコトバの上だけでも“構造化マンマシン・インタフェース”を提唱しておきたい。

斎藤信男／慶応大学　数理工学科

◆マルチメディアへの展望

マルチメディアにしたときに、人間の創造性にどれだけ寄与するか。かえって、使いにくくなるのか。昔ながらのキーボードの方が良いのか(マウスとキーボードの両方を操作するのは以外と面倒である。)

◆マンマシンインタフェースの自動生成

マンマシンインタフェースを、ユーザの好みに応じて生成することは、必ず必要になる。

和田　勉／長野大学　産業社会学部

これは半分は他人の受けうり、もう半分も急にいい加減に思いついただけのものですが…

今普及しつつあるマウスやペンタブレット等のポインティングデバイスは、言うまでもなく画面上の何か、またはどこかを直接示すためのものです。これは例えば従来のエディタ(vi など)のようにキーボードからの操作でカーソルを動かして示すことしかなかったことから比べれば大きな飛躍ですが、マウスやタブレットは机上にあり、指すべき点はディスプレイ上にある、というギャップがまだ残っていると言えるのではないかと思います。

この点をもう一段飛躍したハードウェア、というアイデアはいかがでしょう。具体的には、ゲームセンタ用のテレビゲーム台のような形でビットマップディスプレイを机に水平にはめ込み、上にタッチセンサないしは透明なペンタブレットをかぶせます。その上を指またはタブレット用ペンで触れるとクリック、画面に指先やペン先を触れたまま引きずるとドラグ、というぐあいです。手書き文字認識技術を用いて、ペンで文字を各とそれがその位置に入力される、となるともっといいわけですが。

ソフトウェアの開発環境とは関係ありませんが、計算機を触ったことのない事務屋さんにも、抵抗なく使ってもらえるのではないかと思います。

新田　稔／ＳＲＡ

◆利用者をよく理解しない

そのツールなりアプリケーションなりが使用される環境をよく知っているのは、そのユーザであり、また解こうとしている問題について、知識を多く持っているのもユーザである、という考え方でツールを作っていきたい。

佐原　伸／ＳＲＡ

◆ Mac 対 UNIX

ユーザーインタフェースの代表として、パーソナル環境の Mac と、複数の人間が使う UNIX は、どちらも利点・欠点がある。

この２つの方式を統合する概念を考えられないだろうか？

福田充利／日本電気ソフトウェア　OA システム事業部第三開発部

◆ユーザパラダイムの段階的実験としての MMI

「使い易い」ということはユーザパラダイムと操作の対応がシンプルだということだろう。ユーザが対象分野について持っているモデルとパラダイムに、操作のために付け加えなければならないモデル要素／パラダイムをいかに少なくするかということだろう。

歌代和正／ＳＲＡ環境開発部

◆速いマシンをどうやって休ませないか？

今までのソフトウェアの技術は限られた資源をどのように有効に利用するかということだった。しかし、ハードウェアの技術は着実に進歩しているため、あり余る資源を手に入れる日が来るだろう。その時代のマン・マシン・インターフェースとは、あり余る CPU 資源を利用して、いかに頼まれないことまで考えるということだと思う。やってほしいと思うことをやってしまっても十分ヒマなのだから。

酒匂　寛／ＳＲＡ環境開発部

◆機能統合とマシンインターフェイス

効果的な環境は、そのマンマシンインターフェイスのできによること大である。しかしながら、現時点での議論の中心は、『マルチウィンドウが良い』とか、『マウスが必要』といったレベルにとどまっている(場合が多い)。マルチウィンドウ、マウスなどが一般的になりつつある今、やっと概念レベルの議論を実践的な見地から行なえるようになった気がする。ここでは環境に機能を統合していく際に必要なアーキテクチャと、そこに与えられるマンマシンインターフェイスの形について議論したい。

桜井麻里／ＳＲＡ　環境開発部

◆現存の環境を少しでも良くするために

人間はある一つの世界に入ると、なるべくその世界から出たがらない性質をもっている。そういう人間のために、今あるばらばらなツールやユーティリティをある一つの世界の見方でまとめあげる努力が必要ではないか。そして、一つ分かれば、みんなわかるというようになれば、少しでも気持ち良く使えるのではないだろうか。ビットマップディスプレイやウィンドウシステム、ネットワークシステムや分散 OS は、そのための手段でしかない。まずは、既存のツールやユーティリティを、できる範囲でいいから、一つの世界にまとめてみよう。できるだけかわいく!

◆わかりやすいユーザ・インタフェイス構築のために

どうすればやりたいことができるかということが、一目でわかるようなものが望ましい。そのためには、いろいろと作って試して見る必要がある。しかし、現在のウィンドウシステムでは、"とりあえず作って見る" というのがむずかしく、"とりあえず作った" だけで疲れて終わってしまう。いろいろと作り替えて試してみるには、ユーザ・インタフェイス構築のための使いやすいツールキットの登場が望まれる。そのツールキットは、レイアウトばかりでなく、動きも定義できることが望ましい。そして、そのツールキットを使ったアプリケーションをどんどん作って世に送り出すことが重要である。

田中正則／ＳＲＡ　開発第１部

◆ WYSIWYG 方式か？バッジ方式か？

文章清書ツールの U/I の方式に WYSIWYG 方式とバッジ方式がある。では、どちらを選ぶかというと、どちらにも長所、短所があると思う。従って、これからの文書清書ツールの U/I は、どちらにも対応しているものが望ましい。

北野義明／ KCS　ソフト開発部技術第一課

◆利用者各人にとっての操作の簡易性とは

利用者は、一口にソフトウェア開発に携わる人と言ってもコンピュータに触れるのが全くの初心者から、優秀な開発者、運用管理者等、様々なレベルの人達である。人によって使用しようとする機能にも差がある。又マニュアルやディスプレイ表示される説明についても、読んでもわからない人、読めば何とかなるひと、読んでも充分な説明がない(もっと奥深く知りたい)という人等、ガイダンスのレベルにも配慮しなければならない。多機能、高性能な WS を幅広く利用できるような MMI について考えたい。

◆人間、社会、文化と MMI

1.スイッチ ON といえば、"XXXX" のように一般社会で、あるいは人の感性としてなじみのある(TRON のような)考え方。

2.ファミコン・ブームや学校へのコンピュータの普及、一方高齢化社会等、コンピュータを取り巻く世間の動向。そして高度情報化社会への確実な動き。

3.知識、集約産業とやばれながら、残業、残業、…、人手が足りない等、現場の泥臭さのギャップ。 ～ MMI への基本的な要求が見えてこないだろうか？

田中一夫／山一コンピュータ・センター　開発第二部　基本システム課

◆総合化エントリ

マルチ・ベンダーとなりエディタからして根本的に違いがあり、利用者にとっては非常に使いづらさを感じる今日このごろです。 そこで、 YCC では開発標準手順(AURORA)にそった統合化エントリなるものを構築して使用するコンピュータに関係なく同じ様なやり方で仕事が出来るようにしようとしている。

参考資料: AURORA 開発工程と開発支援分野、開発支援全体構成図

田中慎一郎／ＳＲＡソフトウェア工学研究所

◆マン・マシン・インタフェース向上の目的とは

単純に言ってしまえば、操作が楽になることにより、生産性が上がるというような話になると思われるが、本当は楽になることよりも、ある程度の水準での正確さが保証されることではないかと考える。コンピュータに限らず、一般に操作が楽なものはアマチュア用で色々難しいことを知っていて、それがうまく操えてこそプロだというような認識がある。カメラの世界でも、EE カメラなどはアマチュアの使うもの、プロはマニュアル機を使うのだという話もあるが、一方では EE カメラの登場を最も喜んだのはプロだと言っている人もいる。それはやはり、「楽だ」ということだけでなく、一定水準の正確さが保証されているからだと思う。表面的には取り立てて違いはないかも知れないが、実際にその方面に携わっていらしゃる方々のお考えをおききしたい。

小池幸徳／日本システムサイエンスシステム部システム技術課

◆可視的開発環境の構築

従来、紙と文字により行なわれてきた開発は、マウス、ビットマップディスプレイ、LBP と行ったすぐれたマン・マシン・インタフェースを利用することにより、画面上で対話的に行なうことが可能になり、設計、コーディング段階で記述される様々なダイアグラムを容易に画面上に実現し、可視的表現により物理的、論理的にターゲットシステムとの整合性が取れるようになるだろう。しかし、この場合、開発工程で様々なダイアグラムが利用されるが、この上位レベルにおいて必ず、首尾一貫性が保たれていなければならない。

青木　淳／富士ゼロックス情報システム技術推進室

◆コマンドをキータイプして RETURN キーを打つのは

やめましょう

ビットマップ・ディスプレイ＋マウスを装備したワークステーションの時代になり、ウィンドウシステムが登場しても、コマンドをキータイプして RETURN を打つという TTY 端末のインタフェースは依然として残っています。少なくともこのインタフェースは過去のものにしましょう。

鐘友　良／富士ゼロックス情報システム　企画課

◆自然言語のインタフェース

人とマシンの交流については、1つは、思想の方法、例えば、オブジェクト指向の方法は両方に近い。一つは交流の方式、これは、自然言語を使えば、理想的なものであろう。今の状態で、メニューに基づく自然言語インタフェースは現実的な方法として興味を持っている。

古田とおる／富士通ビーエスシー　商品企画開発部開発課

◆マウスについて

マウスは現在のところ使いやすいポインティングデバイスと思われますが、他により使いやすいものは今後どのようなものか考えたいと思います。

◆ Smalltalk、Mac

Smalltalk や Macintosh のような、マン・マシン・インタフェースは今後どのような形で発展していくか？

野中　哲／日本電気マイクロ波衛生通信事業部

何か書きたいのですが、pc-9800 しか使ったことがないのでどうしようもない。 ビットアップの端末を使ってみたい。近々、EWS-4800 が来るはずなので、感想はその後で。

◆日本語の入力について

「一太郎」など、かなり良く出来ていると思うが、更に一層頑張って欲しい。NEC 社内では、4.3bsd 上に、日本語エディタがあるが、これがどうしようもない。UNIX 上での標準的な、かな漢字変換が出来て欲しい。(ローマ字ではない) UNIX の世界で、どうやってカナキーボードを扱うか？？

堀内泰輔／長野工業高等専門学校機械工学科

◆キーボードの運命は如何に？

情報処理教育の立場で、キーボード教育を行なう必要があると思うが、マウスの進展いかんではキーボード不用となることはないのか？

◆日本語入力時のキーボード配列はいかにあるべきか？

最近、NTT で開発された、SKY 配列を練習したのだが、それ自体非常に良く出来ているが、旧来の QWERTY 配列がうまく扱えなくなってしまうという不安もある。また、ローマ字入力か、カナ入力かという、古くて新しい議論も、ソフトウェア技術者の立場で行なってほしい。

井上　尚司／ソフトバンク総合研究所

私が直接使うのは、Sun のコンソールと PC-9801 の 2 種です。いまだに PC-9801 を使うのは日本語入力の点からで、Wnn にしろ SJ にしろ、今いちだなあと感じてしまうのは...ぼくだけかな？ PC-9801 のソフトみたいにアクロバット的、かつ MS-DOS プログラム的なソフトを UNIX でも実現しないと、いつまでも 98 を使わなくてはいけないのかなあ。98 の「アッ」という間の変換にもそろそろウンザリしている今日このごろ。

小林明彦／東電ソフトウェア

◆社内ユーザへのユーザインタフェースの提供レベル

OA の一貫として、エンジニアも、一般職も、役印も同じシステムを使用する際、ユーザの区別なく、ユーザインタフェースは最適なものが存在するのか？もし、存在しないとしたならば、ユーザのレベルに応じたユーザインタフェースを提供すべきなのか、どちらの側に立脚して考えたらよいのか？逆に御意見をうかがいたい。

河田　亨／シャープ

◆日本人にキーボードは必要か

アルファベットを母国語とし、タイプライタが一般的な国々と異なり、日本人には近年ワープロにより一般にまでキーボードが目にふれるようになったが、はたして物まね的に導入してよいものであろうか。コンピュータのプロフェッショナルならキーボードは当然とも言えるが、あえて考え直してみたい。

近藤康二／ソニースーパーマイクロ事業本部ワークステーション事業部

◆ウィンドウ、マウスなどにあったインターフェイスを

ウィンドウ、マウス等によるユーザーインタフェイスは、今や常識になりつつある。しかしながら、依然としてソフトウェアが追従していないように思う。例えば、UNIX が良い例で、“ls”、“cat”、“cc”等がとても考慮しているとは思えない。ウィンドウのサイズを決めた後に、その中で上記のコマンドを走らせた場合、ウィンドウの情報

に情報が消えていって慌てた経験をお持ちの方が多いと思う。これはまさに、そのコマンドがウィンドウというものを全く意識していない証拠である。従って、スクロールバー等という逃げの手段ではなく、いくつかウィンドウを意識したソフトウェアを試作してみる必要がある気がする。

井川裕基／日本電子計算開発本部ファイプロジェクト

◆日本語入力ツール

NEWS の SJ2 に馴染めず、PC + VJE が手放せません。Wnn や JNIX の話を聞き、デモを見てもあまり良くわかりません。実際に使って(作って？)おられる方のお話しを直接聞けたら。。

加藤　朗／東工大情報工学部

◆マンマシンインターフェースの改善

キーボードの配列を変えたくらいでは、画期的なマンマシンインターフェースの改善を望むことはできない。たとえば、脳波と発音、視線などと、その人が考えていることとの関連性が明かになればいいのだが。

計算機の使いやすさや使いにくさを定量的に評価かすることができれば、もっとおもしろいことができるだろうなあ。もっとも、計算機が使いにくくても、自分の(昔に作った、あるいは人に作らせた)プログラムが速く動さえすれば良い、と思っている人が多いうちは、計算機のパフォーマンスを多少犠牲にしても、使い易くすべき、ということのコンセンサスが得られないような気がする。Unix やウィンドウシステムの普及で、少しはマシになったと言うべきか。

中村暢夫／日立ビジネス機器技術部

◆運用方法について

ハードの進歩にソフトないし運用が追いついていないのが、現状であろうかと思います。使いこなしているという事例を収集したい。

3rd SEA Environment Workshop Position Paper

氏名	青木 淳	種別	□会員 no □一般
所属	技術推進室 第2グループ	富士ゼロックス情報システム(株)	
住所	〒(160) 東京都新宿区西新宿3-16-6 (西新宿水野ビル)		
TEL	(03) 378-8010	FAX	(03) 378-7298

賛助

仕　事

Smalltalk-80 によるアプリケーション開発

- ユーザインタフェースの改善
- ソフトウェア設計開発支援ツール
- Unix と Smalltalk-80 の結合

作業環境

パーティションが付いた 180cm(横幅) の机.

XEROX 1161 ワークステーション (プログラム開発用)

JStar ワークステーション (ドキュメント作成用)

} 専有している.

ワークステーションは ビル全体のネットワーク および 国内外のネットワークに接続している.

現在の問題

Smalltalk-80 でプログラムすることが難しい。

オブジェクト指向の設計を Smalltalk-80 プログラムに反映させること.

数人で Smalltalk-80 プログラムを作成する時に生じる問題.

次ページの討論テーマ票にも，必要事項をご記入の上本紙とあわせて 12 月15 日まで（締め切り厳守！）に下記のSEA 事務局までお送りください．もちろん，本式の Position Paper も同封していただくほうがベターです（既発表の論文を添付していただいても構いません）．

ソフトウェア技術者協会 (SEA) 〒102 千代田区隼町2-12　藤和半蔵門コープビル505

TEL: 03(234)9455　FAX: 03(234)9454

Smalltalk-80による視覚的UNIX環境

富士ゼロックス情報システム(株)
青木　淳

1. はじめに

ソフトウェアの開発環境は、TSS+キャラクタ端末の時代から、ワークステーション+ビットマップ·ディスプレイの時代へと移行しています。実践的ソフトウェアの開発環境として評価の高いUNIX環境は、TSS+キャラクタ端末の時代の産物です。このUNIX環境をSmalltalk-80を利用して、ワークステーション+ビットマップ·ディスプレイの時代に適合するようにしたいと考え、作成したのが「Smalltalk-80による視覚的UNIX環境」です。

2. 視覚的UNIXインタフェースの必要性

UNIXのユーザ·インタフェースを一手に引き受けているシェルは、従来のTTY端末を仮定したインタフェースで構築されています。高解像度ビットマップ·ディスプレイやマウスを装備したワークステーションが普及しつつある現在でも、この古いインタフェースが使用されています。

また、UNIXの中に装備されているテキスト·エディタは、vi,emacsであり、豊富な機能を持っているのですが、キーボードを主体としたインタフェースであるがゆえに、使いこなすには相当の訓練が必要です。

それから、UNIXの特徴である階層ファイル·システムは非常に強力ですが、利用者は階層ファイル·シルテムの構造を、常に頭の中に覚えて作業を行わなければなりません。つまり、自分のワーキング·ディレクトリが現在どこなのかを、いつも意識しなければならないわけです。

もっと視覚的で認識性が良いインタフェースが望まれます。この「Smalltalk-80による視覚的UNIX環境」は、Smalltalk環境に慣れ親しんだ技術者が必要にせまられて作成したものです。ソフトウェアの開発効率を高めるために、Smalltalk-80をユーザ·インタフェースに使用しています。そのため、視覚的で直接的な操作が可能であり、プログラミングの時間を大幅に短縮します。

3. 視覚的UNIX環境の特徴

(1) 視覚的な操作環境

Smalltalk-80のユーザ·インタフェースと同様に、マルチ·ウインドウおよびマウス·オペレーションによる操作が基本です。操作の対象となるものを選び、それに対する操作をメッセージで送信するというオブジェクト指向のインタフェースで統一されています。これは利用者にオブジェクトを直接操作しているという感覚を与えます。

(2) 拡張可能な開発環境

この視覚的UNIX環境はSmalltalk-80で記述されていて、そのソース･プログラムがすべて提供されます。そのため、利用者は自分の好みの環境に改良したり拡張したりすることができます。

(3) Smalltalk-80とUNIXの結合

富士ゼロックス1161ワークステーションは、基本OSにUNIXを搭載しています。このUNIXとインタフェースするためのクラスがSmalltalk-80の中に用意されています。これを利用して、UNIX上の定評のあるツール群をSmalltalk-80の優れたユーザ･インタフェースを通して使用できます。

(4) 強力なグラファ･ライブラリ

この視覚的UNIX環境の中には、ディレクトリの階層構造をグラフィカルに表示するグラファがあります。このグラファはノード間の関係をグラフィカルに表示できる機能であり、階層構造を持つものやオブジェクト間の論理関係を視覚的に表示する汎用ライブラリ･パッケージです。

4. 開発ツール

(1) UnixFileEditor

Smalltalk-80の高度なエディタをUNIXのテキスト･ファイルの編集のために専用化したものです。カット･アンド･ペーストによるテキスト編集、スクロール･バーによるページ参照、括弧の対応を白黒反転させるパラグラフ処理など、視覚的な編集作業を行うことができます。また、マルチ･ウインドウであるため、複数のエディタを起動して、他のファイルの内容を見ながら、編集作業を進めることができます。

(2) UnixFileTree

UNIXの階層ファイルシステムの構造をツリー状に表示できます。mkdir,rmdir,lsなどのコマンドも、このウインドウのポップ･アップ･メニューから行うことができます。

(3) UnixFileHolder

ディレクトリの中に存在するファイルの一覧を表示します。パターンによるファイルの絞り込み、モード/オーナ/日付などのファイル･ステータスの表示、このディレクトリの中に存在するサブ･ディレクトリのスポーニングなど豊富な機能を有します。

(4) UnixFileLister

UnixFileTreeとUnixFileHolderを統合したツールです。ツリー状に表示されたディレクトリを選択するだけで、そのディレクトリに格納されているファイルの一覧を得ることができます。

(5) UnixFileMover

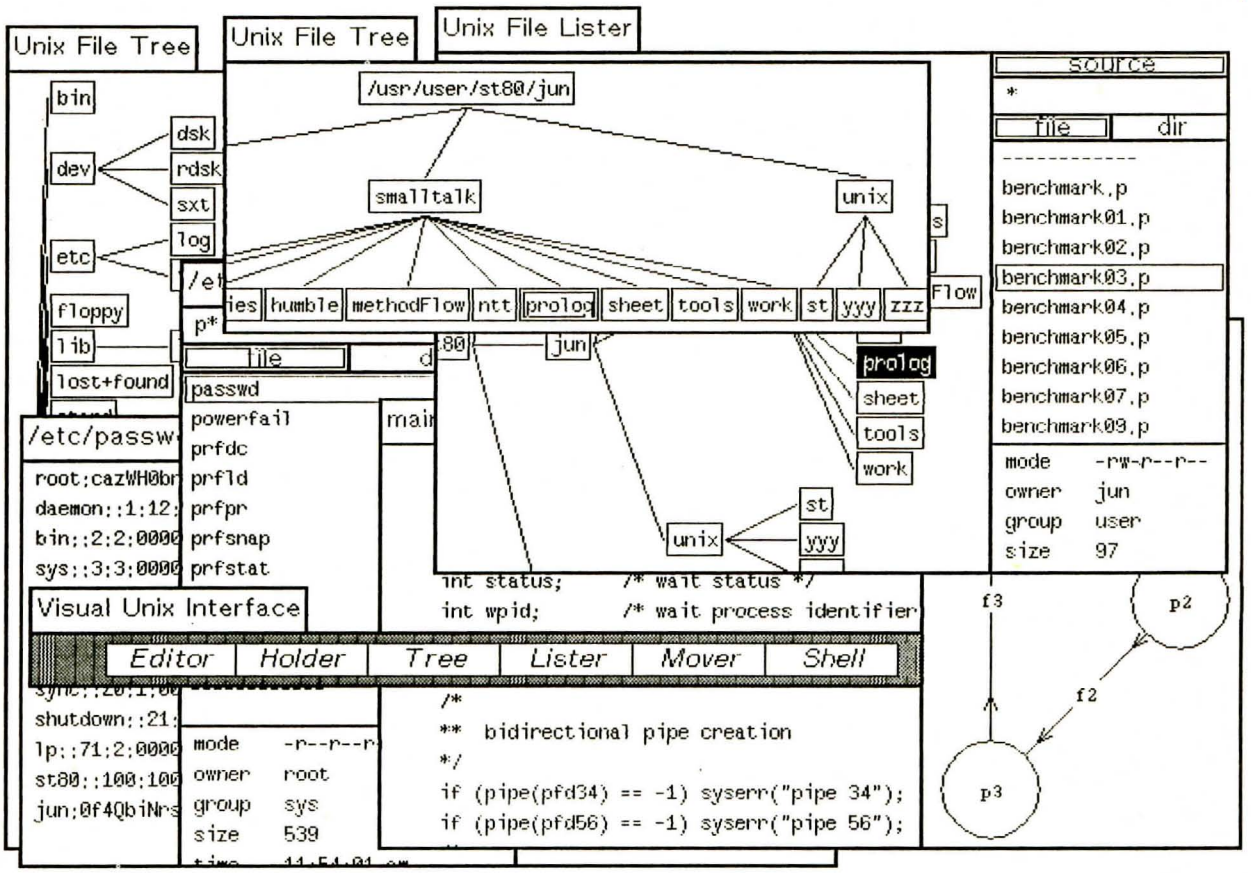
UnixFileTreeと2個のUnixFileHolderを統合したツールです。ファイルの転記/移動を専門に行います。

(6) ShellUnix

UNIXのコマンドを実行できます。コマンドの実行の履歴を蓄えるヒストリ機能とコマンド･ラインのエディタを装備しているので、以前に実行したコマンドの再実行や、タイプ･ミスしたコマンドの修正が容易に行えます。

(7) VisualUnix

上記のツール群を有機的に統合します。編集したファイルや、参照したディレクトリを保持し、それらを編集できるので、ツール･マネージャとして最適です。



5. おわりに

UNIXの特徴は、「ファイル･システム」と「プロセス」です。この「Smalltalk-80による視覚的UNIX環境」は、現在「ファイル･システム」に重点を置いています。これからは、UNIXのもうひとつの特徴である「プロセス」を視覚的にしたインタフェースを作成し、より強力なソフトウェア開発環境にしたいと思います。また、ソフトウェア設計のための視覚的なツールも開発したいと考えています。

氏名　吉村　智香子

種別　一般

所属　ヒラタソフトウェアテクノロジー（株）

住所　８６２　熊本氏渡鹿６－１－４５

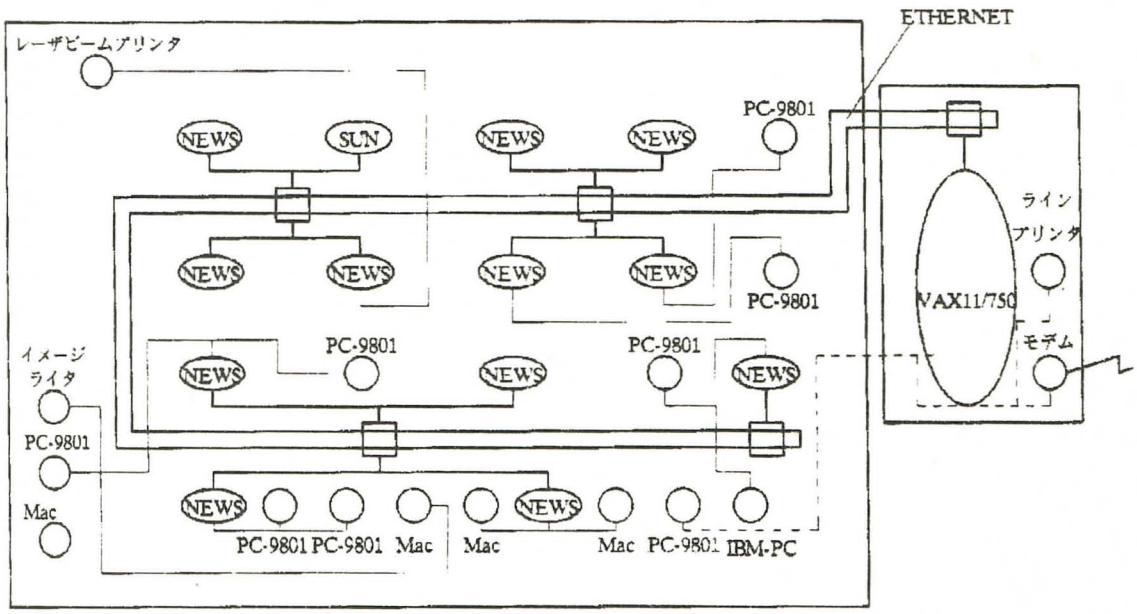
TEL　096(371)7300

FAX　096(363)3253

仕事　主にＦＡ関連業務

作業環境

ＨＳＴのハードウェア環境



NFS により分散環境が構築されている。

現在の問題

・発散環境
・よりよいユーザーインターフェースの実現
・データベース化への歩み

実戦的ソフトウェア開発環境に関する集中討論
(3rd SEA Environment Workshop)
マン・マシン・インターフェースの改善に関する
ポジション・ペーパー

ユーザーインターフェースに関する一考察
—ロボットプログラミング支援システムを通して—

ヒラタソフトウェアテクノロジー（株）
Hirata Software Technology Co., Ltd.
吉村　智香子
chika@hstvax.hst.junet

1. はじめに

弊社は自動機省力機メーカー平田機工よりソフト部門の強化を目的として独立した会社であり、まだわずかに１歳である。現在、ＦＡ関連業務およびグラッフィック関連業務を中心にやっている。従来、平田機工のＦＡ部門においてロボットプログラムラミング支援システムや数々の生産管理システムが作られていた。現在、弊社においてロボットプログラムラミング支援システムや生産管理システムが新たに構築されている。

最近、IBM-PC 上に構築されてきたロボットプログラミング開発支援システムを Macintosh 上に再構築する仕事に携わる機会を得た。 Macintosh 版のユーザーインターフェースは IBM-PC 版とはまったく異なっており、ユーザーにとってよりなじみやすいものとなっている。本ペーパーでは、ロボットプログラミング支援システムの再構築を取り上げ、IBM-PC 版と Macintosh 版のユーザーインターフェースの比較、およびこのシステムを構築する上でユーザーインターフェースに関して色々と思い描いたことを述べる。

2. ユーザーインターフェースの比較

2.1. IBM-PC 版ユーザーインターフェース

すべてのメニューが階層メニュー構造に基づいている。操作方法を述べると、

(1) システムを立ち上げるとメインメニュー画面が現われる。

(2) ソースを編集したければまずエディトメニューを選択する。エディトメニュー画面が現われる。

(3) 編集が終了したら、エディトメニューを抜けメインメニュー画面に戻る。

(4) 編集がすんたのだから、当然次はコンパイルであろう。コンパイルメニューを選択する。コンパイルメニュー画面が現われる。

(5) コンパイルが終了したらコンパイルメニューを抜けメインメニュー画面に戻る。

(6) 次はリンクだ。リンクメニューを選択する。

(7) あっ、もうひとつ編集しなくちゃいけなかった。また、リンクメニューを抜けメインメニュー画面に戻る. . .

```
+-------------------+-----------------------------------------------+
| Program Name      | H-3525                                        |
+-------------------+-----------------------------------------------+
| Function          | HARL-II Compiler for IBM-PC                   |
+-------------------+-----------------------------------------------+
| Edited on         | 1987 - 1 - 28     Ver. 3.01                   |
+-------------------+-----------------------------------------------+
|          (R) (C) All rights reserved by                           |
|      Hirata Industrial Machineries Co., Ltd.                      |
+-------------------------------------------------------------------+

F1  : EDIT mode ----- Create HARL-II program using IBM-PC BASIC
F2  : COMPILE mode -- Compile HARL-II program
F3  : LINK mode ----- Link HARL-II program
F4  : DEBUG mode ---- Debugging, patching, and verifying HARL-II program
F5  : KEY-IN mode --- Edit position data, SG data or Definition Table
F6  : SET-UP mode --- Tune your IBM PC by editing set-up file
F7  : FILES  -------- Show HARL-II program file names in disk B:
F8  : LLIST  -------- Print out HARL-II program in disk B:
F9  : HELP   -------- Explanations on HARL-II compiler
F10 : EXIT   -------- Return to IBM PC DOS

Select function key < F1 - F10 > ?

1EDIT   2COMPIL 3LINK   4DEBUG  5KEY-IN 6SET-UP 7FILES  8LLIST  9HELP   10EXIT
```

Fig.1 Main Menu

```
+-------------------+-----------------------------------------------+
| Program Name      | H-3525-1          COMPILE MODE                |
+-------------------+-----------------------------------------------+
| Function          | Compile HARL-II program                       |
+-------------------+-----------------------------------------------+
| Edited on         | 1987 - 1 - 28     Ver. 3.01                   |
+-------------------+-----------------------------------------------+
|          (R) (C) All rights reserved by                           |
|      Hirata Industrial Machineries Co., Ltd.                      |
+-------------------------------------------------------------------+

F1  : EXECUTE ----- Execute compilation of HARL-II source files
F2  :
F3  :
F4  :
F5  :
F6  :
F7  : FILES   ----- Show file names in disk B:
F8  : LLIST   ----- Print out object file or label.list file in disk B:
F9  : HELP    ----- Explanations on COMPILE MODE
F10 : EXIT    ----- Return to initial menu

Select function key < F1 - F10 > ?

1EXECUT 2       3       4       5       6       7FILES  8LLIST  9HELP   10EXIT
```

Fig.2 Compile Mode Menu

何かをしたければ、その何かに到達するまでに同じ操作を何度も繰り返ささえなければならない。面倒である。

2.2. Macintosh 版ユーザーインターフェース

ビットマップ・ディスプレイとマウスのようなユーザーフレンドリなデバイスを駆使している。スクリーンにはいくつかのウィンドウを積み重ねることができ、異なるメニューをそれぞれのウィンドウを通して同時に見ることで、現在のプログラム環境をいつでも把握できる。例えば、あるウィンドウではプログラムを作成し、別のウィンドウではそのプログラムを走らせ（ダウンロードメニューを選択するとロボットが動く）、さらに別のウィンドウで別のプログラムを編集できる。

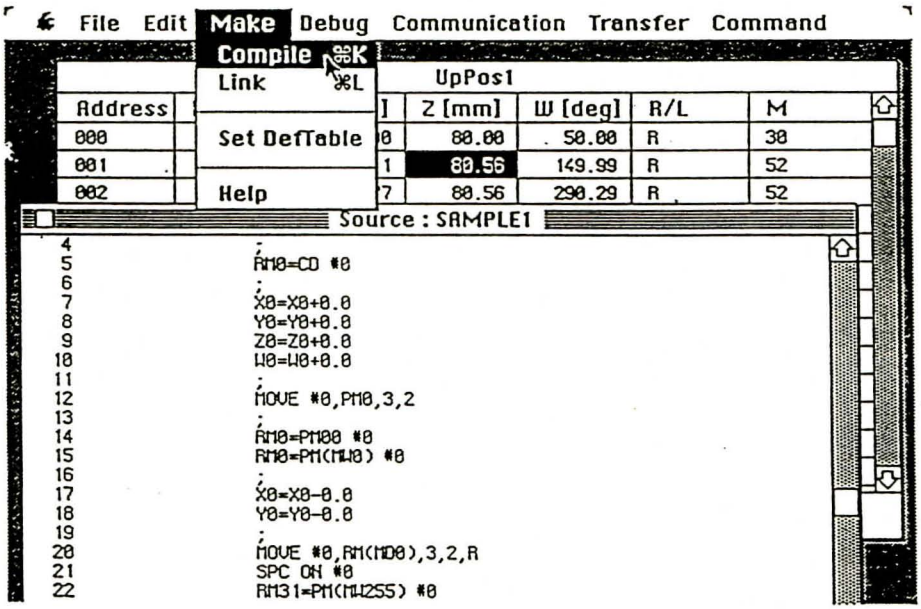


Fig.3 Menu

ある作業を行ないながら、それに関連する別作業を、あるいは気分転換のために別作業を同時に行ないたいというのは人の心理であろう。マルチウィンドウはこの人の心理を満足させるものである。

3. ユーザーインターフェース雑感

今回は以上のようにロボットプログラミング支援システムのユーザーインターフェースをまったく別のものに変えた。このシステムを構築していく過程で、ユーザーインターフェースについて考えたことをまとめて述べたいと思う。

（雑感１） ユーザーインターフェースではユーザーを忘れてはいけない。

IBM-PC 版のユーザーインターフェースより Macintosh 版の方が、おそらくユーザーにとっては馴染みやすいであろう。しかし、マルチウィンドウとマウスが付いたからといって、それで本当に優れたユーザーインターフェースが提供されたとはいえない。マルチウィンド

ウであるから確かに従来の煩わしさは改善された。

しかし、マルチウィンドウには欠点がある。あっという間に画面がウィンドウで埋め尽くされるのである。「積み重ね」は、私たちが限られた大きさの机の上に数多くの本やドキュメントを載せていることと同じである。自分の机の上にただ書類が積み重ねられているのであれば、私たちはある程度の期間はどこに何があるかをだいたい覚えているものである。しかし、私たちが望んでいるのは書類の山ではなくて、それらがわかりやすく整理されたものである。

また、必要な書類を探すのに書類の山の中にあちこち手を突っ込んでみるのも面倒である。同様に、積み重ねられたウィンドウからいま必要とする1枚のウィンドウを探すときに、全部のウィンドウを目で追って必要なウィンドウを見つけてマウスをそこまで持っていくのも面倒である。

私たちは労せずして作業をしたいのである。ユーザーインターフェースではユーザーのつまり私たちのそういう心理を忘れてはならない。

（雑感2）　ユーザーインターフェースは手順・状況そのものである。

人は内容は忘れていても手順や状況は覚えている場合が多い。例えば、書類の山の中のどこに書類をおいたかという状況である。ユーザーインターフェースは手順、状況そのものと言える。つまり、ユーザーインターフェースは覚えやすいように考慮される必要がある。そのためには、

・操作は首尾一貫したものでなければならない
・視覚的な連続性を保たねばならない

であろう。

また、状況によって検索できるようなユーザーインターフェースも望まれる。このためには、ユーザーとの会話（手順、状況）の履歴が保存される必要がある。

（雑感3）　ユーザーは様々である。

実際に利用する人が初心者なのか、たまにしか使わない人なのか、それとも経験豊かな人なのかによって相異なるユーザーインターフェースを設定する必要も生じる（マルチモード入力）。例えば初心者ならば、メニューガイド方式で、操作はマウスのクリックだけですむというのが馴染みやすい。しかし、経験者であればコマンド方式でキーボードの入力ですむほうが馴染みやすい。ユーザーが自然と成長していけるようなユーザーインターフェースが提供されていればなおよいであろう。ユーザーにとって「修得しやすい」ものであって、「信頼性」、「発展性」のあるユーザーインターフェースである。

マシンと人とのインターフェースは、私たちの周りにあるのと同じ物質的、感覚的、知的な空間である。この空間をマシンと人との自然な対話ができるところとにするために、今後も努力していかねばならない。ユーザーインターフェースを設計するうえでの基本姿勢は、人の心理を読み取ること、つまり

”端末の先には人がいる”

ということでである。

4. 最後に—ワークショップへの期待—

ひじょうに当たり前のことを述べてきたが、その当たり前のこと—よりよいユーザーインターフェースの実現—がいまだに確立されていないのである。ユーザーインターフェースはソフトウェアを評価するうえでの重要ポイントである。機能がどんなに充実していても、使い勝手が悪ければその機能は充分に活かされないのである。いま一度、—よりよいユーザーインターフェースの実現—へ向けてどうしたらよいのかを考えてみよう。

あるシステムを開発する際には、まずユーザーへのインタビューが必要であろう。設計は、当然それに基づいてなされなければならない。これらを実際に行なう”ユーザーインターフェース・エキスパート”が、今後必要ではないか？

また、ユーザーインターフェースがひじょうに重要であることを考えれば、社内に”ユーザーインターフェース・グループ”が在ってもよいのではないか？
そして、このグループのメンバーには”女性的感覚”と"男性的気配り”が必要であろう。

また、現状ではユーザーインターフェースの設計からテストまでに要する時間がかなりのものとなっている。この解決として、ユーザーインターフェース部分のライブラリ化を考えたい。そうすることでユーザーインターフェースの設計からテストまでに要する時間は大幅に短縮され、さらに良い品質のユーザーインターフェースを考える時間が得られる。また、この場合いつもばらつきのないユーザーインターフェースを提供できる。

—よりよいユーザーインターフェースの実現—へ向けての道は、まだまだ考えられるであろう。ワークショップに参加し、その道を議論していきたい。

3rd SEA Environment Workshop Position Paper

氏名	新田 稔 種別 ☑会員 no ☐一般
所属	(株)ソフトウェア・リサーチ・アソシエイツ 企画本部 最適開発部
住所	〒102 千代田区平河町1-1-1
TEL	(03)234-2692 FAX ( )

仕 事

ソフトウェア開発支援のツール作成.

最近は. プロトタイパースワークベンチ(PWB) および モデリングエディタ(ERAS) を開発している。

作業環境

Macintosh および SONY/NEWS, VAX11 (UNIX BSD 4.2)

現在の問題

モデリングエディタ ERAS のメタルール作成たなやんでいる。

その他の問題はあまりない。(感じていない?)

次ページの討論テーマ票にも，必要事項をご記入の上本紙とあわせて 12 月 15 日まで(締め切り厳守！)に下記の SEA 事務局までお送りください．もちろん，本式の Position Paper も同封していただくほうがベターです(既発表の論文を添付していただいても構いません)．

ソフトウェア技術者協会 (SEA) 〒102 千代田区隼町 2-12 藤和半蔵門コーブビル 505
TEL: 03(234)9455 FAX: 03(234)9454

ＥＲＡＳのユーザ・インターフェイスの紹介

（株）ソフトウェア・リサーチ・アソシエイツ
企画本部　環境開発部
新田稔

〒102 東京都千代田区平河町1-1-1
電話 03(234)2611
E-Mail nitta@srava.sra.junet

０．はじめに

ユーザ・インターフェイスは、ソフトウェア・ツールの最も重要な部分である。生産効率をどのように高く上げる可能性を持つツールであろうとも、操作方法の習得や使用に必要な背景の学習が難しくて実際に使えるユーザが制限されるなら、全体としてソフトウェア開発作業への貢献度は低くなってしまうだろう。また、最近のツールには特定の方法論を支援するものが増えてきているが、方法論を堅く固定してしまうことは、半面、それが利用できる場面を少なくしたり、ユーザが自分の環境や作業内容に合わせてチューニングする余地をなくしてしまうことにもなり、広くは使われないツールとする恐れもある。

このようなことを避け、使い良いソフトウェア・ツールを提供するため、次の三つのユーザ・インターフェイス目標を提案する

(1)操作を試行すること、および操作方法を他のツールの使用経験やそのツールの他の操作の経験から類推することができ、もし操作を間違えてもすぐにリカバーできること。

(2)背景に高度な理論を持っていても、それを感じさせないこと

(3)ユーザが自分の環境や作業内容、作業の進捗に合わせて自由に応用することができる「柔らかな」ツールとなること

またこの他にも、単に操作上のインターフェイスだけでなく、広くユーザとツールの存在関係についても研究すべきではないだろうか。

ここでは、ＥＲＡＳのユーザ・インターフェイスについて、その工夫の一部を紹介する。ＥＲＡＳは、エンティティ・リレーション・アトリビュート・モデリング（ＥＲＡモデリング）をもとにした図式エディタでアップル社マッキントッシュで動作する（ＵＮＩＸのＸ－Ｗｉｎｄｏｗ版も開発中）。ユーザは、ＥＲＡモデリングに基づいた図式のルール（例えば「ペトリネットは、プレース、トランジション、アーク、トークンから成る」というような）を自ら定義してそのルールに従った図式を作成することができる。ＥＲＡＳは、ルール違反が起きないように、(1)ルール違反を作る操作を行なわせない、(2)ルール違反となった部分を「灰色」表示で知らせる、(3)ユーザの指示で図式全体のチェックを行なう、などの機能を持っている。

１．多くを語るメニュー

ＥＲＡＳのメニュー・システムは、マッキントッシュのメニュー・マネージャを利用している。マッキントッシュのメニュー・マネージャは、スクリーン上部のメニュー・バーに複数のメニュー・タイトルがあり、その上にポインタを置いてマウス・ボタンを押すとさらに複数のメニュー項目が現われるという、プル・ダウン式のメニューを採用している。

このプル・ダウン式のメニューのひとつの長所は、簡単な操作ですべての（その時点で選択不可能なものも含めて）メニュー項目が一覧できることである。

選択不可能なメニュー項目は薄い字（dimmed）で表示されるほか、メニュー項目には、アイコン、チェックマーク、字体の変化（下線や斜体など）を付けることができる。ＥＲＡＳでは、これらを次のように利用している。

(1)ルール違反を作るメニュー項目は選択不可能にする

(2)他の部分と競合する要素を作るメニュー項目は斜体で示す
(3)図式に必須な要素を作るメニュー項目は太字で示す
(4)既に存在する要素を作るメニュー項目にはチェックマークを付ける
これらにより、ユーザは、ルール違反を作る操作を避け、また操作の作用を詳しく予測することができる。

2．マウスの時間差攻撃

マッキントッシュのマウスは、１ボタンである。１ボタン・マウスはボタンの使い分けを覚える必要がなく、また、何の上にポインタを置いてボタン操作するかによってその作用が変わるという方法を採れば、１ボタンでもユーザに分かりやすい形で複数の役割を区別することもできる。

しかし、同一の目的物に対して異なった作用を与える必要があるとき、それらを区別するためにキーボードとの組み合わせを使うアプリケーションが多いが、これでは多ボタン・マウスに優るところはなく、かえって面倒でもある。

ＥＲＡＳでは、１ボタン・マウスの長所を活かすため、ボタンを押している時間によって役割を区別する方法を採っており、どの役割に解釈されたかは、ポインタの形を変えてユーザにフィード・バックされる。例えば、空白位置にポインタを置いてドラッグ動作（ボタンを押したままでマウスを移動すること）を行なうとき、ボタンを押してすぐにドラッグすると矩形範囲の選択になり、ボタンを押してポインタの形が「手」に変わるまで待ってからドラッグするとスクロール操作になるのである。

この時間差は、ユーザが、後の役割になるまでの「待ち」を感じない程度には短いが、少しの慣れで自由に使い分けできる程度には長い、という微妙な長さで、また、ユーザが微調節することもできるようになっている。

3．ルール違反の指摘と許容

ＥＲＡＳは、ルール違反のない図式を作成するためのエディタであるが、なにがなんでもルール違反は許さないという堅いシステムではなく、積極的にルール違反を作る操作は許さないが、作成途中で自然に出てくる違反は許容し、機会ある毎にそれを指摘するという柔らかなシステムである。

例えば、エンティティ間に成立しないリレーションを作ることはできない（メニュー項目が選べない）が、リレーションの付け換え時などに生じるルール違反のエンティティとリレーションの組み合わせは許す、という具合にである。

ルール違反の指摘は、(1)ユーザが違反に関係する要素を選んだとき、違反部分を灰色に表示する、(2)ユーザの指示によって図式全体を調査し、違反をみつけてその詳しい内容をメッセージで表示する、という２通りの方法で行なう。

これらにより、構造エディタや特定文法向きエディタにありがちなユーザの束縛感を少なくし、また、ルールを途中変更するときのフレキシビリティも増している。

4．最後に

ＥＲＡＳは、ツールと方法論（図式のルール）を切り離しを強く意識したツールである。本来、どの方法を用いるのが良いかは、ツールの作成者よりも、問題に直面しているユーザの方がよく知っていることであり、ユーザに選ぶ権利があるはずである。ところが、方法論がツールに固定されてしまうと、方法論が違うからといってツールを換えたり、新しい方法論が採用される毎にツールを作らなければならない、ということなり、これはツールの作成者とユーザ双方の不利益になると考えたからである。

ソフトウェア開発過程は、進捗によって手順や方法そのものが変化したり、あるいは、あるところまで行ってみなければその先の方針が立てられないという性格を持っている。このグニャグニャとしてつかみどころのないソフトウェア開発作業には、やはり柔らかくて臨機応変に振舞えるツールが必要だと思われる。

追記

ポジション・ペーパにはＥＲＡＳそのものに関する説明が不足していましたので、それを補うために、その前身となっているＥＲＡとＭＯＳＳの解説書を付加します。ただしこれらは以前他の目的に書いたものですので（特にＭＯＳＳの解説書は下手な英語で書いてあります）、読み難さについてはあらかじめお詫び申し上げます。

ＥＲＡ　－　ルール指向型図式エディタ

（株）ソフトウェア・リサーチ・アソシェイツ
環境開発本部
新田稔

0 はじめに

ＥＲＡは、エンティティー(Entity)・リレーション(Relation)・アトリビュート(Attribute)方式を応用した「規則指向型図式エディタ」である。現在ＥＲＡは、Smalltalk-80、およびアップル社のMacintoshで動作しており、UNIX搭載のワークステーションのＸ－ウィンド環境にも移植中であるが、ここでは、Macintosh用のバージョン2.2の機能をもとに解説する。

1 ＥＲＡの概要

ＥＲＡの特徴は、(1)エンティティー・リレーション・アトリビュート方式を基本とする規則にしたがった図式を描くことができる、(2)ユーザが規則を定義することができる、(3)オブジェクト指向的な操作により規則違反をおかさない、ことである。

1.1 エンティティー・リレーション・アトリビュート方式

エンティティー・リレーション・アトリビュート方式とは、物事を、もの（エンティティー）、ものとものとの関係（リレーション）、およびものや関係の特性（アトリビュート）の三つの要素を用いて表す方法で、この三要素を適当に設定することにより、いろいろな事柄を表現することができる。

たとえば、「私」が「ホームズの冒険」を「読む」、という事柄を表すには、「私」という「人間」型のエンティティーと「ホームズの冒険」という「本」型のエンティティーとを、「読む」というリレーションで結べばよい。「人間」型のエンティティーは、「性別」、「年齢」などというアトリビュートを持つであろう。これらのアトリビュートは「男」、「２０」などという値を持つことになる。

さて、次の図１と図２は、状態遷移図とペトリネットで、簡単なエンティティー・リレーション・アトリビュート図の例である。

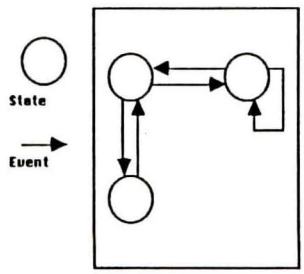


図１　状態遷移図

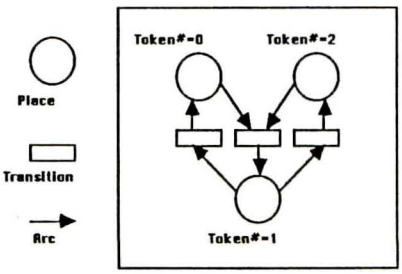


図２　ペトリネット

図１の状態遷移図には、エンティティーの型として「状態(State)」が、リレーションの型として「出来事(Event)」がある。「出来事」は、単に二つの「状態」を結ぶだけでなく、ある「状態」から次の「状態」へと方向を持っている。一般に、リレーションは、ひとつのエンティティーから他のエンティティーへと方向を持つ。これは言い換えると、リレーションで結ばれたエンティティーは、それぞれ、そのリレーションのもと、リレーションの先、という役割を与えられることになる。このうち、リレーションのもとになるエンティティーをサブジェクト・エンティティー、リレーションの先になるエンティティーをオブジェクト・エンティティー、と呼ぶ。

この状態遷移図の場合、「状態」は、「出来事」のサブジェクトにもオブジェクトにもなることができ、さらに同一の「状態」が同時に「出来事」のサブジェクトとオブジェクトにもなることができる。

図２のペトリネット図には、エンティティーの型として「場所(Place)」と「遷移(Transition)」があり、リレーションの型として「アーク(Arc)」がある。「場所」、「出来事」ともに「アーク」のサブジェクトにもオブジェクトにもなれるが、「場所」は、「場所」をサブジェクトとする「アーク」のオブジェクトにはなれず、また「出来事」は、「出来事」をサブジェクトとする「アーク」のオブジェクトにはなれない。

「場所」は、「トークン数(Token#)」というアトリビュートを持っているが、「出来事」は、このアトリビュートを持つことができない。一般に、アトリビュートは、それを持つことができるエンティティーが限定される。

1.2 ＥＲＡで扱う規則

ＥＲＡで扱う規則は、次のとおり。

(a)エンティティーに関して

(a-1)エンティティー型の種類

(b)リレーションに関して

(b-1)リレーション型の種類

(b-2)サブジェクトとなるエンティティー型とオブジェクトとなるエンティティー型の組み合わせ

(b-3)同一のエンティティーを同時にサブジェクトにできるかどうか

(b-4)同一のエンティティーを同時にオブジェクトにできるかどうか

(b-5)サブジェクトとオブジェクトのエンティティー型が同じとき、同一のエンティティーを同時にサブジェクトとオブジェクトにできるかどうか

(b-6)同じサブジェクトとオブジェクトの間に複数のリレーションができるかどうか

(c)アトリビュートに関して

(c-1)アトリビュート型の種類

(c-2)アトリビュートが付属するエンティティー、およびリレーション型の種類

(c-3)必須アトリビュートか、オプショナル・アトリビュートか

これらの規則は、ＥＲＡのユーティリティ「ERA Rule Maker」を用いて、ユーザが自由に設定することができる。例えば、前述のペトリネットでは、次のような規則になる。

(a-1)Place、Transition

(b-1)Arc

(b-2)組み合わせ－１：Place（サブジェクト）・Transition（オブジェクト）、および組み合わせ－２：Place（サブジェクト）・Transition（オブジェクト）

(b-3)組み合わせ－１、２に関して：できる

(b-4)組み合わせ－１、２に関して：できる

(b-5)サブジェクトとオブジェクトのエンティティー型が同じ組み合わせはない

(b-6)組み合わせ－１、２に関して：できる

(c-1)Token#

(c-2)Token#が付属するのはPlace

(c-3)PlaceにToken#は必須

1.3 ＥＲＡの規則指向

ＥＲＡは、操作の対象物が指定された後、その対象物に対して有効な処理項目だけがメニューから選べるような、オブジェクト指向的な操作手順を取り入れることによって、ユーザにルール違反を起こさせないようになっている。上にあげたそれぞれのルールに対して、具体的な操作は、次のようになる。

(a-1)に対して：規定されたエンティティー型しかメニューになく、またこのメニューは、エンティティーを作る場所を指定した後にだけ選べる。
(b-1)に対して：規定されたリレーション型しかメニューになく、またこのメニューは、リレーションで結ぶエンティティーを指定した後にだけ選べる。
(b-2)に対して：選ばれたエンティティーの組み合わせに対して可能なリレーション型だけがメニューから選べる。
(b-3)、(b-4)、(b-5)、(b-6)に対して：違反するリレーションを新しく作ると、既存のリレーションが削除される。
(c-1)に対して：規定されたアトリビュート型しかメニューになく、またこのメニューは、アトリビュートを付けるエンティティーを指定した後にだけ選べる。
(c-2)に対して：選ばれたエンティティーまたはリレーションに対して可能なアトリビュート型のみメニューから選べる。
(c-3)に対して：必須アトリビュートは、エンティティーまたはリレーションが作られたときに付属する。

またCutやPasteなどは、同じメニュー項目でも対象物によってその処理内容が変わるようになっている。

2 ＥＲＡの応用例

ＥＲＡは、ソフトウェア開発での、要求定義や設計のグラフィカルな記述に応用できる。グラフィカルな表現は、ソフトウェアの非専門家にも理解しやすく、またユーザが自由にルールを設定・変更できるので、従来の文法を固定した記述言語にはなかった問題に合わせたフレキシブルな記述や、最適の記述方法をさがしながら作業を進めることが可能である。次にＥＲＡで描いた図式の例をいくつか紹介する。

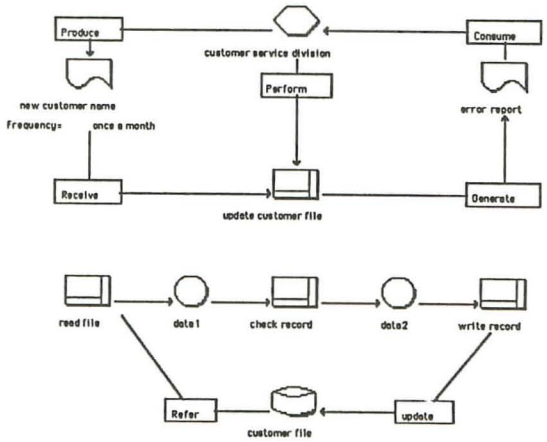


図３　コンセプチュアル・プロセス・モデル

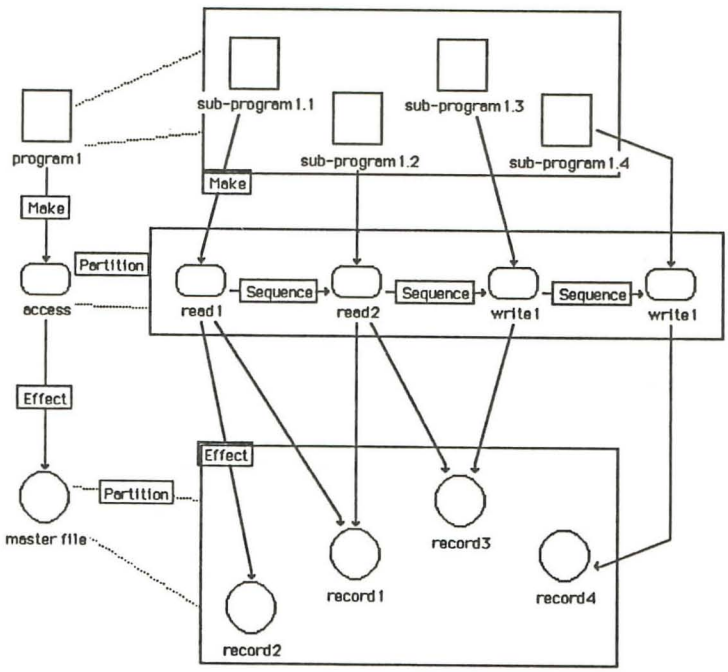


図４　モジュール構造図

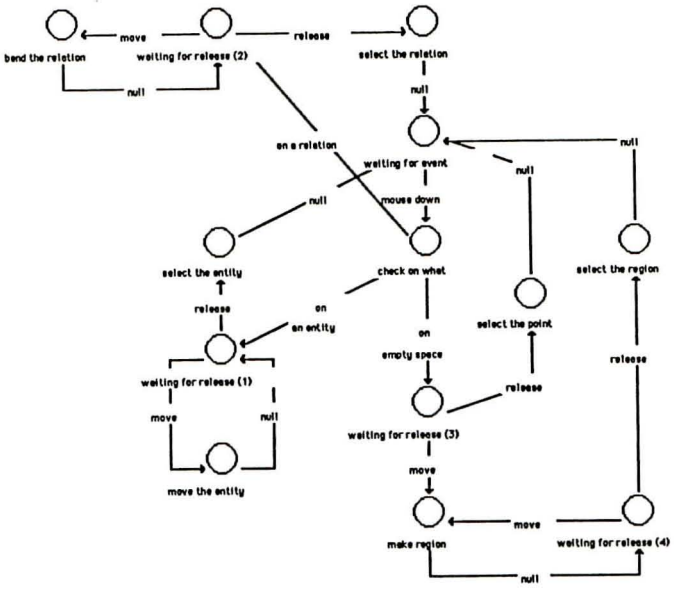


図５　状態遷移図

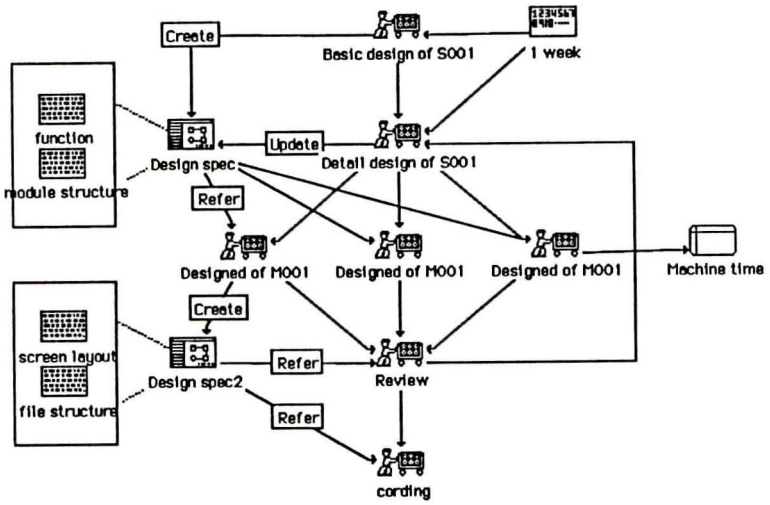


図６　作業手順図

Modeling Notation and Executing Procedure of MOSS

(Multi- Object System Modeling Editor & Simulator)

Minoru Nitta
Software Research Associates, Inc.
Tokyo, JAPAN

0. introduction

MOSS (Multi- Object System Modeling Editor & Simulator) is a experimental software tool. It has an ability to represent the dynamic behavior of a software/hardware system which contains multiple objects which perform a sequence of actions individually. It can be applied for visual programing, prototyping and executable specifications. Three types of graphical modeling notation; 1)action flow modeling, 2)state transition modeling and 3)petri net modeling are used in MOSS. We will discuss the modeling notation and the executing procedure of the models. MOSS is now working on Macintosh™ and the discussion below is based on the version 0.2.

Keywords - visual programing, prototyping, executable specification, graphic editor, state transition diagram, petri net, modeling method.

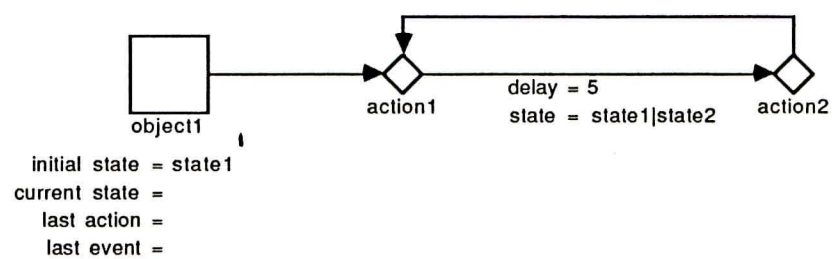
1. Modeling Notation

Three types of element; 1)node, 2)arc and 3)attribute are used to describe models. Nodes are expressed by icons and have names. Arcs are expressed by directive lines from node to node. Attributes are associated with nodes and arcs and are expressed in text form.

Action Flow Modeling

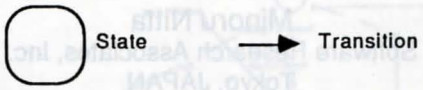
The action flow model represents the objects in the target system and the action sequences taken by the objects. There are two types of node; 1)*object* and 2)*action* and one type of arc; *sequence*.

Objects have four attributes; 1)*initial state*, 2)*current state*, 3)*last action* and 4)*last event*. The value of *initial state* is specified by the user and the values of the other three attributes are changed automatically during execution. The name of an *actions* can contain the object-name-substitution which is represented by the character "*". For example, when an *object* named "obj1" takes an *action* named "*-act", the actual action name becomes "obj1-act". *Sequences* link objects and their first *actions*, and *actions* and their successors. *Sequences* have two attributes; 1)*delay* which indicates the time lag between an *action* and its successor, and 2)*state* which indicates the conditional state to take the next *action*. The value of *delay* is set to zero or positive integer (default 1) by the user. The value zero has a special meaning - the random time lag. The value of *state* is set to one of the state names in the state transition model or an or-list of them, or can be omitted. The or-list is represented by the character "|", like "state1|state2".

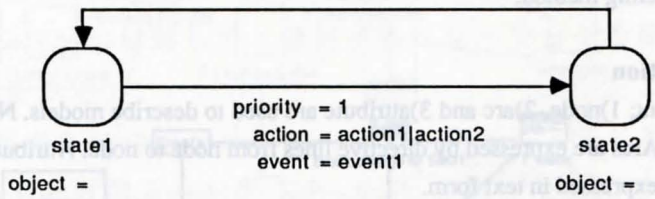


State Transition Modeling

The state transition model represents the relationship among actions, transitions and events. There are one type of node; *state* and one type of arc; *transition*.

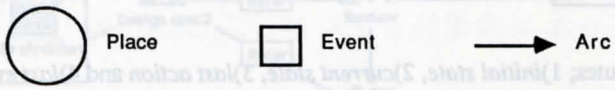


States have an attribute; *object* which indicates the object names which stay on the *states*. The value of *object* are changed automatically during execution. *Transitions* have three attributes; 1)*action* which indicates the action names which invoke the *transitions*, 2)*priority* which indicates the priority when there are two or more *transitions* which originates from the same *state* and can be invoked by the same actions, and 3)*event* which indicates the event name which will happen in the whole system when the *transitions* are invoked. The value of *action* is set to one of the action names in the action flow model or an or-list of them, or it can be omitted. The value of *priority* is set to zero or positive integer (default 1) by the user. Smaller values mean higher priorities. The value of *event* is set to one of the event names in the petri net model or can be omitted. It can also contain the object-name-substitution.

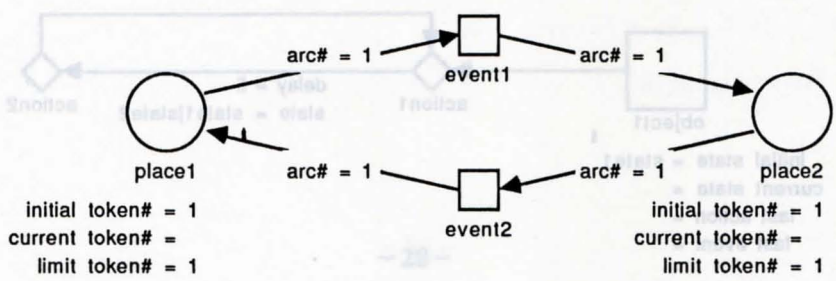


Petri Net Modeling

The petri net model represents the state of the whole system and how the events change it. This model will make the relationships (and the constraints) among the individual objects. There are two types of node; 1)*place* which contains tokens and 2)*event* which moves tokens from *place* to *place*. There is one type of arc; *arc* which is the path of token transfer.



Places have three attributes; 1)*initial token#*, 2)*current token#* and 3)*limit token#*. The value of *initial token#* is set to zero or positive integer (default 1) by the user. The value of *limit token#* is set to zero or positive integer, or can be omitted. The value of *current token#* is changed automatically during execution. *Arcs* link *places* and *events*. *Arcs* have an attribute; *arc#* which indicates the number of token transferred when an event happens. The value of *arc#* is set to zero or positive integer (default 1) by user. The value zero has a special meaning - inhibitor arc. *Events* are fireble when the value of *arc#* of each *arc* which gets toward the *events* is not zero and less than or equal to *current token#* of the *place* from which the *arc* originates, or both *arc#* and *current token#* are zero.



2. User Interface

We touch MOSS's user interface very briefly here. For detail, please refer to the "MOSS user's manual".

Multi- Window

MOSS uses three windows. The action flow model, the state transition model and the petri net model are described in each window respectively. Because each model can be saved/loaded separately and the relations among models are taken place symbolically, it is easy to replace models by alternatives.

Mouse and Menu

The user generally takes two steps to do something with MOSS; 1)selecting a target of the operation by the pointer (mouse) and 2)choosing an item from the menu. Because only items which have some effects on the target are enabled in the menu after the selection, user can't chose any meaningless items from the menu and can't make models violating the modeling notation.

3. Executing procedure

The execution of the models are represented by highlighting nodes and arcs, and changing values of attributes.

Initialization

Before the execution, the models are initialized. In the action flow model, the value of *current state* of each *object* is set to its *initial state*. In the state transition model, the value of *object* of each *state* is set to the *object* name which *initial state* is this state. And the value of *current token#* of each *place* are set to its *initial token#* in the petri net model. The total number of tokens is also calculated for the conservation checking.

Action Flow Model

The execution procedure of the action flow model is:

(1) at first, each *object* takes an *action* which is connected to the *object* by *sequence*

(2) after an *object* took an *action*, the value of its *last action* is set to the *action* name

(3) the *action* name is also transferred to the state transition model and it will invoke the state transition of the *object*

(4) after an *action* was taken, one of *sequences* which originates from the *action* is selected. The *sequence* must be; i) the value of *delay* is greater than or equal to the the time passed after the *action* was taken, and ii) the value of *state* (if specified) is equal to the state name on which the *object* stays. If there are two or more candidates, one of them is selected randomly. If there are no candidates, no *actions* are taken

(5) if no sequences originates from the *action*, the execution of the *object* is halted.

State Transition Model

The execution procedure of the state transition model is:

(1) receiving the action name, one of *transitions* which originate from the *state* on which the *object* stays is selected. The *transition* must be: i) the value of *action* is equal to the action name or is omitted, and ii) the value of *event* is fireble in the petri net model. If there are two or more candidates, one which has highest priority is selected. If there are candidates which have same priority, one of them is selected randomly

(2) the value of the *event* is transferred to the petri net model

(3) after a *transition* was invoked, the state of the *object* transits to the destination *state*. The value of *object* of the *states* (old and new one) and the value of *current state* of the *object* are changed.

Petri Net Model

The execution procedure in the petri net model is:

(1) receiving the *event* name, for each *arc* which gets toward the *event*, the value of *current token#* of the *place* from which the *arc* originates is decreased by the value of *arc#* of the *arc*

(2) for each *arc* which originates from the *event*, the value of *current token#* of the *place* which is the destination of the *arc* is increased by the value of *arc#* of the *arc*

4. Error Detection

Syntax Errors

Because the operations to make models are syntax oriented, the user can't violate the modeling notation.

Errors on Initialization

When the models are initialized, the following errors which are fatal for execution are detected.

(1) Unspecified Initial State - the *initial state* of an *object* is not specified in the action flow model.

(2) Undefined Initial State - the *initial state* of an *object* is not defined in the state transition model.

(3) Multi- defined Initial State - the *initial state* of an *object* is multiply defined in the state transition model.

Errors on Executing

When the following errors are detected on executing, the execution is paused.

(1) Ineffective Action - no *transitions* are invoked by the *action*; there are no *transition* which *action* value is equal to the *action* name taken by the *object*, or the *events* of all possible *transitions* can't happen. This error detection can be turned off and the *action* is simply ignored.

(2) Undefined Event - the *event* of an transition is not defined in the petri net model. This error detection can be turned off and the *event* is assumed to be fireble.

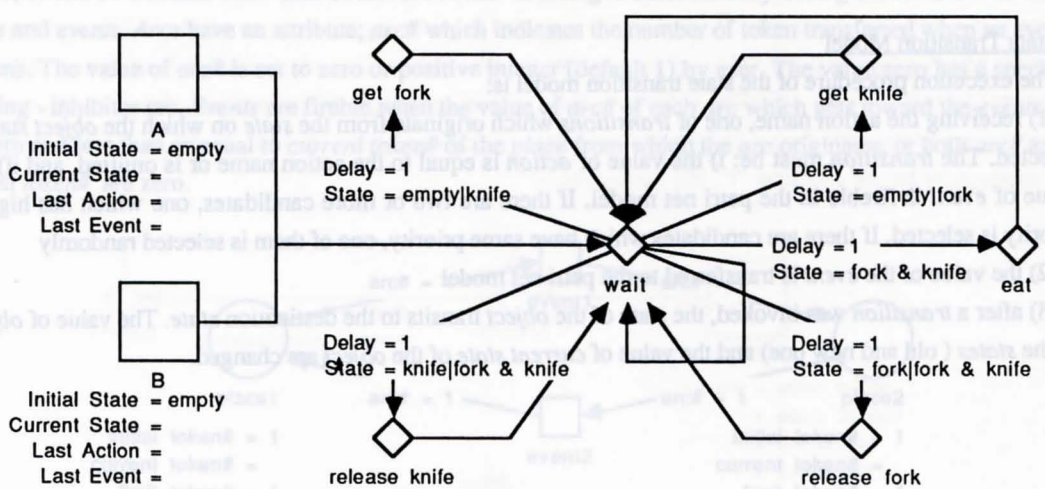
(3) Violation of Bundedness of Petri Net - the value of *current token#* of a *place* exceeded the value of *limit token#* (if specified) in the petri net model. This error detection can be turned off.

(4) Violation of Conservation of Petri Net - the sum of *current token#* isn't conserved. This error detection can be turned off.

5. Examples

5.1 Example1:phylosophers' supper

This example represents a system in which two phylosophers have supper scrambling for a set of fork and knife.



Action flow model

There are two *objects*; "A" and "B" in the model. Their *initial states* are "empty". At first, "A" and "B" take an *action* "wait". The next *action* depends on the *current state* of the *object*.

(1) *object* can take *action* "get fork" when it stays on *state* "empty" or "knife".

(2) *object* can take *action* "get knife" when it stays on *state* "empty" or "fork".

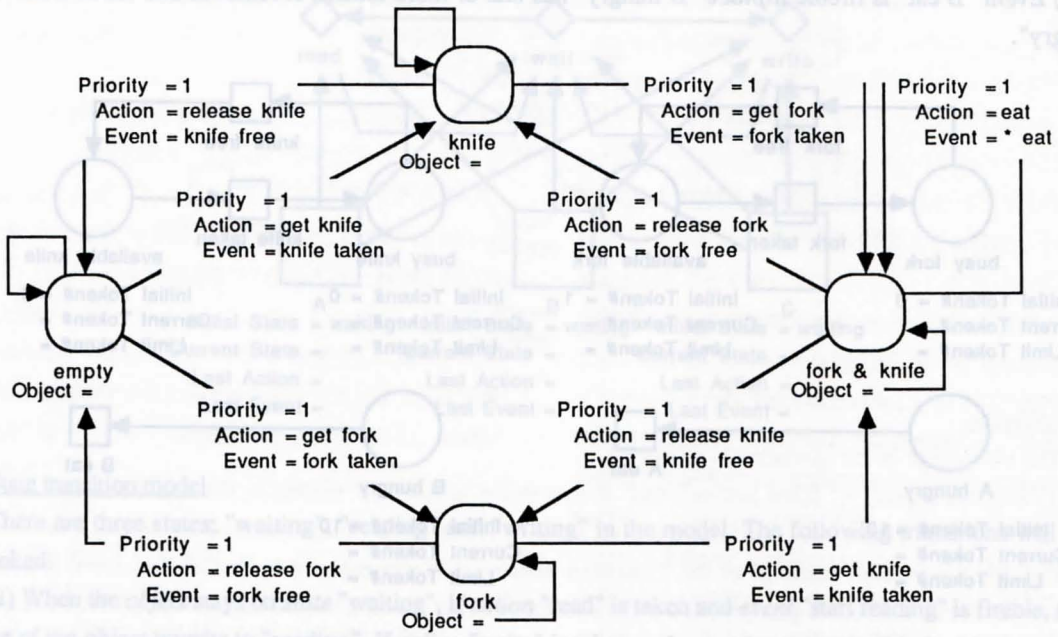
(3) *object* can take *action* "release fork" when it stays on *state* "fork" or "fork & knife".

(4) *object* can take *action* "release knife" when it stays on *state* "knife" of "fork & knife".

(5) *object* can take *action* "eat" when it stays on *state* "fork & knife".

(6) *object* can take *action* "wait" always.

After *actions* other than "wait", "wait" must be taken.



State transition model

There are four *states*; "empty", "knife", "fork" and "fork & knife" in the model.

(1) If the *object* stays on *state* "empty", when *action* "get knife"is taken, the *transition* which *event* is "knife taken" is invoked and the *state* of the *object* transits to *state* "knife". When *action* "get fork" is taken, the *transition* which *event* is "fork taken" is invoked and the *state* of the *object* transits to *state* "fork". When the *events* are not fireble or other *actions* are taken, the *object* remains on *state* "empty".

(2) If the *object* stays on *state* "fork", when *action* "get knife" is taken, the *transition* which *event* is "knife taken" is invoked and the *state* of the *object* transits to *state* "fork & knife". When *action* "release fork" is taken, the *transition* which *event* is "fork free" is invoked and the *state* of the *object* transits to *state* "empty". When the *events* are not fireble or other *actions* are taken, the object remains on *state* "fork".

(3) If the *object* stays on *state* "knife", when *action* "get fork" is taken, the *transition* which *event* is "fork taken" is invoked and the *state* of the *object* transits to *state* "fork & knife". When *action* "release knife" is taken, the *transition* which *event* is "knife free" is invoked and the *state* of the *object* transits to *state* "empty". When the *events* are not fireble or other *actions* are taken, the *object* remains on *state* "knife".

(4) If the *object* stays on *state* "fork & knife", when *action* "release fork" is taken, the *transition* which *event* is "fork free" is invoked and the *state* of the *object* transits to *state* "knife". When *action* "release knife" is taken, the *transition* which *event* is "knife free" is invoked and the *state* of the *object* transits to *state* "fork". When *action* "eat" is taken, the *transition* which *event* is "* eat" is invoked and the *object* remains on *state* "fork & knife". When the *events* are not fireble or other *actions* are taken, the *object* remains on *state* "fork & knife".

Petri net model

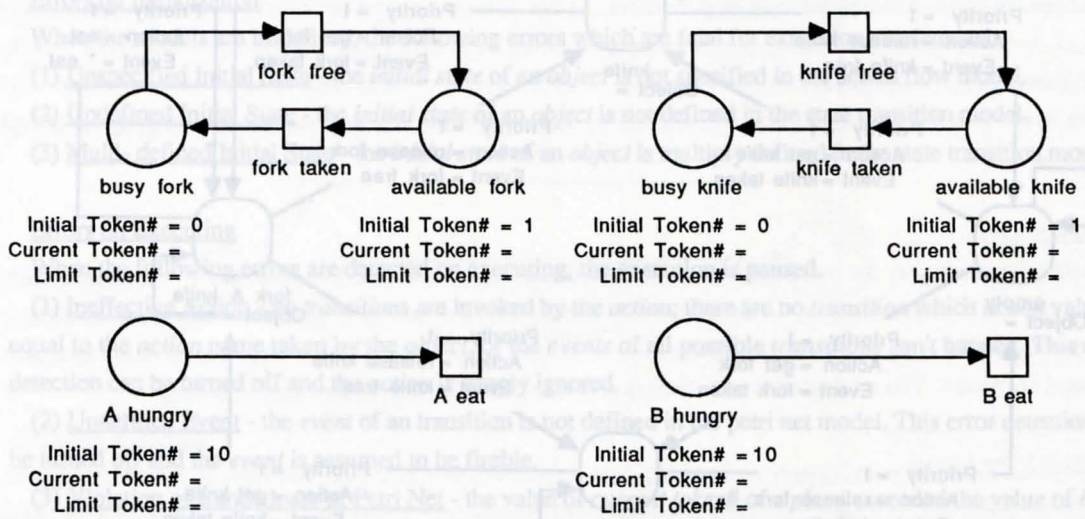
This model includes four sub- sections.

(1) *Event* "fork taken" is fireble if *place* "available fork" has one or more tokens. It transfers one token from "available fork" to "busy fork" when fired. *Event* "fork free" is fireble if *place* "busy fork" has one or more tokens. It transfers one token from "busy fork" to "available fork".

(2) *Event* "knife taken" is fireble if *place* "available knife" has one or more tokens. It transfers one token from "available knife" to "busy knife" when fired. *Event* "knife free" is fireble if *place* "busy knife" has one or more tokens and it transfers one token from "busy knife" to "available knife".

(3) *Event* "A eat" is fireble if *place* "A hungry" has one or more tokens. It removes one token from "A hungry".

(4) *Event* "B eat" is fireble if *place* "B hungry" has one or more tokens. It removes one token from "B hungry".



Execution

The early stage of execution is following.

(1) *Object* "A" takes its first *action* "wait" in the action flow model. Because there are no *transitions* which originate from *state* "empty" and are invoked by *action* "wait" in the state transition model, *action* "wait" is ignored (assume the error detection is turned off) and the *state* of *object* "A" remains on state "empty".

(2) The next *action* of *object* "A" is "get fork" or "get knife". One of them is selected randomly. Assume *action* "get fork" is taken.

(3) There is a *transition* which originates from *state* "empty" and is invoked by "get fork" in the state transition model. The value of *event* is "fork taken".

(4) In the petri net model, there is an *arc* which gets toward *event* "fork taken". Its *arc#* is 1 and it originates from *place* "available fork". If the value of *current token#* of "available fork" is grater than or equal to 1, the *event* "fork taken" is fireble. In this case, one token flows from "available fork" to "busy fork" and the values of *current token#* are modified. Then the *transition* is invoked and the value of *current state* of *object* "A" is set to the destination *state* "fork", *last action* is "get fork" and *last event* is "fork taken".

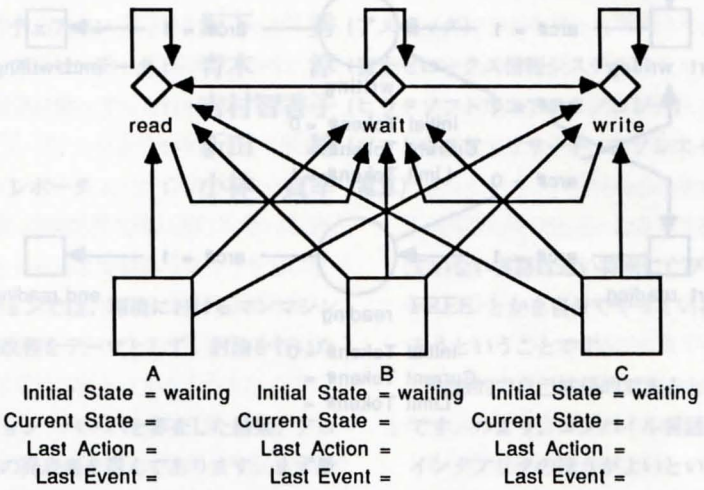
(5) But if *object* "B" took the fork first and the value of *current token#* of "available fork" is zero, the *event* is not fireble and the *transition* isn't invoked. Then *object* "A" remains on *state* "empty" and the value of *current state* is unchanged. The value of *last action* is set to "get fork" but *last event* is set to null.

5.2 Example2:read and write problem

This example represents a system in which three identical tasks; "A", "B" and "C" try to read and write the same file simultaneously.

Action flow model

There are three *objects*, "A", "B" and "C" in the model. Their *initial states* are "waiting". The first *actions* are "read", "write" or "wait". One of them is selected randomly. The next *actions* are 1)"read" or "write" following "wait", 2)"wait" or "read" following "read", or 3)"wait" or "write" following "write".



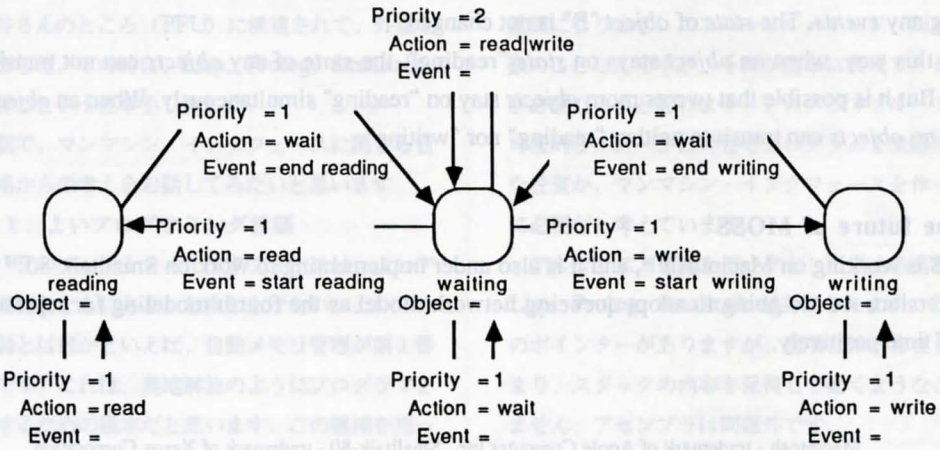
State transition model

There are three states; "waiting", "reading" and "writing" in the model. The following transitions will be invoked.

(1) When the *object* stays on *state* "waiting", if *action* "read" is taken and *event* "start reading" is fireble, the state of the *object* transits to "reading". If *action* "write" is taken and *event* "start writing" is fireble, the state of the *object* transits to "writing". If the *events* are not fireble or *action* "wait" is taken, the *object* remains on "waiting".

(2) When the *object* stays on *state* "reading", if *action* "read" is taken the *object* remains on "reading". If *action* "wait" is taken, *event* "end reading" happens and the *state* of the *object* transits to "waiting".

(3) When the *object* stays on *state* "writing", if *action* "write" is taken the *object* remains on "writing". If *action* "wait" is taken, *event* "end writing" happens and the *state* of the *object* transits to "waiting".

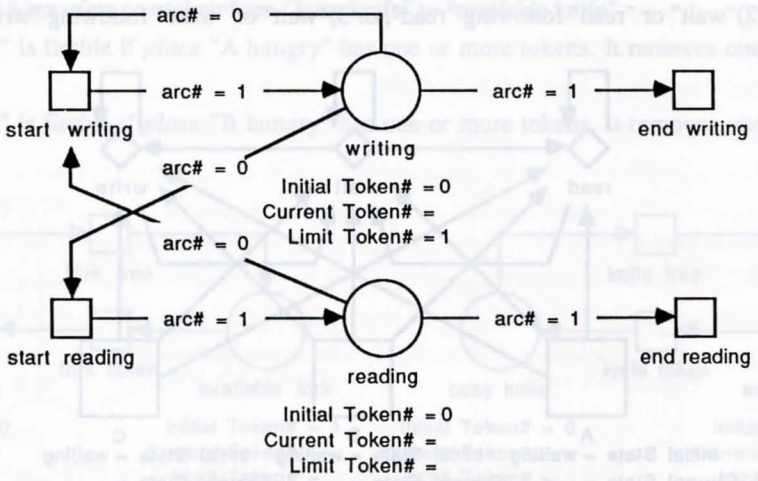


Petri net model

This model includes two sub- sections.

(1) *Event* "start reading" is fireble when there are no tokens in *place* "writing". It appends one token to *place* "reading". *Event* "end reading" is fireble always. It removes one token from "reading".

(2) *Event* "start writing" is fireble when there are no tokens in both "writing" and "reading". It appends one token to place "writing". *Event* "end writing" is fireble always. It removes one token from "writing".



Execution

The early stage of the execution is following.

(1) Assume *object* "A" takes *action* "read" at first in the action flow model. In the state transition model, there are two *transitions* which originates from *state* "waiting" and can be invoked by *action* "read". It is checked that *event* "start reading" of the *transition* which has higher priority is fireble. The *event* is fireble because there is an *arc* which gets toward the *event* in the petri net model and both its *arc#* and *current token#* of *place* "writing" from which the *arc* originates are zero. The *transition* is invoked and the *state* of *object* "A" transits to "reading". The *event* "start reading" happens and it appends one token to *place* "reading".

(2) Assume *object* "B" takes *action* "write". There are two *transitions* which originates from *state* "waiting" and can be invoked by *action* "write". It is checked that *event* "start writing" of the *transition* which has higher priority is fireble. There are two *arcs* which get toward the *event* in the petri net model. Because *arc#* of the *arc* which originates from *place* "reading" is zero but *current token#* of "reading" is not zero (it is 1 now), the *event* is not fireble.

(3) The value of *event* of the lower priority *transition* is not specified. This *transition* is invoked without associating any *events*. The *state* of *object* "B" is not changed.

(4) In this way, when an *object* stays on *state* "reading", the *state* of any *objects* can not transit to *state* "writing". But it is possible that two or more *objects* stay on "reading" simultaneously. When an *object* stays on "writing", no *objects* can transit to neither "reading" nor "writing".

6. The future of MOSS

MOSS is working on Macintosh™, and it is also under implementing to work on Smalltalk-80™ and X. In the next version, we are going to adopt queueing network model as the fourth modeling for representing the concept of time positively.

Macintosh - trademark of Apple Computer Inc., Smalltalk-80 - trademark of Xerox Corporation
X - a network transparent windowing system developed at MIT

第３回 SEA 環境ワークショップ

セッション１

マン・マシン・インタフェースの改善

チェアマン : 坂下 秀（アステック）
プレゼンテータ : 青木 淳（富士ゼロックス情報システム）
: 吉村智香子（ヒラタソフトウエアテクノロジー）
: 新田 稔（ソフトウエア・リサーチ・アソシエイツ）
レポータ : 小林 貞幸（電算）

１．はじめに

坂下： このセッションでは，環境におけるマンマシン・インタフェースの改善をテーマとして，討論を行いたいと思います．

みなさんのポジション・ペーパを審査した結果，プログラム委員会で３人の発表者を選んであります．まず最初に，富士ゼロックス情報システムの青木さん，お願いします．

２．プレゼンテーション

２．１．SUMMIT

青木： 「破壊と建設を繰り返して美が生まれる」というのが，ソフトウェアを作る時の私の美学です．既存の物をできるかぎり壊して，その中からまた新しいもの，美しいものを作り出してゆこうということが根本にあります．

私は言語屋です．今は富士ゼロックス情報システムにいますが，その前は富士通ビーエスシーにおりまして，そこで，アセンブラや C の処理系を作っていました．，その後熊谷さんのところ（PFU）に派遣されて，外注として働きました．その時に，Lisp，Prolog，Smalltalk の処理系を作るという仕事をしました．

そんな訳で，マンマシン・インタフェースに関する言語屋の立場からの考えをお話してみたいと思います．

２．１．１．よいプログラミング言語

まず，「よいマンマシン・インタフェースは，よいプログラム言語とともにある」と申し上げたい．よいプログラム言語とは何かといえば，自動メモリ管理が第１番目の条件です．これは，農地解放のようにプログラマを皆平等にするための根本だと思います．この機構を持っていない言語は近い将来に亡びるでしょう．ALOC とか FREE とかを自分でやっているような言語はもうダメだろうということです．

対話的で自己拡張的であるというのが，２番目の条件です．つまり，コンパイル言語は，やっぱりもうダメで，インタプリタのほうがよいということですが，実行速度が問題になるので，その辺を何とか解決して，コンパイル言語とインタプリタ言語のあいの子みたいなものがよいのではないかと思っています．

ソースを書いて一括コンパイルしてメモリに乗せて走らせるような言語は，もうナンセンスという感じです．少なくとも，そういう言語の場合には，インクルメンタルローダを自分で作っておいて，動的リンクをしながらやっていくようにすべきでしょう．そうすれば，プログラムが間違っていたら，その部分だけ入換えることができます．マンマシン・インタフェースを作っていく時に，これはきわめて重要なことだと思います．

３番目には，アルゴリズムをデータとして扱えるという特性が，特にマンマシン・インタフェースにとっては，重要になります．ユニバーサルメソッド，Lisp の万能関数のことなんですが，それが簡単に実現できる言語構造が必要だろうと思います．プログラマブル・アイコン，可視的なイメージに対してプログラムを束縛できるような性質が，マンマシン・インタフェースを作っていく時に必要だと考えています．

アルゴリズム自体をデータとして扱える言語としては，Lisp，Prolog，Smalltalk があげられます．C にも関数へのポインターがありますが，環境自体が存在しない．つまり，スタックの内容を保持して動くようなことができません．アセンブラは問題外です．

それから４番目に，対象物が自然に表現できること．すんなりモデル化できて扱えることが重要です．これは，Smalltalk が得意だといわれていることですけど，一応オブジェクト指向の考え方です．

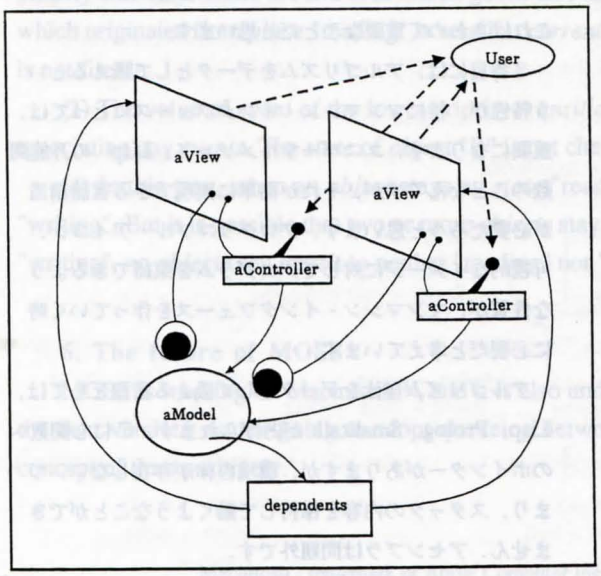
オブジェクトがいっぱいあると，対象物間の関係を表現できることが必要になります．つまり，あるマシンとマシンの間のネットワークとかがスンナリ表現できる言語体系が必要だろうと思っています．Smalltalk で対象物を自然に表現することは，勉強すればそれなりにできるようになるんですが，そうやってできたオブジェクト間の関係を記述するのはむずかしい．

ですから，この２つができるような言語が，これから望まれるんじゃないかと思います．で，そういう言語の上に初めて，よりよいマンマシン・インタフェースが作られるだろうというのが，私の持論です．

２．１．２．SmalltalkのMMI

それで私，このところずっと Smalltalk をやっています．昔は処理系を作っていたんですけど，最近は Smalltalk を使っていろいろなものを作っています．

Smalltalk のマンマシン・インタフェースはユニフォームな構造に特徴があります．NTT の竹内さんは，Smalltalkはインタフェース言語だと言い切っていますが，私もそれに賛成です．物と物とが話しあうようなことを，根底からきれいに記述するのに向いていると思います．そのユニフォームな構造がどんなものかということを，少しお話します．



Smalltalk の中には，すべてのオブジェクトを見るためのツールが揃っています．たとえば，１とか２とか整数を見るためのウィンドすらあります．実際には，インスペクターと呼ばれるものです．それを専用化していくと，システム・ブラウザーとかファイル・リスターとかになっていきます．よくいわれる MVC つまりモデル・ビュー・コントローラとは，ある物をあるビューから見ている，そのビューにはコントローラが付いていて，ユーザは，レバー操作によって，モデルを見る視点を変えられる．

ふつう，MVC については，このことばかりが強調されていて，１番大事な「依存性」の概念が抜けていると思います．これを拡張すると，エンカプセレータとかデーモンになっていくわけですが，そのことを無視した MVC の解説書が多いので，困っています．

どういうことかといえば，同じものを，人によって見方が違うと言うようにとれる．ある人がコントローラをいじれば，モデルの内容が変ります．その時に，違う視点から同じものを見ていた人に対しても，眺めが変化して欲しいわけです．そうすると，普通の MVC ではダメて，モデルに対してビュー，コントローラが依存しているような形態が必要になります．

それが，Smalltalk の中には用意されています．コントローラによって変化が起こると，自分が変化したよと伝えて，それが見ている人達にブロードキャストされていくという形です．MVC 自体はそんなに大したことはなくて，その先にある依存性とか，それをもっと発展させますとエンカプスレーションみたいなものになるんですが，オブジェクト間の関係を記述するものです．オブジェクト指向をもう少しリレーションまで入れたような発展のしかたです．Smalltalk のマンマシン・インタフェースは，すべてこの方式で作られています．

２．１．３．よりよいMMIのために

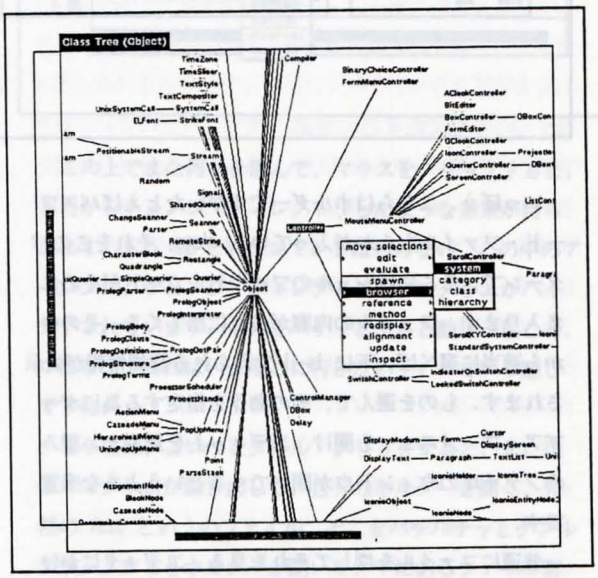
私は，よいマンマシン・インタフェースのためには，拡張性と親和性が重要だろうと思っています．

先程から述べてきたプログラマブルアイコン，プログラマブルイメージでもいいんですけど，何か視覚的なものにプログラムを束縛できることが，この拡張性を生みだします．１つの絵を書いて，それに対してプログラムを自分で作れる作ってバインドできる，束縛できるという構造が必要です．その１番根本には，先程いったようなアルゴリズムをデータとして扱える言語構造が必要だ

と思います．

それともう１点，ダイレクトマニュピレーション，直接操作性と呼ばれているものが重要です．対象となる物を選択してそれに対して動作を指定できるようなものです．私は家に Mac を持っているんですけど，フライト・シュミレータを動かすと，何か自分で飛行機を操縦しているような奮囲気になりますね．直接操作している幹事がやっぱり大事だろうと思います．それは，ユーザにとって，何か本当にその物に触っているという感覚を与えてくれますので，CPU パワーが上がったらここを頑張るという感じですね．

下の図は Smalltalk の小宇宙を示しています．オブジェクトを中心にして，これが１番抽象的なものです．それをどんどん具体化していったクラスの階層構造を示します．オブジェクトとコントローラの間にはサブクラスという関係があるんですけれども，この関係をただ単に線で結んでいるだけです．これを通常ただ線で結んだと考えるだけだと，もうほとんど Smalltalk の世界ではナンセンスで，このオブジェクトとオブジェクトの間にサブクラスという関係があるということを処理しなければいけない．



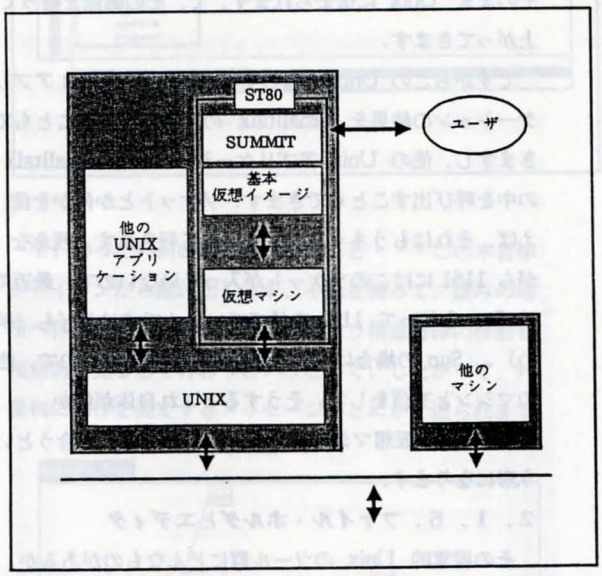
ここに出ている文字列自体は，プログラマ・ブルアイコンになっています．ですからこれは，ただ単に文字がここに書いてあるだけではなくて，これをひっぱたたくと，クラス・ブラウザが開きます．つまり，このアイコンにブラウザーを開くというプログラムがバインドしてある．ですから，ここをダブル・クリックすると，コントローラを持ったクラス・ブラウザーが開いてくるようになっているのです．

この中に沢山のものがありますが，そのうちのどれかを自分で撰ぶ，つまり対象となるものを選択する．その後それに対して動作を指定する．実際には，ポップアップ・メニューを使うのが普通です．ポップアップ・メニューを開けて，例えばブラウザーを開くという順序です．

Smalltalk における MMI は，この形態でほとんど統一されています．Mac と同様に，ツールによって使い方が異なるということがほとんどなく，それを犯すこともタブーとされています．ある意味では，非常にプログラマが苦労する．つまり，ときにはそのタブーを破ることが，本当は必要なんだけど，やっぱり皆に使ってもらうためには，そのフレームワークを守らなければいけないというジレンマに陥ります．

２．１．４．視覚的 Unix の試み

それでは Smalltalk を使って，いったいどんなものを作ったか，１つの事例をお話します．それは， Unix をできるだけ視覚的に見せてあげようという試みです．Summit と呼んでいます．絶頂とか頂上とかいう意味ですが，実は Smalltalk で Unix の MMI を作ったいうだけのことです．



このアイディアのキーになっているのは，Smalltalk の中にある Unix の System Call クラスというものです．Smalltalk の中から Unix のシステムコールが非常にオブ

ジェクトオリエンテッド風に使える状態です．システムコールの中では，最近話題の IPC を使ってあって，ソケット，メッセージキュー，共有メモリ，いろいろ IPC のやり方はあると思うんですけど，そうしたシステムコール群が全部使えます．今までずっと C でやってきたけれど，なにも Unix のシステムプログラミング言語は C ばかりではない，Smalltalk で書けば記述力が１００倍になる（笑い）．

先程のブラウザーをいっぱい出てきたものも，本質的には２日とか３日ぐらいで作れます．これから説明するシステムも，実際にはアイディアを考えて作るのには数日しかかかっていない．その後の時間は，チューンナップやこまかいバグつぶしにかかっているだけで，とりあえず動くものができるのは，非常に早い．これが Smalltalk の特徴です．

内部的な Summit の構造ですけれども，Xerox の 1161 というマシンに Unix が乗っていて，実際にはシステムVです．その上の１プロセスとして Smalltalk が走っています．その中には，仮想マシン，仮想イメージ，その上にこの視覚的 Unix ツールを乗せています．ここととやりとりしているのが Unix System Call クラスと呼ばれているものです．仮想イメージの中にあるんですけれども，仮想マシンの中に入っていて，ほとんど素通りです．そのまま Unix に伝えられます．で，その機能を持って上がってきます．

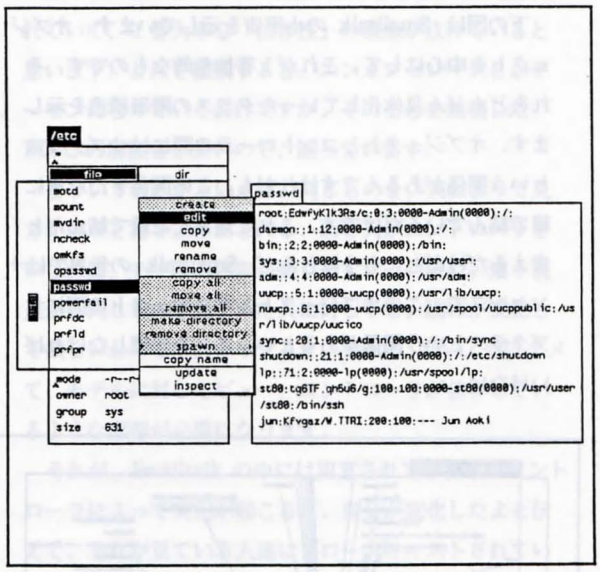
ですからこの Unix を通して IPC，他の Unix アプリケーションの結果を Smalltalk の中に取り込むこともできますし，他の Unix アプリケーションから Smalltalk の中を呼び出すこともできます．ソケットとか何かを使えば，それはもうネットワークに出て行きます．残念ながら 1161 にはこのソケットが入っていないので，最近では Sun を使って 1161 を捨てているんですけれども（笑い）． Sun の場合には，ここから出て行きますので，他のマシンと交信をして，そうするとこれ自体が何か Smalltalk の仮想マシンに見え，それと交信をし合うという形になります．

２．１．５．ファイル・ホルダとエディタ

その視覚的 Unix のツール群にどんなものがあるか，全部は無理ですので，目新しいところだけをご紹介します．

実際には，Unix の特徴はプロセスとファイルにあると思うんですけれど， 1161 というマシンはその辺があまり強くない．プロセスを視覚的にしようと思うとパワーが足りない．今後は，そっちに手をつけてゆこうと思っていますが，とりあえず安易にできるものは何かといえばファイルなので，その中身をできるだけ視覚的にしようと考えてみました．

ファイル・ホルダーと呼ばれているものとエディタを作ってみたんですけれど，エディタはほとんど Smalltalk のパラグラフ・エディタです． Unix の世界ではフォントの幅が一定だというのが普通ですので，そういう風にするようにしただけのものです．

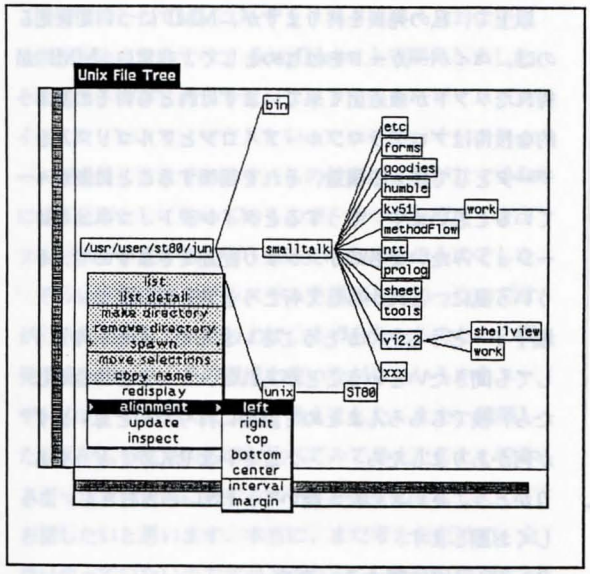


いっぽう，こちらはホルダーですが，たとえばパスワード・ファイルなんかが入っている /etc，それをこのパターンで，ここにはシェルのワイルド・カードがそのまま入ります，ファイルの内容がここに出てくる．その中から適当に選べば，下に ls -l で得られる詳細情報が表示されます．ものを選んで，その動作を指定する為にポップアップ・メニューを開け，エディットを選ぶと，望みのファイルのウィンドウが開いてくるというような感覚です．

普通にファイルを探してそれを見る，エディタにかけるという一連動作をコマンドで ls -l とかをやっていますと，いくらスクロールバーを付けても出力は上に逃げて行ってしまいますので，それ自体をウィンドウとしてホールドすることが大事だと思います．これからの ls 風のコマンドはこういう形が望ましいんじゃないかと思って作ってみたものです．

２．１．６．ファイル・ツリー

それから，ファイル・ツリーですけれども，とにかく人間は５階層以上，相当な天才でも７階層以上のイメージは頭の中にないという話です．それくらいが，記憶の限度らしいんですね．そこで，ディレクトリの階層構造を視覚的に表現してあげようというものです．ここに枠で囲われているのが，実際に私が作業しているディレクトリ構造ですけれども，それを視覚的に現わしています．



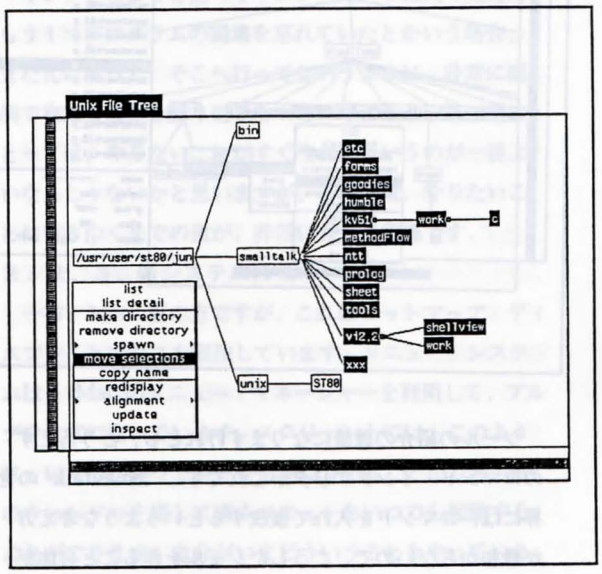
この上でまた何かを選んで，マウスをクリックすると，あたかもチェンジ・ディレクトリしたような効果が得られます．もう cd というコマンドはいらない．この中のアイコンに，チェンジ・ディレクトリのプログラムがバインドされています，それをパチッとやると動作が起こる．つまり，「おれはいまそこに行きたい！」っていう感じですね．

で今度は，こいつをダブル・クリックすると，また違うプログラムが動き出して，自分のホルダーを開く，先程の /etc というのファイル，そこをパチパチッとダブルクリックするとホルダーが開いて，中にあるファイル群が見える．そのプログラムがこのアイコンに束縛されているのです．このメニューは，そうですね Smalltalk と Mac を足したような感じです．選択ができないところはシェードがかかって選べません．一応，ユーザをナビゲーションしている感じですね．それで，こちらからサブメニューが出るようになっていまして，このメニューのことをカスケード・メニューと呼んでいます．

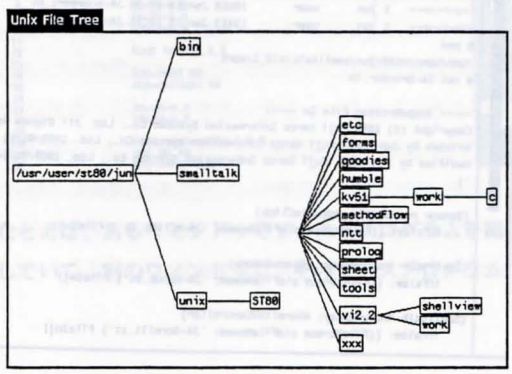
２．１．７　ダイレクト・マニピュレーション

今度はダイレクト・マニュピレーションの話です．ディレクトリ階層全部の移動をいったいどうやって実現するのかを例として，お話します．

システム　V　では　tar　でやるんですけど（笑い）．Summit では，対象物となるものをマウスでクリックするだけです．この図では　Smalltalk　k という文字列のちょっと下の部分です（笑い）．皆が集まったところをカチンとクリックする，そうすると，そこにつながっているサブディレクトリ全部が選択されます．次に，動作を指定するためにメニューを開き，選択したものを移動したいという意志を指示します．

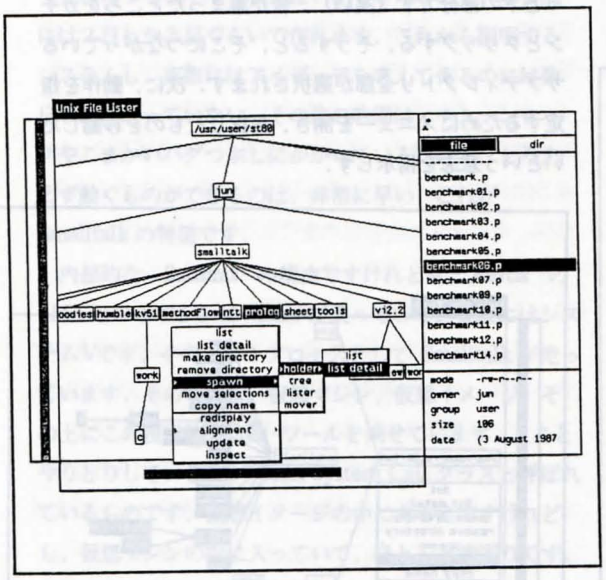


で，いったい何が起こるかというと・・・この木自体がアイコンから離れ出します．それを持って，望みの場所へ移動するだけです．ディリクトリ構造自体の移動も視覚的に見る形で行おうということで，したがって，対象物に動作を規定するメッセージがとにかく送られます．

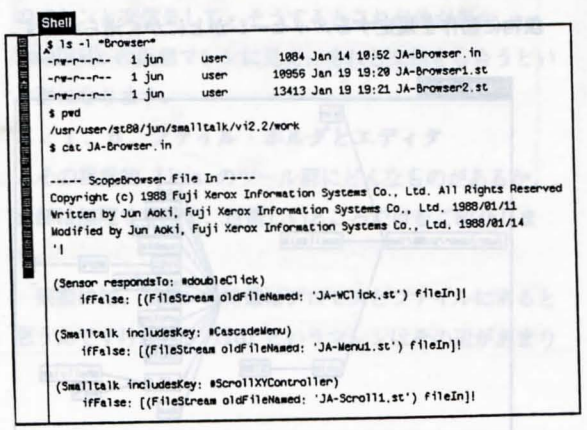


２．１．８．ファイル・リスタとコマンド・インタプリタ

ファイル・リスタと呼ばれるものがあります．これは，前述のフォルダーとツリーをくっつけたようなものです．ここをパシッとやると，その中のファイル群がこちらに現われる．それを選んでポップアップ・メニューを開け，エディットや何かができますす．ツリーとフォルダーの統合，cdやlsやeditますので，それをウィンドゥ・システムに合のスムーズな流れを実現しようと思って作ったものです．



ツールの紹介の最後になりますけれども，とりあえずのコマンド・インタプリタがこれです． Smalltalk の世界には，コマンドを入れて復改するというような考え方が最初からないので，こうしたシェルを作ること自体が，もうナンセンスではあるのすけれども（笑い）． 1161 自体がそんなに強力ではないので，プロセスを視覚的にできないために，仕方なくとりあえずシェルを作ったという感覚です（笑い）．



これはもう，通常のウィンドウ・システムの MMI と思ったらいいですね．たとえば，XとかSunViewとかは，Unix にウィンドウ・システムを乗せていると言っても，本質的にはこれを提供しているだけですから．インタフェースの立場からすれば， Unix のコマンド群の寿命はもう尽きていると思います．しかし，やっぱり過去を引きずってますので，それをウィンドゥ・システムに合うような入出力形態に，もし Unix を使うのであれば，変えなきゃいけないのだろうと（笑い）こう思っている訳です．

以上で，私の発表を終りますが，MMI について言えるのは，ハイパーカードをはじめとして，非常に MMI が秀れだソフトが最近出て来ていますけれども，その基本的な技術はプログラマブル・アイコンとアルゴリズムをデータとして使える構造，それを構築することにかかっていると思います．そうするとダイレクト・マニュピレーションみたいなものがスンナリ記述できますので，そういう風に，これからして行こうと考えています．

坂下： どうもありがとうございました．もし，今どうしても聞きたいということがあれば，１つか２つ位でしたら，後でもちろんまとめて議論に打りたいと思いますが何かありましたら・・・．ございませんか，どうもありがとうございました．続いて，HST の吉村さん，よろしくお願します．

２．２．U/Iに関する一考察

吉村： ＳＥＡのワークショップに参加するのは今回が初めてです，どうぞよろしくお願いします．

ポジション・ペーパにも書きましたが，我が社で，ロボット・プログラミング開発支援システムの再構築するという仕事を行いました．その中で，ユーザ・インタフェースに関していろいろ考えたことを，これからお話したいと思います．

まず，仕事の内容について申し上げると，HST という会社は，自動化省力機メーカーの平田機工から，FA におけるソフト部門の強化を目的として独立した会社です．まだ独立して約１年ですが，これまでに FA 関連のソフトウェアの開発，主に製造管理システムなどをいくつか手がけてきています．

２．２．１ ロボット・プログラミング開発支援システム

私自身，昨年の夏，ロボット・プログラミング開発支援システムを再構築をするという仕事に携わりました．

これは，従来平田機工の方で IBM-PC 上に構築されてきたシステムですが，それを今回新たに Mac に移植するという仕事でした．名前が示す通り，これはロボット・プログラムの開発におけるコーディングからテストまでを支援するものです．対象となるユーザは，主に工場の作業員ですね．これまでコンピュータに一度も触れたことがない，コンピュータって恐いと思っているような人たちです．また，平田機工で生産ラインを作っている技術者たち，今まで IBM-PC 版のシステムを使って仕事をしてきた人たち．そういうユーザを対象としています．

開発環境としては，ユーザ・インタフェースおよび通信部分を除いて，すべて Sony/News 上で開発しました．そこには，御存じのように，ビットマップ・ディスプレイ，Xウィンドゥ，マウスといったユーザ・インタフェースが提供しされています．この仕事は，我が社で Mac に触る仕事として初めてのもので，かつユーザ・インタフェースを作成するという意味でも初めてのものでした．

そういう環境の中でシステムを構築していったのですが，それを通して，たとえば，まず自分たちのための開発環境を作って，その環境を使いながら，いろいろユーザ・インタフェースについて考えました．また，構築したシステムを従来のものと比べてみて考えたこと，できあがったシステムに対するユーザ評価などをまじえて，お話したいと思います．本当に，まだ考えただけで，全然は伴っていないのですが，これから，それを実践する方法についてのヒントを，討論のなかで得られたら幸いです．

では，まず簡単に，従来のものとMac 版の方のシステムを比較説明したいと思います．

２．２．２．従来システムのU/I

これが，従来の IBM-PC 版のユーザ・インタフェースですが，すべてのメニューが階層構造になっています．

Program Name	H-3525
Function	HARL-II Compiler for IBM-PC
Edited on	1987 - 1 - 28 Ver. 3.01
(R) (C) All rights reserved by Hirata Industrial Machineries Co., Ltd.	

```
F1  : EDIT mode ------ Create HARL-II program using IBM-PC BASIC
F2  : COMPILE mode -- Compile HARL-II program
F3  : LINK mode ------ Link HARL-II program
F4  : DEBUG mode ---- Debugging, patching, and verifying HARL-II program
F5  : KEY-IN mode ---- Edit position data, SC data or Definition Table
F6  : SET-UP mode --- Tune your IBM PC by editing set-up file
F7  : FILES  -------- Show HARL-II program file names in disk B:
F8  : LLIST  -------- Print out HARL-II program in disk B:
F9  : HELP   -------- Explanations on HARL-II compiler
F10 : EXIT   -------- Return to IBM PC DOS

Select function key < F1 - F10 > ?

1EDIT  2COMPIL 3LINK  4DEBUG  5KEY-IN 6SET-UP 7FILES  8LLIST  9HELP  10EXIT
```

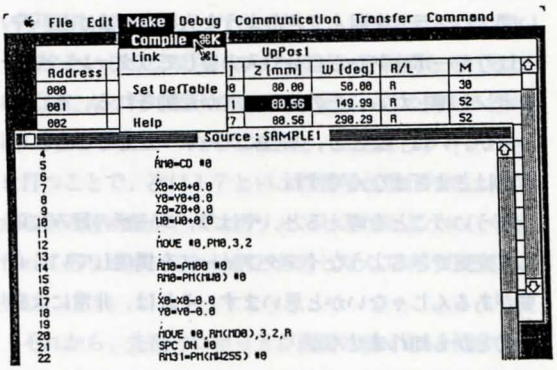
メニューの入力は，ファンクション・キーによるキーボードからの入力のみです．たとえば，システムを立ち上げると，ここに示すような初期メニュー画面が現われます．ロボット・プログラムを編集する場合は，（F1）ファンクション１のキーを押すと，編集モードのメニュー画面が現われて，そこで仕事をします．終了したら，そこで EXIT(F10) キーを押して，初期メニュー画面に戻ります．

次は当然コンパイルでしょうから，コンパイル・モードをここで選びます．そうするとまた，同じようなコンパイル・メニュー画面が現われます．コンパイルが終了したらメイン・メニューに戻って，次は，リンクのために，また同じような操作を繰り返すわけです．

リンクをしようと思ってリンク・モードに入った時に，もう１つプログラムの編集を忘れていたとかいう場合，また元に戻って，そこへ行ってというように，非常に面倒で複雑な操作を繰り返さないといけません．ユーザにとっては，やりたいことがすぐやれるというのが一番よいなんじゃないかと思いますが，これでは，やりたいことに行きつくまでの道が，非常に長くて苦痛です．

２．２．３．新システムの U/I

一方，Mac 版の方ですが，これはビットマップ・ディスプレイとマウスを駆使しています．メニュー・システムは，Mac のメニュー・マネージャーを利用して，プルダウン式になっています．スクリーン上では，このように，いくつかのウィンドゥを重ね合せることができ，別のウィンドゥを通して違うメニューをいつでも起動することができます．自分がいまどういうことをやっているのかを，いつでも把握することができます．



たとえば，あるウィンドゥでソース・プログラムを編集していて，別のウィンドゥで，そのソースプログラム

をコンパイル・リンク・ダウンロードして，実行させながら，あるウィンドゥではロボットの位置決めデータの編集をやってダウンロードさせるといった具合いです．つまり，今までのロボットの動きが，新らしくダウンロードされたデーダにしたがって変わってきます．マルチウィンドゥですので，そういうことが，同時に行えるわけです．

このように，関連する事柄をいつも一緒に見ておきたいとか，気分転換に他の作業をしたいとか，やりたいことをすぐやりたいとかいうのが，普通の人間の気持ちだと思います．マルチウィンドゥのモードレス・メニューは，そういう意味で，私たちの気分を満足させてくれるもののように思います．

２．２．４．ユーザは十人十色

以上のように，ロボット開発支援システムを再構築したわけですが，そのでいろいろユーザ・インタフェースについて考えたこと（先程もいいましたが本当に考えただけなんですが），をちょっと述べたいと思います．まあ，雑感を述べるというような感じですね．

まず，第１に，ユーザはさまざまだということですね．よくいわれるんですが，初心者だったり，経験者だったり，頭のヤワイ人であることもあれば，逆にモノスゴク頭が堅い人であったり（笑い），本当にさまざまなんですね．たとえば，今回のロボット開発支援システムですが，我が社の新入社員は「メニュー・ガイド方式じゃないと非常にやりにくい」とかいうんですね．ところが，以前 IBM-PC を使っていた人は，Mac を見て「わあ，こいつは可愛い」ですとか「こいつはすごい，面白い」とか「工場で作業するのには，こういう可愛いのが必要ですね」とか，結構そういう事をいったりする．でも，長い間プログラムを組んできたような人になりますと「やっぱりキーボードから自分で入力をしてリターン・キーをポンと押したら，そのコマンドが起動される，そういう方がいいね」などと，本当にユーザーが思うことというのはさまざまなんですね．

そういうことを考えると，やはり，ユーザの好みに応じて変更できるようなインタフェースを提供していく必要があるんじゃないかと思います．それは，非常に大変なことかも知れませんが．

また，使いながら自然とユーザ自身が成長して行けるようなインタフェースですとか，逆にユーザの成長に合せてインタフェースそのものも成長していくようになっていれば，なお素晴らしいんじゃないかなと思います．ユーザにとって，習得しやすく，信頼性が高く，発展性があって，しかも自分の好みに合うというインタフェースを，私たちは今後提供して行かなければならないと思います．

２．２．５．利用目的／状況

それから，ユーザがそのシステムをいったい何に使うのかを，しっかり把握しておかなければいけないと思います．今回のシステムでは，我が社の方でハードも選択したのですが，たとえば IBM7532 という非常にドデカイ工業用のマシンがありまして，過去に，そういうものにこのシステムを乗せたわけですね，そうすると，実際の生産ラインというのは，長いものになると２～３００ｍあって，その中にロボットが２～３０台くっついている．その個々のロボット向けにプログラミングをしていくんですが，ある場所で作業していて，今度は向こうへ行かなきゃいけないという時には，何と３人掛りで IBN7532 を運んで行ってました（笑い）．何で IBM を選んだのかというと，IBM は大会社だからだというそれだけの理由だったらしいんです（笑い）．

それが Mac になれば，首にマウスをちょっと引っかけて両手でこう持って行けるわけですね．そういうことを考えてあげるだけで，ユーザさんというのは結構喜んだりしますので，そこまで考えなければいけないと，最近大声でいってるんですけど．

２．２．６．ユーザ心理

それから，当然のことですが，ユーザの心理を忘れてはいけません．たとえば，マルチウィンドゥを例にとれば，これはたしかにいいんですけど，欠点と言ったら悪いんですけどちょっと嫌なところがありますね．それは，仕事をしているうちに，ウィンドゥが山のように重なってくるんですね．その状況は，机の上にいっぱい書類を重ねているのと何ら変らない．あるウィンドゥでテスト仕様書を書いてはずだと思いながら，それを探すのにマウスを何回もカチカチカチやりながら，自分でイライラすることがある．

私の机の上は，月曜日の朝はキレイに片づいていて，土曜日の午後になると山のようにちらかってしまう．それを整理してまた月曜の朝キレイになってというサイクルを繰り返しているんですが，何かその状況と全然変わらないような気がする．

でもマルチウィンドゥの本来の機能は，それなんです

ね．机の上の書類をそのままディスプレイにポンと入れるような TV コマーシャルがありますが，まったくそれでしかない．私たちは，もっとそれを整理したのが欲しいといつも思っていて，たとえばファイル・キャビネットを付けたりするわけですから，マルチウィンドゥに対しても，それを管理するもう１つ上のレベルの何かを考えて行くべきでしょう．書類の山の中に手を突こんで，いろいろ探しまわるのと，マルチウィンドゥのなかをマウスで探しまわるのも，何となく同じことですから，もうちょっと何とかしたいものです．

２．２．７．　U/I と手順・状況

最後に，ユーザ・インタフェースというのは手順・状況そのものだろうと思います．人間は，内容は忘れても状況とか手順とかは覚えていることが結構多い．たとえば，あの書類はあそこに斜めにして置いたとか，そういう状況は，覚えていたりする．ユーザ・インタフェースは，この手順・状況そのもののように思います．

つまりユーザ・インタフェースというのは，覚え易いように考慮されていなければならないと思います．そのために，はユーザ・インタフェースには統一した全体的な思想というのが，１つ必ず貫かれていかなければならないと思います．たとえば，首尾一貫した操作ができるとか，視覚的な連続性が保たれるとかですね．それと手順・状況によって検索ができるようなユーザ・インタフェースが望ましい．それには，ユーザとの対話が，常に保存されていないといけません．そうした過去の記録を学習して，ユーザ好みのインタフェースに変わっていくようなシステムが夢です．

２．２．８．終わりに

以上，いろいろ雑感を述べてきましたが，いいたいことは，どうもふだん私たちは，ユーザ・インタフェース開発する場合に，「今はこれしかない」とかいうふうにあきらめているような気がして，本当にユーザのことを考えているんだろうか，という反省がいつも心のどこかにあります．

最後に，我が社としては，先程も述べたように，この仕事がユーザ・インタフェースを作るという初めての仕事でした．その後，今もう１つ別のシステムを作っているんですが，まだその２つしか経験がありません．どうも会社では，ユーザ・インタフェースがメジャーじゃなくて他の部分で一生懸命頑張って「ユーザ・インタフェースは適当に作ればいいや」なんていうような意識で皆が仕事している，というかそこまで手がまわらないというような感じを受けています．

でもよく考えれば，どんなに機能が充実していても，ユーザ・インタフェースが使い勝手が悪ければ，使ってもらえない．今後我が社でも，ユーザ・インタフェース専門のグループを作って，エキスパートを育てて行かなければいけないと思います．もちろん，センスのない人や，根気のない人は，いくら教えてもダメでしょうが（笑い）．そういうエキスパートの養成を是非やりたいなという考えています．

また，ユーザー・インタフェースのライブラリ化も，是非やって行く必要がある．ユーザ・インタフェースの設計からテストまでに要する時間は，かなり莫大なものです．今のところ，うちは各システムごとに１品生産をしているために，時間がかかっていますが，ライブラリ化によって，時間が大幅に短縮され，その分より高品質のユーザ・インタフェースを考える時間がとれることになります．またライブラリ化されていますと品質面のバラツキも減ってくるでしょう．

２．２．９．Q&A

坂下：　どうもありがとうございました．何かここで質問がありましたら，どうぞ．

Q（佐原）:ソース・プログラム編集画面の OHP で，横に行番号が付いていたのですが，なぜあんなものが付いているのでしょうか？また，ロボットがコントロールされる様子が，画面上でグラフィカルに見れたりすると面白いと思うのですが，そのあたりについてはいかがですか？

吉村：　先ずこの行番号のことですが，Mac のテキスト・エディタ使って，どうしても行番号を付けて欲しいという要求がありましたので（笑い），あのテキスト・エディタをいじりながら，ニューラインを判別して，そこに入れこんで行くわけです．たとえば，この第１６行の次に何か入れたかったとしますと，ここをリターンキーを打つことで，次に１７という行番号が現われて，そこから下の行全部が，インクリメントされるように，今は作ってあります．これは割と苦労したんですけれども（笑い）．

それから，先程のロボットが画面で動くという話ですが，いろいろ考えはしましたが時間との戦いで具体化はしていません．今考えているのは，この種のプログラムは，コンパイルした後，特殊なコントローラにダウンロ

ードして実行されます．今回の開発に際しては，News 上にコントローラのシュミレーション・プログラムを作りました．それは，今の段階では，ただ何かを受けて ACK を返すだの NAK を返すだのということしかやってないのですが，今後は，X ウィンドゥを使って，いかにもロボットがアームを動かすようなシミュレーションをやって行こうというふうな計画を，今立てています．

Q：プログラム言語は特殊なものなのですか？

吉村： 名前は，HARL2 コンパイラです．すなわちヒラタ・アセンブリ・ロボット・ランゲージという言語がありまして，それで実際にロボットを動かすプログラムを書きます．この言語は，いかにも Basic に似ていて，IF 文とかが使えます（笑い）．

坂下： なかなか面白かったんですが，さっきの行番号なんか，いろいろなユーザがいるというまったくよい例だと思いますね．

吉村： 平田機工が，元来 Basic しか使わない会社だったんですね．

坂下： すでに汚染されていたわけですね（笑い）．

吉村： もう「言語は Basic」という固定観念がありまして，で，絶対に行番号を付けてほしいというのが・・・（笑い）．ということで仕方なく付けているわけです．

坂下： なるほど．非常に，なかなか現実の例としては面白かったと思います．どうもありがとうございました（拍手）．

それでは最後に，SRA の新田さんにお願します．

２．３．ERAS

新田： 用意してあるＯＨＰの数が少ないので，なるべく節約して使って行きます（笑い）．なぜかというと，今回発表対象のツールが，実はまだ未完成で，一昨夜もコンパイルしていました（笑い）．そういうわけで，ウソの絵を見せるよりは Mac II 上のデモを明日見ていただいた方がいいという意味です．

２．３．１．Mac について

最初のＯＨＰをお見せする前に，Mac について少しお話しようと思います．たとえば，一般には，Smalltalk がよくユーザ・インタフェースのよい例に上げられるんですが，Mac を持ってから感じたことは， Smalltalk のユーザ・インタフェースは非常にヒドかったな（笑い）ということです．

Unix を初めて触った瞬間に，昔のというか，大型機のＯＳってヒドかったなあと思うのと同じように， Mac を使ってみると Smalltalk はヒドイ．どこがヒドイんだと反論される方が多分いらっしゃると思うので，例を上げておきましょう．

ファイル・アウトという処理があります． Smalltalk のイメージの中に持っているプログラムなりデータをディスクに書き込む作業です．それをやろうとして，メニューからファイル・アウトを選びます．すると，フロッピー・ディスクを入れなさいといわれる．そこで，フロッピー・ディスクを入れる．すると，このフロッピーフォーマットしてない，ダメダいって，やらしてくれない．

フォーマットするのにはどうすればよいかというと，Smalltalk の中からはできない（笑い）．しかたなく，一旦 Smalltalk を抜けて，上のＯＳに戻る．そこでディスクをフォーマットするんですが，何枚フォーマットしてよいのか，その時点では私にはわからない（笑い），適当に４枚位でよいだろうと思ってフォーマットして，再びファイルアウトしていく．すると，４枚では入り切れずに，５枚目を入れなさいといってくる，それはもうヒドイものです（大笑い）．

どういうことをいいたかったかというと，概念的に閉じた世界でユーザ・インタフェースのよしあしを論ずるのではなくて，これからは周辺機器その他，人間が手に触れるものはすべてシステムの要素として，インタフェースのことを考えて行かなきゃいけないんじゃないかと，私は思います．

Mac のことをもう少し話しますと，人の人生というのは，いろんな人と出合うとかあるいは本を読むとかで変わると思うんですけれども，私の場合は Macintosh を買ったことで，すべてが変わってしまった（笑い）．どういうふうに変わったかというと，あれを買ったことで私のプログラマとしての寿命がかなり縮まってしまった．どうしてといえば，Mac を知らなければ，あれが世の中になければ，ああいうものを開発するために，あと何年か仕事ができたわけですけれども，その分が減ってしまった．そろそろ私のプログラマ寿命も尽きるんじゃないかなと思うので，ここら最終兵器を作っておこうと思っています，

その最終兵器こそが，今からお話しする ERAS というツールです．

２．３．２．背景

この ERAS の歴史をちょっと述べると，３～４年前にハワイ大学で仕事をしていたんですけれども，その時に

ERA（時代）という名前のエディタを作りました．

それはどういうエディタかというと，いろんな事柄をアイコンで表わして，エンティティ（実体）と呼びます．いろんなエンティティとその間に成り立っているリレーション（関係），それとそれらの性質を数値とか言葉とかで表わしたアトリビュート（属性）の３つを使って，世の中のものを何でも表わしてしまおうというものです．

この，エンティティ・リレーション・アトリビュート頭文字をとって，ERAというんです．このエディタのもう１つの特徴は，何だったかというと，どういう絵を描くのかというあたりまで，ユーザが決めることができるというシステムでした．

どうしてそういうふうな複雑なものを考えたかというと，ハワイ大学に宮本勲先生という人がいて，生徒に対しても無茶苦茶な要求を出すんですけれども（笑い），私に対しても変な要求を出しまして，カーネギーメロンにＡＬＯＥというランゲージ・オリエンテッド・エディタのジェネレータがあって，それに言語の仕様と動作を入れれば，その言語でプログラミングをサポートするエディタをジェネレートしてくれる，そのグラフィック版を作ったらどうかという，ほとんど不可能ともいえる（笑い）要求が来ました．ところが，はじめ私は不可能と思っていたんですけれども，Mac を使っていろいろ試行錯誤している内に，これはできると確信して作ったわけです．

もう一度 ERA の特徴をいうと，エンティティとリレーションとアトリビュートでものを記述しようということ，それから単に絵として描くのではなくて，何か意味を持っている，つまり絵を描く上でのルールがあって，そのルールに対して正しく描いてあるか，そうでないかという意味のチェックもしてくれる．そしてその意味を判断するというか，ルールを判断する基準は，ユーザが自分で定義できるようにしようというものでした．

これが，だんだん発展してきて，先程熊谷さんがちょっとおっしゃったプロタイパーズ・ワークベンチという，これもソフトウェア開発の要求定義をプロトタイピングするための仕掛けを作るという，正にすごいプロジェクトでしたが（笑い），その時にユーザの要求を絵でモデル化して，それをプロトタイピングと呼ぼうと考えた．しかし，どういう絵を描いたらよいか，最初のうちはわからなかった，で，その時にこの ERA という汎用のモデリング・エディタを作ったわけです．その時点で，ERAは実はMacの上でしか動いていなかったんですが，宮本先生の真似をして，１人の学生にソースコードを見せて「これを Smalltalk に書き直せ」と要求したら，やってくれました（笑い）．

２．２．３．ERAS の狙い

以上が ERA の歴史ですが，で，どうしてそれに S がついたかというと，さっきちょっといったように，もうそろそろ最後のツールをと思って，ERAのスペシャル版，スーパー・バージョンという意味で，お手持の黄色いファイルには，SUPERA と書いてありますけれども，それじゃああんまりなので ERAS にしました（笑い）．このSの意味は，明日のデモをご覧になればわかるはずですが，本当は Scene（場面）の S です．いろんなものをＥＲＡ方式で描くと，ごちゃごちゃしてくるんですね．そこで，「場面」という概念を新たに導入して，このシーンではこれとこれだけが見えるようにしようというふうな意味合いです．いずれは，もっと発展して S の後に A とか O がつき，「偉そう」とか「偉さ」（笑い）に結び付くはずです．

それで，何をしようとしているかといえば，今はただスタティックな絵を描いているんですけれども，いずれは，描いた絵をそのまま動かしてしまおうと考えています．動かすためのアルゴリズムは，大体もうわかかっています．そうすると，ERAS は，それ自体がエディタであり，プログラミング環境であり，実行機構にもなる．それを作ってしまえば，あとはもう何もいらないというようにしよう，と考えています．

２．３．４．U/I 設計の基本方針

ERAS にもユーザ・インタフェースがあります．どういう考えを基本にして，そのユーザ・インタフェースを作ったかというと，次の４つです：

(1) メニュー＋ウィンドー，キーボード＋マウス
(2) 有効なコマンドだけ選べる＋いつでも全部のコマンドが見える．
(3極力モードレス．モードへの出入りをいしきさせない．
(4) コマンドの数を少なく，操作の種類＋操作対象の組み合わせ．

この４つは，実は，Macのアプリケーションの作り方に，こういうことを守ってやりなさいと，きつく書かれているものです．

まず，ユーザがわかるものはメニューとウィンドゥ，それから手で触るものはキーボードとマウス，もちろん

フロッピーも触ります．Mac の場合は，フォーマットが必要な場合はフォーマットをしてくれます（笑い）．プロセスの途中でフォーマットをしてくれて，続いて仕事をやってくれるということです．

メニューについてですが，ユーザがこのツールを使っていて，ある所に来ています，そこではすべてのコマンドが使えるわけではない．使えるコマンドだけが選らべる．先程，青木さんの Smalltalk のメニューでシェードがかかっていて選らべないというのがありましたけれども，Mac でも同じです．しかし，いつでもすべてのコマンドが見えている．つまり，これだけのコマンドが用意されているんだけれども，今この場合では使えませんよ，ということがユーザにわかるようになっています．

それから，いわゆるモードをなくします．たとえば，今から名前とかテキストを書換えるからテキスト・エディットのモードに入る，今からエンティティを作るからエンティティを作るモードに入る，終ったらモードから出てくる，といった切り換えをなくしました．ユーザから見てなくしたという事で，プログラムの中ではちゃんと区別があるんだけれども，あることをしようとすると自然にそのモードへ入るというふうにしてあります．

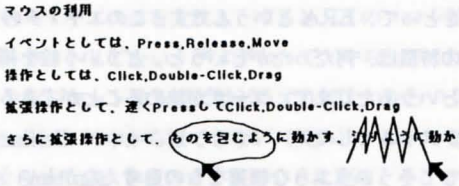
コマンドの数は，できるだけ少なくしようと考えています．たとえば，ファイルを消去するのとディレクトリを消去するのと，２種類のコマンドは必要ないわけで，ファイルに消えろと云えばファイルが消えるし，ディレクトリに消えろと云えばディレクトリが消えればよい．そういうふうにしてコマンドの数を減らす．その代り操作の種類と対象物で，何をするかということを判断する．

以上は，基本と思っていることで，既にやっていることです．これからいっぱいしたいことがあるんですけど，その中の１つでマウスの利用ということを，ちょっと説明したいと思います．

２．３．５．マウスについて

マウスには，ご存じの通りボタンがあって，ボタンを押す，はなす，それからマウスを動かすという３つの操作，イベントを起こすことができます．それを，人間の方から見た操作として分類すると，クリックする，ダブルクリックする，ドラッグするということになります．つまり，コンピュータのプリムティブなイベントと，人間のものの見方は異なる．

すでに，Mac や Smalltalk がやっていることの他に，拡張機能として，速く押すとか，ゆっくり押すとかの区別を取入れてみました．人間はそういうことができる．キーを速く押すとか，ゆっくり押すとか，実際身の回りにある機械を操作する時には，そうしている．たとえば，壊れそうなものは軽く持つとか，重い物はしっかり持つとかしています．それが，マウスやキーボードになると，持つか持たないか，押すか押さないかという単純なコードに変換されてしまう．



これはどうもまずいんじゃないか．我々の仕事の手助けをするはずのコンピュータが，逆に仕事を制限している．そこで，速く押すとかゆっくり押すとかいうことを考えました．本当はマウス自体，と云うかハードウェアがもっと進歩して，そのスピードを検出してくれればいいんですけれども，今はまだ，残念ながら Mac のマウスも，Smalltalk のマウスも，そういうことをしてくれませんので，自分の中で押された時間と離された時間を見て，区別する．

実際にどう使うかといえば，たとえば，あるアイコンを速くカチッと押してパッと持ってくると，ここに移動する．今度は，ゆっくり押す．するとカーソルの形が変わる．で，ピッと持ってくるとここにリレーションが張れる．つまり，持って来るのとつなぐのとは，別の動作だという使い分けができるわけです．

将来は，さらにどういうことができればよいかというと，マウスの動きによって動作の種類まで表現してしまいたい．今は，対象物を選んでメニューから動作を選んでいますが，我々は普通の生活の中では，そんなことは特に意識していません．

たとえば，字を消したい時は消したい字を選んでから消しゴムを取るかというと，そうじゃない（笑い）．消しゴムを持って来て適当にガチャガチャやる．同じように人間の動作そのものを，やりたいことに変換してしまいたい．つまり，例えばこうやって物を包むように動かすと，選んでそれから包むというふうなメニューを選ぶ．消したい時には，ガシャガシャガシャとマウスを動かすと，その文字列を選んでイレーズというメニューを選ぶ

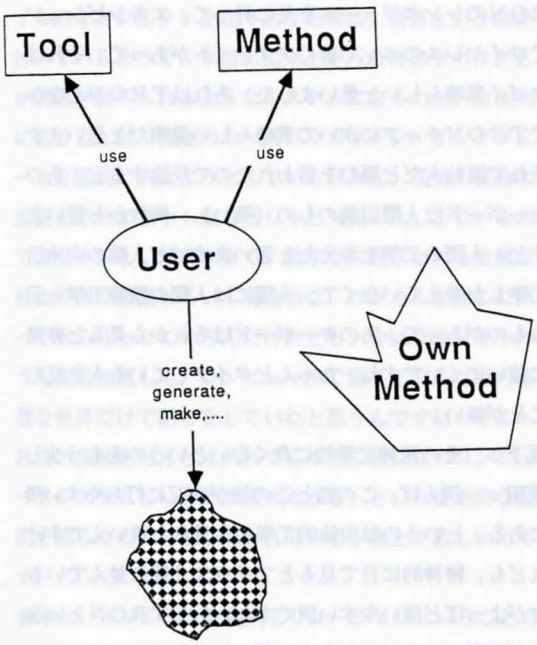
のと同じような操作ができればよい．

つまり，ユーザ・インタフェースとしての人間が，いろんなことができるのに，その選択の幅を切ってしまわないで，いろいろな違いをコンピュータに伝えたいなとに思っています．

２．３．６．終わりに

今までお話ししたことは，ツールの操作という面でのユーザ・インタフェースの話題でしたが，最後にもっと大きな視点で，つまり我々人間の世界にツールがどう入ってくるかということ，逆にいえば，たとえばソフトウェア開発の環境があって，その中に人間がどういうふうに入って行くかというふうな意味で，ユーザ・インタフェースについて，ちょっと意見をいいたいと思います．

ユーザがツールを使う．ある方法論を使う．ツールと方法論が密接に結びついている場合もあるし，全然無関係な場合もある．たとえば，ソフトウェアの開発の能率を上げたいというので，それを目標にしたツールが多くなって来ています．技術者はそれらを使って，開発作業をやって行くわけですが，自分で今までやって来た仕事のやり方とか方法論を持っているので，まったく別の方法論を支援するツールを使いなさいといわれても，なかなかむずかしい．



ERAS では，そこをどうしたかというと，さっきいったように，ユーザ自身がルールを定義する，つまりERAS は，メタ方法論しか提供しませんよ，あなたが今までやってきたノウハウをここに入れて下さい．それで，全体を貴方の仕事の役に立てて下さいという方針です．多分そういうふうなものが，使う人のレベルにもよるんでけど，これからのツールにとって１つの大切な方向じゃあないかと考えています．

それで，今，ERAS の開発が遅れているのはなぜかというと，ツールとメソッドを切り離してしまったんですが，ツールを美しくする為に，自分自身をルールで定義してしまおうと，それをメタルールと呼んでいるんですけれども，つまり自分自身で自分をモデル化してしまおうと考えているのです．蛇が自分の尻尾を食べているようなものなんですけれども，いろいろむずかしいことが起りまして，あっちを直すとこっちも直さないといけない．それで遅れているんですけれども，方針としてはなるべくツールと方法論を切り離してしまう，つまりペンチで釘を打ってもいいんではないかと云うふうな考えです（笑い）．

ちょっと話が中途半端で終ってしまいましたが，以上です．

（拍手）

坂下： なかなか面白いお話だったんですが，何か質問はございませんでしょうか

Q： ERAS のルールを ERAS で記述するということですが，もう少し詳しく説明してください．

新田： 実はブートルールがありまして，ブートルールで自分が作ったメタルールを読み込みます．するとある程度のメタルールが出来て，つまりすべてのことは書けないんだけれども，あることは正確に書けるというルールをまず作っておきます．それで今から作ろうとするものをデバッグするわけです． Mac の上で作っているのですが，MAc 上のアプリケーションというかファイルはトリッキーで，２つの部分から成り立っているんですね，すべてのファイルが．１つはソース部分と呼んでいて，１つはデータ部分と呼んでいます．ソース部分の中にプログラムが入っています．データ部分は普通我々がデータと呼んでいるものが入っている．アプリケーションは普通，データは空っぽで，ソース部分がいっぱい．普通のデータファイルはデータがいっぱいで，ソース部分はゼロなんですけれど，ERAS は自分のメタルールを自分のデータフォークに持っている．

それで，さっき云いましたブートルールで新しいルールを作って cat するんです．これは Unix の cat と同じ

で，すると新しい ERAS ができて，それで今作ったメタルールを読み込んで，シンタックスチェックをやって，そういうふうなデバッグをしています．

Q：インプリメントで特殊なことをしているのですか？

新田：　プログラムの部分では，ルールにディペンドする部分とインディペンドの部分に分かれて，ユーザ・インタフェースはどんなルールになっても一定，ということになっています．関わるのは，最初にＥＲＡＳが動くと自分自身をもう一度読むんですけれども，その時にルールをもう一度コンパイルするんです．自分で自分をコンパイルして動き出すんですけれども，そのコンパイラの部分がメタルールにディペンドしているだけで，あとは全部フリーです．

Q：その点とユーザが自分のルールを定義できることとの関係は？

新田：　ERAS については，ユーザが自分でルールを定義すると云いましたけど，どうやって定義するかと云うと，それを ERA のダイアグラムで描く，ということです．それで，ERA が立ち上がった時に先ず自分自身のメタルールを読んで，コンパイルして，自分自身の動作を決めます．それからユーザに「貴方が今から書きたい図式は，どのルールに則りますか」って聞いて来ますから，ユーザは自分で作ったルールを指定して．これで書きますって．

坂下：それでは約１０分程休憩を入れてから，ディスカッションへ移りたいと思います．

３．討論

坂下：　それではそろそろ討論に移りたいと思います．

林：　プログラム委員長の林です（笑い）．全体を通じて，もうキャラクタ端末をベースにしてユーザ・インタフェースを語るのは江戸時代の話しになったなと云うのが感想でして，大変良いことだと思います．良いことだと思うんですが１つ疑問に思っていたことは，ＴＲＯＮの坂村　健さんがよくいうことですが，マウスで何かすると云うのは原始人のやることで，石で絵を書いているのと同じで，あんなんで良いのかという話しがあるんですが，発表者全員の方が大体マウスベースの話しをされたんで，その点についてはどう思われるかお聞きしたい．

青木：　Smalltalk の立場からなんですが，先程ＭＶＣのお話しをしました．ただ単に Smalltalk の中ではコントローラというユーザ・インタフェースの部分を司るものがあるんですが，それはただ単に今マウスにバインドされている，ただ単にマウスとなっているというだけです．ですからもしマウスよりも良いインタフェースがハード的に開発されたり，またそう云うものが出れば，そこを置換えるだけですので，　Smalltalk　の立場からすると問題無いんですね．どんなものもオブジェクトに見ていますので，そういうところはユニフォームに出来ています．

吉村：　私自身も，今風潮としてビットマップディスプレイとマウスさえあれば，良いユーザ・インタフェースが提供されると云う様な風潮があるような気がするんですが．何かそれはちょっと違っていて，例えばいろいろウィンドゥを出して操作している際に，先程も云いましたが，何かしたいところにいちいちマウスを持って行くのは，すごく煩わしかったりする訳ですね．もうちょっと，例えば音声で判断するとか，視線のやり方で出来るだとか，そう云うのがあったらスゴイなと思うんですけど．

新田：　ＴＲＯＮの悪口を云ってもいいかもしれない・・？

坂下：　どうぞ，いいんじゃないですか（笑い）．

新田：　確かにマウスは石鹸で書いています．去年のＴＲＯＮのシンポジュームを見に行って，エキシビションでワイヤレスのペンで書くタブレットがあって，それはスゴイ素晴らしいと思いました．あれはＴＲＯＮのなかでＴＲＯＮチップに次いで素晴らしい発明だと思います．それで原始人だと悪口を言われたので反論すると，あのキーボードは人間以前のもの（笑い）．何故かと言いますと，人間の工学を考えたと言いますけど人間の肉体的工学しか考えていなくて，人間には人間の精神工学と云うものがあって，あのキーボードはそれから見ると非常に使いにくいですね．ちゃんとタイプしている人を見たことが無い．

坂下：　その精神工学的に良くないというのをもう少し．

新田：　例えば，この指とこの指が交互に打ちやすい所にある，というのは肉体的工学から見ると良いんですけれども，精神的に目で見るとアイウエオ順に並んでいる方がよっぽど使いやすい訳ですね．もしＴＲＯＮというのが専門のタイピストをターゲットとしているキーボードであれば，あれですごく素晴らしいと思います．でもあれは，もっと広く一般の人が使おうとしているというふうに提唱していますから，そういう人達にとっては少

し位指が遅く動いてもアイウエオ順に並んでいるとか，ＡＢＣ順に並んでいるのが１番良いやり方じゃないかなと思っています．

臼井： プログラム委員長の臼井です．マンマシン・インタフェースのお話し，殆どマルチウィンドゥ・ベースの話が多かったんですが，例えばマルチウィンドゥを使ってマンマシン・インタフェースを良くしようということなんですが，先程の青木さんのお話しにあったように，机の上のゴミの山と変わらないようになって行く状況とか，世の中にある実物と同じように奇麗なものを操作して動かそうとする，そういうあたりが非常に使いやすいインタフェースじゃないかという方へ，何となく収束しておるように思うんですけど．本当に世の中に既にある物をわざわざコンピュータの中に絵を入れて押えないといけない，ということは全く進歩したことになっていないと思うんですね．それと例えば，オブジェクト指向とそれを繋ぐ，関係付けるデーモンの話しなんですけれど．例えば私が今話していることは，１人１人のオブジェクトにメッセージを送っていることになっていると思いますが，その１人１人は私の話しを聞いて全部違う解釈をしていると思うんですね．それをいろいろなネットで関係付けられていると思うんですが，その関係付けられた１つのオブジェクトを取りあげた時に，何百という糸が付いているようで，本当に出来上がった時にデバッグとか関連が調べられるようなシステムになるのか．全体的に思うことは，機械の中へ良いマンマシン・インタフェースを図式で入れようとしているけれど，現実の社会の混乱をそのまま中へ放り込もうと（笑い），しているのではないかと．結局今まで，数学なら数学という奇麗な社会の中で論議していて理論的な，ロジカルな話しに持って行こうとしているんだけれども，やはり今はそういう現実を無視して，コンピュータで動くところだけの奇麗な世界だけで話しをしていたと思うんですね．そこへ社会の混乱をどんどん入れても本当に役に立つコンピュータシステムというのが出来るかな，という根本的な疑問を感じたので，どう思われているか教えて欲しいんです．

青木： オブジェクト指向とリレーションということでお話ししましたが，オブジェクトを沢山作りまして，オブジェクトとオブジェクトの関係を選り分けることができるものが必要でしょう．つまり自分の持っているテリトリ，自分が見れる範囲を，関係をプロジェクト出来る，ある自分のテリトリを持ってその中で見ているものだけが見えるような機構を作って行かなければと思っています．ですから今度はリレーションをマニュピレイトする，そういう時になって来るとＰｒｏｌｏｇみたいなものが力を出して来ると思うんですが，そういう傾向で言語屋としては考えています．

新田： 直接の答にはならないんですが，１つは現実世界をコンピュータの世界に入れているということですが，例えば私の机の上がちらかっていますけれど，どんなにちらかっていても欲しいファイルはちょっと探すだけで手に入る訳ですね．ところがそれをコンピュータの中へ入れてしまうと，なかなか探すのは難しいです．現実の世界はいろいろな混乱がありますが，人は何とか巧くやっている（笑い）．それをコンピュータの中へ入れた瞬間にどうも巧くいかなくなる，ということがあると思うんです．それはどうしてかと考え，私が思うには，現実世界にオブジェクトというのは境目がデジタルではなくて，段々と境目が少なくなって行って，また他のオブジェクトの境目が始まる．その境目と境目は交じり合っていることもあると思うんですね．それをモデルにしてコンピュータの世界の中へ入れてしまうと，ここからここまでが１つというふうに，机のオブジェクトの存在になってしまう．つまり混乱しながら巧くやっていることを，コンピュータの中へ持ち込んだ時に何か巧くいかないというふうなことだと思います．

落水： 新田さんの話しの中で，人間には不器用な人も器用な人もいますから，それに合せて調整するという話しは判かりますが，動かし方とか動作とか基準のはっきりしないものをマンマシン・インタフェースの基準にするというのは，人によって速度が違うから判からないでしょうし，こうやって消すと云うんですが，カーソルが何処かへ飛んで行ってしまって戻そうとした時に全部消えてしまわないのですか．

新田： ＴＲＯＮの話しをすると，答えるためにするんですが，車の運転はどの車でも標準化されているのに，なぜコンピュータは標準化されていないのかと坂村先生はおっしゃるんですが，僕が思うのには，ハンドルを同じ角度だけ曲げた時に全ての車が同じだけ曲がるかというとそうではない．つまり，車によってバラバラで，アクセルを同じ角度だけ踏むと同じだけスピードが出るかというとそうでもない．そう云う意味においては車というのはバラバラなんですね．ところが人間というのは車

を巧く運転しています．それは人が学習する能力がある，フィードバックの能力があるからだと思っています．それで先程言った早く押すとか，ガシャガシャガシャとやるのも，私がＥＲＡＳに組み込もうとしているのは，人が学習出来るようにしようと思ってやっています．つまり，ガシャガシャってやって自分は向こうに持って行くつもりだったんだが消えちゃった，あこれは間違いなんだな，勿論消えたらすぐ元へ戻る機能を入れておきますが，今度ガチャガチャとやったらうまく消えたじゃあこれからこういうふうにやれば消えるんだなと．勿論ある幅を持たせておきますが．そういうふうにして，早く押す押さないも，ゆっくり押すとカーソルの形が変わって，これだけ押せばこうなるんだなと人が発見できる，学習できるようなユーザ・インタフェースにしたいと思っています．

井上： その話しの続きなんですが，コンピュータの方が勉強するというのは無いんですか．例を上げれば，消しゴムが２種類あって，消してみたらＡの消しゴムはよく消え，Ｂの消しゴムは消えが悪いとすれば，人間はＡの消しゴムしか使わなくなると思うんですが．コンピュータの方で使おうと思ったら，Ｂの消しゴムタイプだったら何となく馴らされるのがいやだという人が出てくると思うんですけど，そういうことは考えませんか．

新田： 考えています．それで対処法がありまして，今作っているＥＲＡＳでは，ゆっくり押すとか早く押すとかいいましたが，その時間を外から変えられるように，そんなに長い時間ではないんですが，手で押して何か間隔が違うなという３つか４つの中から１つを選ぶというふうにしています．それで自分の好みを伝えられるようにしてあります．

坂下： その選び方にもインタフェースがあるんですか．

新田： マッキントッシュにコントロールパネルというのがありまして，マウスの動くスピードとかキーボードのキーのリピートのスピードとかを，１つのアプリケーションではなくて，そのマシンに対して人が自分の好みに合せてある程度調節できるんですが，その中でスピードを変えられるようにしてあります．

中村： こう話しを聞いていますと，マウスは言わば我々の鉛筆でありまして，鉛筆がなかなかこの頃使いやすくなったという感じです．言葉を変えて言うなら木彫りをやるのに，一刀彫りと各種セットの揃ったのを使うのと２通りある訳ですね．言わばツールの方と言うかコンピュータの方が非常にセンスが良くて，逆におせっかいな，こんなことまで要らないのにと云うような機能を持たせるのか，もっとシンプルにさせて人間の方がそれにアダプトしてしまう．靴に足を合せるのか足に靴を合せるのかという話しもありますが，現実の社会の中には両方あるわけでして，それぞれ特徴を持っている．それなら両方選らべるようにしてしまえ・・というのは非常に日本人的発想かもしれませんが，本来どちらであるべきなのか決めてしまうのは冒涜かもしれませんが，その辺を考えてみるべきじゃないか．今マンマシン・インタフェースというと非常に多種多彩なものが出てきて，確かに便利そうに見えるんですが，果してそれで仕事をする時に便利なのかと思う時があるんですよね．その辺をどうお考えなのかをお答え頂たい．

青木： 先程プログラマブルというお話しをしましたが，全くそれと同義ですね．マンマシン・インタフェースの拡張ということでプログラマブルのアイコンとかを挙げているわけですが，中にバインドできるプログラムの質，それが一刀彫りであったり多様なセットであったり，それはユーザが選らべるような，それを提供するシステムがあるんであれば，そういうものを使ってその上に一刀彫りのインタフェースを付けて出されるのが良いんじゃないかなあと，今の現状で考えられる１番の手段だろうと思っています．

新田： 私はさっきMac の上で仕事をしていると云いましたが，実はMac の上でＳｈｅｌｌとｖｉとＣコンパイラを動かして，コマンド・インタプリタで仕事をしています．要するに一刀彫りでやっているということですね．人によって基本的には好きなものを選らべば良い，誰からも選らばれないツールは消えて行く，そういうことじゃないかと思います．

佐原： 皆さのお話しではマルチウィンドゥとマウスは何かおかしいけれど，取りあえず使いやすいね，という論議が今まであったように思います．私は使いやすいのは確かなんですが，それだけではなくて，今Mac とUnix 両方使っていてどちらも使いやすいんですが，どちらも使いにくいところがあるんですね．それは目的によって違って，Mac はとりあえずひらめいた事を何かやる時は使いやすいけれど，云わゆるバッチ処理的な大量に同じことをやる時は使いにくい面があります．で，その辺のユーザ・インタフェースが２つある，その１つの方しか皆さんしていないんじゃないかということと，先程

臼井さんがおっしゃった人間のモデルを混乱状態のまま取り込むのはイカンとおっしゃるんですが．現在のコンピュータシステムというのは，何もモデルが無い状態でユーザ・インタフェースが作られている訳で，それよりは人間の持っているモデルの方がまだ現在ではマシだ，ということから人間のモデルを使おうかという状態なんじゃないかと思うんですね．で，混乱を持ち込むのは人間の方が悪いんで，コンピュータとかソフトウェアが悪い訳じゃないという気がしています．で，伺いたいのは，先ず青木さんにどうして Smalltalk で Unix をいじろうとしたのか．私が考えるのは Unix はそれなりに使いやすいが過去の異物になりつつあるわけで，だったら別に Smalltalk １ｔａｌｋで一生懸命 Unix の面倒を見る必要が無いんじゃないかな（笑い），ということです．それから吉村さんには，先程の行番号のお話しで，ああいうところでやっぱり妥協してはいけないんじゃないか（笑い）．行番号を無くすというのがMac とか Unix の１つの思想なわけで，そこを曲げちゃったらユーザ・イタフェースやなんかのモデル化自体が，実は意味を無くしてしまうんではないかという気がするわけです．

青木： どうして Smalltalk で Unix のインタフェースなんかという話しですが，ＸＥＲＯＸ自体独自ＯＳを持っていまして，全然 Unix なんて目をくれなかったというのが現状です．それだと，これだけ Unix が広まってしまったのでＸＥＲＯＸマシンが売れなくなった（笑い），それで Unix マシンの上に Smalltalk １ｔａｌｋのバーチャルマシンを乗せようかというプロジェクトが起こってくる訳です．世の中に流されている訳ですが，そういう状態でどうしてそれを作ったかと云うと， Smalltalk とか Jstar の環境に慣れ親しんだ人達が，なじめ無かったんですね．佐原さんが云われるように Unix の好きだけどマッキントッシュも好きだという，両方できる人はいいんですが，マッキントッシュ風の，つまり Smalltalk とか Jstar 風の環境に慣れていた人達は，とてもＵＮＩＸに移れなかったんです．その人達の為に昔 Unix をやってＣ言語が分る人間が，たまたま居たので私が作ったという感じです（笑い）．

吉村： 行番号の話しですが，開発しながら確かにMac の思想と行番号は合ってないなと思っていました．でも，使うユーザの立場を先ず考えたいというのが１番にありまして，どうしても行番号を付けて欲しいと云う訳ですね．こちらは行番号は要らないと云うと，向こうはずっとＢａｓｉｃに慣れ親しんで来た人達なので，どうしてもＧＯＴＯ１０をしたいとかＧＯＳＵＢ１０００をしたいんだというのが，先ず１番先に出てきたことなので敢えて今回は行番号を付けると．やはりユーザのことを１番に考えたまでのことです（笑い）．

佐原： 表面的なユーザの要求に応えるだけだと生産効率がせいぜい２倍がいいとこでＧＯＳＵＢ１０００というのは当初のマイクロソフト系のことであって，現在は改良されて来ているのに，それを元に戻してしまうのは Mac を使う世界では悲しいんじゃないかな．そういうユーザはなんとか説き伏せてやった方が良いのではないか．私なんかＣｏｂｏｌの構文エディタを作るプロジェクトをやっていて，行番号を付けろという圧力があって，使わなくてもいいから付けるなといって，結局使われなかったんですが（笑い）．使われないということは，彼等が亡びて行くんだからそれでもかまわない，と思っている位ですから，それ位過激にやって頂たいと思います（笑い）．

吉村： 今回は確かに力不足で，行番号なんて要りませんよと言ったんですが，ダメだったんで．今度は少し強硬にやってみます．

新田： 僕は少し佐原さんと違う考えで，例えばデジタルウオッチで時間が判かるのに，アナログを好む人がいるように，やはり昔からのことを好む人が居るんだと思います．そういう人達のことを考えてあげなければいけないというふうな気がします．それで，行番号を無くすにはどうすれば良いかと言うと，このツールは巧く動くんですが行番号を使っていると時々落ちることがあります（笑い）．というふうにすると，皆使わなくなると思います．

深瀬： 今の説明をさせて頂ます．多分お使いになっているＢａｓｉｃは元々小さく作られたＢａｓｉｃなんですよね．あれは８バイトだからこそああせざるを得なかったという環境がある訳なんです．Mac の上で動くＢａｓｉｃインタプリタなりコンパイラはその必要は全く無い訳です．構造化Ｂａｓｉｃをちゃんとサポートできる訳ですから，無ければしょうがないけれども，有るものを使わないと云うのは僕はちょっとおかしいなと思います．そういう歴史的な背景はあるんですけど，単なる商品ですから良くないと思えば捨てれば良いんで（笑い），今在る物で何を選ぶかということだと思うんです．

それから，僕の質問なんですが，私もMac を使っている時もあるんですが，皆さんマルチウィンドゥ開いてマウスで何か仕事をするのが使いやすい，という話しが出てきたんですが私はそうは思わないんですね．Mac のスクリーンの情報量は非常に小さいものですから，あそこでウィンドゥを開いて仕事をするというのは，机の書類の山から隅を見つけてメモをとるという世界に近いんですね．もちろんマックには他の，サイドキックという類の，画面を全部入換えてしまう方式のウィンドウもあるわけです．ユーザが使う場合には提供できるユーザ・インタフェースの環境は，制約がある程度ありますので，その中から何を選ぶのかと云うのはかなり重要なファクターだと思います．

中野： ＧＯＴＯ１００というのを悲しくて大阪大学で教えています（笑い）今，卒研のテーマで１人ウィンドゥの話しでやっていますので，もしよろしければ皆さんに意見を出して頂ければと思います．

先ず，ウィンドゥが見にくい．これについては，お節介なウィンドゥ・マネージャというのを考えまして，ウィンドゥの最適配置が行えないかということです．最適配置というのは言いすぎで，ルール，僕はエキスパート嫌いなんですけど，各人のルールに合せて触っているうちにウィンドゥが動いてくれれば良い．それから，ちらかった書類については電子秘書というような，プロセスを後ろえ動かせば良いんじゃないかと思っています．もう１つのアイディアは視点を変えると自動的に関連するウィンドゥがお節介マネージャーとか電子秘書によって，ススと連動するようになる物は無いかなということです．ハイパーテキストなんかがそうかなとも思っていますが，教えて下さい．

落水： 感想を言えば慾りすぎではないかと思います．先程吉村さんがおっしゃった，状況とか手順によって情報にアクセスするというのは非常に面白いと思ったんですけれども，ススと変るのは作るのに苦労する割には，あまり誰も使わないんじゃないかなと思います．何を言いたいかと言うと，あまり良くないと言ってるんですが（笑い）．

岸田： 今日配ったアメリカの環境ワークショップのレポートの３４ページの所にユーザ・インタフェースというところがあって，ここにアメリカの人達が２年位前に考えたユーザ・インタフェースの問題点，これは多分臼井さんが出された質問と今の中野先生のお話しと両方関連すると思うんですが．例えばマウスだとかビットマップだとかウィンドゥだとか，それは単なる仕掛と言うか，ユーザ・インタフェースを実現する為のメディアに過ぎない訳で，ユーザ・インタフェースを考える上で１番大事なのは，ここに書いてあるように，システムがユーザの行動をどういうモデルとして捕えているか，という話しとユーザがワークステーションを使って仕事する訳ですが，その仕事のプロセスをどうモデル化できるかという話し．それと，そうやって仕事をして出来上がってくるドキュメントとかソースとかテストデータとかのプロダクト全体のモデルみたいなもの，そういうもののモデルをきちんとしないと良いユーザ・インタフェースにならんじゃないか．ただ問題は，モデルというのはある種のスタンダードで，ユーザの行動はかくあるべしだとか，プロセスというのはこういうふうにしなきゃいかん，とかの標準ということになって来て，ユーザに対する強制力が逆に働いてくる．

多分今 Smalltalk とかMac が良いのは，そういうモデルが無いユーザインタフェースだから，ユーザが勝手に自分のモデル，頭の中にあるモデルに合せてユーザ・インタフェースの仕掛を使えるという辺りが，ごく一部の人達に非常に好まれている．多分普通のユーザにとってはシステムの中に，ある種の自分の行動についてのモデルとか，自分が仕事をする時のモデルをユーザが自分で設定する，或いはシステムの中に標準があってそれをユーザが自分に合せてカスタマイズするとかの仕掛がないといかんのじゃないかと思っています．

今の中野先生のお話しは，それの１つの具体例じゃないかなと．それでさっき臼井さんがおっしゃった，この世の混乱をそのままモデル化せずに押し込むんではなくて，少しは整理して持ちこんだらユーザ・インタフェースは良くなるんじゃないかなと，私は思います．

佐原： 事実関係を多分間違えていたと思うので・・．私とか青木さんが言っているマウスとかマルチウィンドゥと云うのは，そのソフトは，そのモデルをベースとしたマウスとかマルチウィンドゥです．ひょっとして一部の，例えば昔のＰＣ１００ユーザとか，ああいう方は物理的なマルチウィンドゥとかで使いやすくなると思っていらっしゃる方がいるような気がします．それはあくまでもそのソフトを，モデルがベースにあったマルチウィンドゥなりマウスを使ったシステムと，そうではないものとは区別しないといけない筈です．

佐原： 更にですね，視線の動きを追うというのは誰かやっている，基礎的な学問をやっている人がいまして，視線というのはちゃんとあるとこを見ていないらしいんですね．ウィンドゥを見ていて遠くの女性を見ているとか，どうもその線はダメそうです．

加藤： ユーザ・インタフェースを考える時に，提供しているシステムのモデルの話しはもちろん重要で，今まで議論されていたところですが，問題はユーザのモデルなんですよね，多分．その研究というのは多分あめりされていなくて，もっとプリミティブな，例えばある入力を受け付けるのにメニューを出してマウスでたたかせるべきか，キーボードから選らばさせるべきかというようなツールを作る時のディシジョンがあって，やっぱりマウスの方が良いだろうとマウスにしてしまう．というのが今の現状じゃないかと思って，それは非常に安易な方法ですけど，それなりにきっちりやっている研究が幾つかあるらしくて，ユーザがキーを打つのにどれ位時間がかかるとか，キーボードから手を離してマウスへもって行くのにどれ位時間がかかるか，マウスをつかんでポインティングするのにどれ位時間がかかるか，という定量的な話しになってくるとユーザ・インタフェースの評価の方法も変ってくる．今のところそういう定量的な評価が無いので宗教的な話し，つまり行番号があってはいけないと言う人と，欲しいと言う宗教的な話しになる訳ですね（笑い）．ですから，行番号があってはいけないと言う人は，何故あってはいけないかを具体的に説明すべきであり，それが出来ないなら両方用意して行番号でもお使い下さい，と言うのがツールを提供する立場としては正しい．ですから，吉村さんはそれ程いじめられる必要はないんではないか（笑い），と云うのが僕の意見です．

佐原： そういうエディタについては結構調べられていて，例えばエラーの発生率などは Xerox とかでやったもので，行番号付のコマンドを主にするようなエディタは非常に悪いと云うのが，ある程度定量的に出ています．その数字をどこまで信用するかは別として，大体５倍位の開きが１番良いのと悪いのとではあるという．

加藤： プログラミングの作り方にもよりますが，１つのプロシジャーが長いものを作る人は，行番号があるとプロシジャーのどの辺にいるかというような話しは，行番号をうまく使うと，ＲＥＮＵＭなんかする前の状況（笑い），表示できる訳ですね．だからフラットテキストの場合なんか良くないんだろうけど，うまく使うとそれなりに使える．使いたい人というのは，そういうところが多分嬉しいから，使いたい人がいれば，やはり必要だと僕は思います．その人は多少効率が悪いのを容認しつつ，やっぱり良いもんねと言ってる訳ですから．

熊谷： マンマシン・インタフェースの討論を聞いていたんですが，感じたことは，１つは今の計算機をビジュアルに使おうという考えですね．人間と人間は意味を伝える時に，あらゆるメディアを使って伝えていますが，それと同じようにマシンを使った時も，ただ単に計算機を使うのではなく，その中に何を入れてあるかによってマンマシン・インタフェースを考えると，切り口が変ってくる．そうするとマウスとかマルチウィンドゥは，マンマシン・インタフェースを論じる時のメインテーマでは無くなってくる筈なんです．

もう１つは単純な質問なんですが，リレーションをどうプログラミングすれば良いか，工夫があったら聞かせて下さい．

青木： 要は関係性だけをマニュピレイトできるものを，オブジェクトの対象物の世界の上に持つんです．そこからマニュピレイトしているという形でリレーションをインクリメントしています．オブジェクト指向でリレーションを作ると，結局オブジェクトになっちゃうんですけれども，私自身としてはリレーションはオブジェクトではないと思っていますので，そういうインクリメントをしていきたいと思います．

落水： どういうリレーションシップを中に埋め込まれているか，人間に任せる問題は何で，ユーザ・インタフェースで処理する問題は何で，どういうタイプのものがユーザ・インタフェースの改善に貢献するのか，というお話しをお伺いしたい．

それから吉村さんには，ユーザ・フレンドリというのは判かるんですが，個別対応していたらきりがない訳でして，何かジェネリックなものを持っていてカスタマイズしていかないとならない訳で，メカニズムの話しに入る前にポリシーがあると思うんで，大学に居ては判からない訳で，実感されていることをお聞かせ下さい．

青木： 本質的にはオブジェクトが無くとも関係性だけはこの世の中に存在するだろう．関係から全てのものを起こせるんじゃないかというものを，自分の根底に持っています．実際にものを扱う時には，対象世界へ落してオブジェクト化して皆に見せる．ですから，先程お見せ

した Unix もその１つのアプリケーションに過ぎなく，本質的には関係とかオブジェクトの方向へ，自分ではどんどん行きたいなと思っています．それをどうやって作るかは，今模索中です．

吉村： 本当はユーザ１人１人の好みに合ったものを作りたいんですが，現状としてはジェネリックなものを幾つか提供して，それをカスタマイズするアプローチしかできません．その中でもユーザが使っていく中で，ユーザの癖に応じてユーザ・インタフェース自体が変化していく，というものを提供したいと考えています．

新田： 私のＥＲＡＳの方針は青木さんと逆でして，What you seas What it is つまり何か皮をかけるのでなく，ユーザが見ているものがそれ自身だということです．変なモデルを持ち込むこと（笑い），ユーザに何かを隠す，本当にあるものじゃないものを見せているような隠し方は，いけないんじゃないかと思っています．

藤野： 多分新田さんの言っている，あるものを It is のままで見せるというのは，それも１つの新田さんのモデルの訳で，ユーザにイルーションを見せるというのがSmalltalkのやり方で，これもまた１つのモデルなんです．そこでまたもう１つ，先程落水先生が言ったように，ジェネリックなモデルというのが考えられる，というのもまた１つメタなモデルだと思うんです．後はどこからスタートするのということだけで，基本的にはユーザが自分でカスタマイズ出来るジェネリックなものが必要でしょうね．だから，１番最初にどこから始めるか，というのが現在よく判かっていないのであって，そこまで考えるとコンピュータの世界へ現実の世界を持っていくと言いましたが，実は現実の世界では出来ないことも出来る訳で，そこまで超えて踏みこんで行かないといけないなと思っている訳なんです．

中村： 元々僕自身今の仕事に入る前は脳の研究をやっていたんですが，我々人間は意識せずに，例えば歩行するとか，物を掴むというのを意識せずにやっています．そういうのは小脳の部分に内在したモデルを持っているんです．そのへんが下位の層として，モデルとの間でフィードバックをやりながら動いていくんです．で，何が言いいたいかと言うと，確かにマンマシン・インタフェースでユーザにこびると云うのが良いと言われているんですが，でも本当言うと人間の方に，これが本当の姿ですよと押し付けても，本当にそれが良ければ，それを使い慣れると使いやすい．言わば今いろいろと論じられている問題で大切なのは，いったいどう云うモデルが１番必要であり，それをいかに作っていくかというのがポイントであり，その間をいかにユーザを移行させていくか，というのが必要になるんではないかと思うのです．先程の話しで，キーボードを打てるのに何秒かかるかと云うのがあったんですが，それよりは人間の内在するモデルというのをいかに構築できるか，というのが本当は１番，ユーザと機械とのインタフェースの上で必要なことじゃないかなと思います．得てしてマンマシン・インタフェースを豊にすると云う時に，そのへんのモデルがあやふやなものとなって，自分自身も答を出せずにいる訳で，本当に人間と機械との間の最適なモデルはいったいなんであるかというふうに思うんですけれど．

堀川： ユーザ・インタフェースがユーザ毎にどんどん変るというのは，やり方としては良くないのではないかという気がしています．１点目の理由として，ユーザというのはスイッチ入れたら自分の好きな環境で動くようになって欲しい訳で，その為にわざわざ定義を変えたりするような苦労はしたくない．もう１つは，ユーザ・インタフェースがどんどん変えられる機構を盛り込んで，それでもの作りを始めた時に，個々のユーザに対してそのジェネリックな機構を使って個々のユーザ・インタフェースを提供した時に，とてもお守りがし切れないと思うんです．そこで重要なのは，内在している本質的なモデルというものを，間違っているかもしれないけど作って，作っていくのが１つの方法ではないかなと考えています．ユーザの目的とそのモデル化というのがあって，それでユーザ・インタフェースというのを考えるべきであって，それは決して個々のユーザがこれを要求するからそこに流されて行こう，というふうな進め方でもないと思うんです．もう１つ，ユーザ・インタフェースのモデル化という話しをした時に例えば，ジェネリックのユーザ・インタフェースのモデルだと言われると，モデルが何なのか判からなくなってくるんですね．

例えば，我々のところでツールを使う時どうやってユーザ・インタフェースを決めるかと言うと，ある計算機があって，そこで使えるユーザ・インタフェースを一通り使ってみる訳なんです．つまり，ユーザ・インタフェースというのは，その計算機があってインタフェースがあるんですから，実際の計算機に依存されて変って行かなければいけないんではないかなと思ってるんです．

そういう話しをするとユーザ・インタフェースという

のは，これが基本になるというところが無い領域じゃないかと，頼りになるのは人間だけではないかと，人間というのはユーザとしての人間，だけではないかという気がしているんです．

林： 後半の部分は正にその通りだと思いまして，ユーザ・インタフェースという位ですから，ユーザしか頼るところは無いのは当然だと思います．人間を基本的にモデル化して，そのモデルに従って作れば良いというんですけど，人間は人間をモデル化出来るのだろうか．というのが最大の疑問でして，出来ないからメタ化するんですね．今のところ最善の方法は，ありとあらゆる我侭を聞けますよ，というのがユーザ・インタフェースとして求めて行く道にしか無いと思っているんでしょう．それで良いとは言いませんが，それしか無いんじゃないかと思うんです．例えばそのことが顕著に出ているのが，Xウィンドゥなんかもそうですね．何でもかんでも変えられるようにしてあって，それをなるべく簡単に変えられるようにしてあるから，皆が文句も言わずに使うんですね．大体ユーザ・インタフェースというのはフィーリングとか感性ですから，そのフィーリングみたいなものがベースになっている以上，そんな感性は沢山あった方が良いですから．人間の感性が１個になったら嫌ですから（笑い），やっぱり自由に変えられるようになっているしかないんじゃなかろうかと思います．

それで熊谷さんの発言にコメントをしたいんですが，つまり Unix の上に Smalltalk の皮を被せたのでは，Unix でできることしか表現出来ない，と言う発言がありました．２つが一緒になるとこれは，今 Unix の上にウィンドゥ・システムを作っている人達が何人かいて，大変な時間をかけてやっている訳です．金かけて．ところが４～５日でほとんど同じ効果のあることが出来てしまう，スゴイんじゃないかと思いましたけど，僕は．しかも Smalltalk の良さまで入っている． Unix の上にウィンドゥ・システム作るんだと，つまらないんですよ （笑い），すみませんつまんなくないんだけど，それよりスゴイんじゃないか・・（大笑い）と僕は思えるんです，青木さんのやった事は．

熊谷： 誤解を受けるとまずいんで，そんなに否定している訳じゃないんです．何を隠そそう，実はうちでもやっていますから（大笑い）．ただ Smalltalk を介した時の Unix のプロセスなんかやると云うのは，ある意味でつまんないんですよね．何故つまんないかと言うと，Smalltalk 自体の制約が結構あって，例えばマルチ・ウィンドゥで動いたとしても，それはXウィンドゥに遥かに及ばないのは想像すればすぐ判かることでしょう．前面的に否定しているのではなくて，いろんな事をやらなくちゃいけないというのは良く判かっているんです．私が言いたかったのは最初の方に，インタフェースだけじゃなくて何をどう表現するかというマンマシン・インタフェースの方が重要であって，という観点から言ったのでそう言ったのです．コンテックスを見ないとダメですので，コンテックスを是非見るようにして下さい（笑い）．

岡本： 私自身は Unix が好きで，１番フィットするマンマシン・インタフェースのような気がするんです．ところが，恐らく日頃 Jstar ａｒだとか 1161 だとかを使われたきたファンの方は，そういう方が例えばニューミュージックをやろうとすれば， Jstar に近いとか 1161 に近いマンマシン・インタフェースを持ち，しかも同じ事が出来るというのが１番フィットすると思うんです．これは１５年位前にジャズ・ロックとかロック・ジャズという話しがあって，ところが何年か経つとフュージョンという言い方になってきて，もっともっと分化した独自の世界を作り上げてきています．そういった意味で，これからはいろんな世界をうまく組み合せて，どんどんいろいろやって行くことが重要ではないかなというふうに思っています．

坂下： ありがとうございました．なかなか今のは良いまとめだったと思います（笑い）．私が言おうと思っていたことを言われてしまいました．確かに，ユーザ・インタフェースについて皆さん色々な意見を持っておられて，その多様性というのを意識しました．今までのユーザ・インタフェースというのはモデルというか，多様性があるようで無いようなシステムを，どっちゃりとぶちまけて来たような感じがします．なかなか楽しいセッションだったと思います．前の３人の方に拍手で終りたいと思います，どうもありがとうございました．（拍手）

山浦恒央

吉村さんのセッションは、ユーザインターフェイスの本質をつくもので、現場とプログラマのギャップの例が面白く、興味深いものであった．しかし、雑感で述べられたことは Software Engineering の教科書に書いてあることであり、それなら、こういう方法はどうだろうというものがなく、その点に不満がある．何か叩き台的なものを propose していただければと思う．

新田さんの方法、即ち「方法論とツールを分割し、使用者が方法論に適合した環境を構築する」という考え方は、私自身の考えと全く異なるもので、非常に興味深く、新鮮であった．ただ、マウスの機能を増大させて、オペレーションを行わせるというのは反対である．マウスが大きくなると、ついには、キーボードと同じ大きさになってしまうのではないか？

杉田義明

・FXIS の青木さんの発表は、ssmalltalk ハッカーたちが通常陥る閉じた世界をそのまま再現したように感じた．non smalltalk people に対して、もうすこし、smalltalk 専門用語を省略して分かりやすく話していただきたかった．しかし、内容の smalltalk と Unix をつなごうとする、ブリッジ・ツールの狙いは、着目がユニークであり、成果が注目される．

・吉村さんの内容は、きわめて現場的であった．このような個々の改善を積み上げていけば、実践的な環境の整備が進むと思われる．

・ERAS のプレゼンテーションは、面白かったが、開発の狙いは、究極のインターフェイスを目ざすとのことであるが、よく伝わらなかったようである．

佐藤千明

・ツールと方法論の分離は、いいのでしょうか．ある方法論を支持するツールが必要と思います．

・smalltalk on UNIX の操作性 MS-DOS のエコロジー、J-Star(MESA) でも似たようなことができる面があります．これらのツールは、何気なく当たり前に使っていますが．

・Visual な I/F は、必須でしょうね．

佐原伸

(1) 自分にない考え方、発想を聞けて面白かった．

(2) 行番号付のエディタが、Mac でも生き残っているとは！！

田中慎一郎

ちょろちょろしていたので、よくわからない．

青木さん: SUMMIT の命名に感心

吉村さん: オーバーラップの汚い重なりは私も嫌いタイリングがいいのではと思う今日このごろ

新田さん: マウスの利用法の辺りがよかったあの辺が本当は大事ではと思う

歌代和正

吉村さんの発表は、現場をこの間見てきたので、面白く聞けました．ＨＳＴは、若い人が多いので、ユニークな考えがいっぱいあって面白いです．

ＥＲＡは、ユーザインターフェイスのことは、良く分かりませんでした．内容は、面白いけれど、何に使うのかはまだ良く分からない．

加藤朗

"･･･が良い／悪い"は、全てのことについて(対象,領域,分野 etc.) について言えるのかしら．でないとしたら、･･･の時の話について、という前提が明確になっていると良かった．吉村さんの"ユーザごとに異なる"わけですから．「xx で oo というのがありますが」すみません．oo を知らないもので．10 秒程度でも説明してほしかった．

吉村鉄太郎

個々のユーザの好みや要求に応じて、ＭＭＩを tune-up することが、もっと行われると良いと思う．

村川貴広

ls　cd　edit という流れを可視的にしたユーザインターフェイスや ERAS のように、エンティティーをグラフィカルに表し、そのリレーションも同様にして表されるユーザインターフェイスは、なかなか面白いものだが．

やはり、ビットマップディスプレイ＋マウスの域は、でないのであろうか？

実践的という意味ではいいと思いますが･･･．

これから、我々でもこうしてビジュアル化をしていきたいと考えました．

三浦あさ子

"ユーザの好みに合わせて、違ったユーザインターフェイスが提供できる"という考え方はすごいと思った.
動作の「対象」と「動作」「動作の仕方」でできる
これは、むずかしい.

熊谷章

青木：前半は良い. 後半はターゲットが UNIX でない方が良い. したがって、前半から後半の連続性が少し、希薄になった気がする. リレーションをオブジェクト表現の時にどう表現するのか?
吉村：ユーザインターフェイスとして、何を作ったのですか. 話では、os と開発者との IF のようにみえた. ロボットとユーザインターフェイスの周辺で問題はないのか?
新田：内容が面白い. ERAS の用途はどこら辺か?

吉村智香子

人間の動作そのものをやってくれるユーザインターフェイス. 人間ができることをすべてコンピュータの中に活かしたいと思います.

臼井義美

·オブジェクト間の関連を的確に表現できるようなデーモンもマルチウインドによる書類の山も現実の世界の混乱をコンピュータの世界に持ち込むことになる.
新たなパラダイムが必要では?
·吉村さんの話で、マルチウインドを管理するウインドを作ることは、メニューに戻ることと同じ状態にならないか?
·現実世界の「物」の図を画面にとりこむことが、本当に最も良い環境なのだろうか?かなりいいことは事実だけれど….

岸田孝一

smalltalk や Mac など、いま世の中で「良い」といわれているユーザインターフェイスについて、「良い点」「悪い点」を具体例を挙げて、討論してほしい.
(新田君があげた Disk & Formatting の例のように)

井川裕基

(1) UNIX コマンドの作り直し(?) は、必要だと思います. どのような形でやられるのでしょう.
(3) 動きが分からないので、デモに期待します.

小林貞幸

· smalltalk を使ってみたい.
· Mac を使ってみたい.
·汎用機の「は」の字、コボルの「コ」の字も触れられていない. ショック!
であるにもかかわらず、共通の概念として理解できる. 不思議.

林香

青木：英語にすれば、USENIX ものの発表であった.
吉村：マルチウインドウシステムの限界点の指摘は良い. 机の上の混乱をディスプレイ上に移動しただけ.
u/i は、「手順と状況」という考え方は、おもしろい. 人間が覚えやすい.
新田：u/i (現在の最新の) を非常にうまくまとめている. 今や u/i を論じるとき、Mac や Smalltalk が中心となり、< キャラクタ端末が死んでしまった感があり > 満足.
u/i のメタ化の考えはすごい.

中野秀男

(1) 思っていること：マルチウインドウ上の各ウインドウの最適配置 (もちろん各人の好みに合わせて) 後ろでプロセスが走って、勝手にかえる.
対象としているもの (文章の一つの単語など) をクリックすると、関連する他のウインドウの内容も自動的に変わるしかけ. (でも、Hyper Card のスイッチのしかけや、青木さんのいうようなデータにメソッドを埋め込めばいいみたい.)
(2) 感じたこと：いろいろ優れたＭＭＩができてきて、ユーザが選択できるようになるのか、標準化するのか.

寺井孝

三氏の発言をお聞きすると、共通点としては、ユーザ個々の様々な要求に答えるには、マシンをより人間に近づける事を目差しているように感じました.

井上尚司

コンピュータが人間の行動を制限しないように実現されるのは、まだまだ先の事だと思っています. マンマシンイ

ンターフェイスの最終目標が達成されるのは、いつの日になるのだろう？

西岡健自

最初の発表が面白い．絵にして動かすときの一般的戦略があれば聞きたい．

盛田政敏

ユニークな話でたいへん面白い．

鐘友良

今日の発表について、大部は、マルチウインドウを用いて、ＭＭＩとつけることですけれど、ＭＭＩについて、別に誰か何かいい方法か、いい考えがあれば知りたい．

藤野晃延

(1) 青木: もう一つ上のレベル、つまり、Meta レベルへの普遍性を求めるのが、今後の問題でしょう．
(2) HST: 面白い．Mac はたしかに UI に革命を起こしうる Potential を持っているし、進むべき方法としては、good．
(3)新田: Meta-Definition Self-Containd-generic な指向で、すばらしいと思う．ただ、User-Define-Rule の sematic な正しさの check 等は問題（課題）だと思う．

小林明彦

エンドユーザサイドよりのＭＭＩの発表がなかったのが残念．私の興味は、エンドユーザへの提供をどのようにしたら良いかにありますので．

青木淳

平田工機さんの方：
Mac で頑張ってほしい！
新田さん：
おもしろい．美を求めるために、自分で自分を記述するのは良い事です．

御喜家

マウスのボタン操作の早さ（速い、遅い）で、ユーザの気持ちが伝わるというのは、とても興味がある．おもしろいなぁ．

海尻

·やはり日本語
·遅れていますが、通常の CRT 上のユーザインタフェイスは．

堀川博史

青木氏：参考になった．実行速度はどんなものかなあ．
吉村さん：実践的な話で、こういう世界もあるのか（こういう世界ばかりかもしれないが）と思った．
新田氏：斬新であり、面白かった．
発表者にバラエティがあり、よかったです．

市川寛

ＭＭＩは、よく分かるが、マシン-マシン-インターフェイスは、どうなっているのか．もっと、システム全体（開発環境全体）を考えてくれるグループはないのだろうか．

小池幸徳

ユーザレベルに合わせたＭＭＩが、重要となってくると思われる．
例えば、キーボード、マウス、マルチウインドウ、タッチセンサーの選択
このようなＭＭＩの選択をどのような方法で行えばいいか分からない．

田中一夫

青木氏：わからん．吉村さん：言っているとおりだと思う．新田氏：面白かった．

中村眞

エディタって何？何だか仕事とはただ文具の鉛筆で、必要だからねと言っているのと同じだよね！

近藤康二

(1) smalltalk-80 smalltalk は、前々から興味があったので、とてもよかった．内容は、全く同感であり、UNIX も時代に合わせて変わらないと取り残されてしまうという感を強くした．
(2) ロボットプログラム現実的な話（特に質問コーナーでは）苦労がよく感じられた．私も会社に戻るとそうだなあ．
(3) ERAS とても理論的な話のように聞こえた．自分で自分を作るというパラダイムは、なかなかだ．明日のデモ

がたのしみである．

久保宏志

「Algorithm を value にもてる変数概念がＭＭＩ言語にとって、本質的である」との意見は、傾聴に値する．

酒匂寛

(1) smalltalk を UI 言語とする考え方は、正しいアプローチだと思います．よく言われる言語の閉鎖性を打ち破る鍵の一つがあるように思われます．

(2) "人間が実際に使う"という視点にたったインターフェイスを忘れてはいけないと思います．

(3) ERAS の将来に期待しています．ERA モデリングの極限を見てみたいと思います．

新田

(1) 私は、コマンドインタプリタで仕事をしているが、ダイレクトに･･･をやっているという感じがある．ダイレクト感は、人によって違うのでは？

(2) マイコンで全てやっていると、例えば、FIle の re-direction や、アーギュメントのやりとりの記述がむずかしい．

(3) またヘンなことを言ってしまったかな．

荻生準一

私自身の業務は、電力供給システムの要求定義、システム設計が主であるが、この session で発表された開発環境におけるＭＭＩの考え方が、客先とのＭＭＩ機能仕様決定に参考になり、興味深かった．

現状、電力供給システムでは、Malti Window 等のＭＭＩですら、取り入れられず、運用している次第である．次システムの機能仕様決定には、この session の発表を参考に取り入れてみたい．

落水浩一郎

(1) 勉強になった．(smalltalk がないので、やはり、欲しいと思った)

(2) 頑張って下さい．(もう少し、話に(技術的内容に)具体性がほしい)

(3) 移動時間等のあいまいなものを(人によって違う)基準にしてよいのか．少し、疑問に思う．

岡本隆一

三人のプレゼンテータの方々から人間の五感のいくつかに触れるユニークな発表があって、楽しく聞かせていただきました．

が、耳にウッタエル話はなかったようですね．

今の私の職場では、この「耳」にとりつくアプローチをやっている若者が居まして、Desk が隣接している私としては、毎日非常に迷惑しています．

こんな話題を参加者の皆様から聞けるといいですね．

桜井麻里

・青木さんの発表は、整理されて分りやすかった．

・人間ができることをコンピュータによって制限されたくないという、新田さんの発表の発想は、とても自然なものだが、それが夢でなく現実的なものになりつつあるのだなあ，凄いなあと思いました．

・吉村さんの発表も現場の苦労が分かって、面白かった．

北野義明

Mac が、実務の場で、こんなに使われているとは思わなかった．

やはり、Cや ST-80 で、アプリケーションを作っているところと、CoBoL 等でホスト上に Keyboard 入力してプログラミングしているところとの差を痛感した．

ＭＭＩの優れたシステムを有するマシン、OS 上で、CoBoL コンパイラが欲しい！

欲しければ自ら作れ！といわれそうだが．

森幸一

・icon に bind される program の種類は、マウスクリック操作に限定されるのか？そうなら、ユーザインターフェイス上かなりの制限になるのではないか？

・ユーザインターフェイス成長の考え方には賛同．

伊野誠

(1) ＭＭＩはむずかしいが、いろいろな考え方、アプローチは面白かった．

(2) もっと基本的な問題があると思う．すなわち、基本的な作法とか、作り方の指針が世の中に何もないということが問題．

kato@cs.titech.junet

青木氏：
Visual shell との差．"本当に" ユーザインターフェースは、改善されているのか．
コマンドの argument の互い file 間の dependencies：i.e. view-/etc/passwd
吉村女史：雑感(?)は、(性的ではあるが) するどい指摘である．学習するユーザインターフェースの実現には？program の実行部とユーザインターフェースの分離が必要？行番号でいじめられましたが、やはり、あれは便利．それほどヒドイものではないと思う．
新田氏：時間軸の導入は鋭い．が、慣れていないとうまくいかない･･･かな．ユーザインターフェースは、simple に、作業内容は複雑に、という正反対の要求のトレードオフは？

田中正則

ＭＭＩに関しては，そのシステムを使う人（ユーザ）が，作成するのが望ましい．このセッションをきいて，その方策として，ジ*ネリックなＭＭＩを提供し，それをユーザにカスタマイズさせれば良いと思った．この時，ジェネリックなＭＭＩの問題はおいておいたとして，残るのは，ユーザの教育の問題，その気（カスタマイズさせる気）にさせるにはどうしたら良いのかも教えていただきたい．

藤野晃延

ＭＭＩの model を generic とすると以下のような問題が起きる．(1)custamize 容易性と，一貫性の trade off
(2)custamizu するというのは，そのユーザにその仕事を「意識」し直すように強制する．
(3)generic した後の sematics に及ぼす変化，その正当性をどう誰が保償するのか．といった問題が新たに派生するが，これを meta-model として，定義することは，試されるべきだろう．

佐藤千明

ＭＭＩ　は，　Model.Man.Interface　又　は，Method.Man.Interface と呼びましょう．

吉村智香子

ビットマップディスプレイとマウスがあれば，良いユーザーインターフェースが提供されたというような風潮があるように思います．”ユーザーの好みにあった”インターフェースを提供すればよいのか．”人間の内在するモデル”を作っていけば良いのかで，盛り上がったようだが，結局は，今あるものを積み上げて，ユーザーインターフェースは進化するということのように思う．

田中慎一郎

ＴＲＯＮのキーボードの話：あれがいいのかどうかは知らないが，あいうえお順はどうかと思う．本当に普及させるつもりであれば，最初覚えるのが大変でも人間工学的に使いやすい配置になっている方が良いのも確かである．
誰にも選ばれないツールは，消えていくというだけの考え方は，危険．人間は，食わず嫌いが多いので，良いと思われるものは，それを使わせる努力なり方法を考えることも重要．ユーザに流されるのは良くないが，ユーザに任せる部分がある方が良いと思う．

桜井麻里

材料は豊富に，組み合わせ選択はいろいろできるように，といってもモデルは，必要である．使いやすいインターフェイスの基本形も必要だ．どんなにカスタマイズできたって，自分では何もカスタマイズしない人がほとんどだから．それにしても，みんな同じことを悩んでいますね．私も悩んでいます．

三浦あさ子

中村さん，佐原さんが話した２種類のユーザインターフェース:
ユーザインターフェースといってもあまりに多様で，話についていくのはけっこう大変だ．私も今，Ｘwindow 上で，ユーザインターフェースを作っていて，カスタマイズというのは，一つの focus であると感じている．context が重要だ，というのは分かるけれど，それがうまく結びつかない．

小林明彦

聞いているには大変おもしろい議論が聞けたと思いました．私もユーザインターフェースは，好みが強いと思っていました．結局，声の大きい奴が力説したものがその組織でのユーザインターフェースとして，取り入れられるようです．

西岡健自

ＭＭＩは，独立に存在しうるのか．目的は何なのか．試行錯誤の連続とモデル不在の焦りを感じる．けれども，とてもおもしろそうだ．

森幸一

ユーザインターフェースを考えるとき，人とコンピュータとのコミュニケーション（＝メッセージの交換）の視点が必要と思う．インターフェースは，メディアとしてのビットマップディスプレイ%マウスやメッセージの手段としてのグラフィックだけでは不充分である．特に重要なのは，コードとコンテクトへの考慮が今後必要と感じる．

北野義明

キャラクター端末＋キーボード:
Edit した JCL を Home キーを押して（コマンド行へカーソルが移る）”SUB”（Submit=実行）と入力して

RETURN をたたくのは，慣れた人なら，よそ見してでもできる．
ポップアップ（プルダウン）メニュー＋ビットマップ＋マウス：
メニューの項目を良く選び損ねる．（直ぐ上や，直ぐ下の項目を選んでしまうことがある）．よくメニューの項目を見ていないとダメ．手先の細かな動きが苦手だと Edit の際には，よく一文字ずれた所をポインティングしてしまう．
TSS ダム端，I-Star，J-Star 中でのエミュレータを使って思うこと．
人や，何をするのかや，どの程度･･･からＭＭＩの要求を探すとキリがない．好きなものを各々が選ぶのが，一番のような気がします．

伊野誠
大変興味深い討論でした．

古田とおる
Mac，Smalltalk など，マウスとウインドウによるＭＭＩだけでなく，キーボード，タッチペン，ジョイスティク，又，新しいポインティングデバイスが登場したときのＭＭＩも考えたいと思います．

井上尚司
コンピュータを使うだけのユーザ，ソフトを作るために使うユーザ，を分けて考えて，ＭＭＩを考える必要が少なくとも現在ではある．
「現在のマウス，ビットマップディスプレイを考えてソフトを作る」という思想を悪いとは思わない．これに対して，「マウスでがまんできるか？」という質問は，的を得ていない．しかし，今すぐにソフトを作れなくてもよいから，現在考えうる理想のハード環境を考えて見るのも良いのでは？

寺井孝
今回の討論に参加された方々は，皆それぞれ開発する立場から発言されているように思いますが．ユーザの感じ方，考え方を自分なりに勝手に判断し，思い込みも入っているように感じました．さらに，ユーザに使い勝手の良いものを開発するためにも開発作業の他にユーザの考え方を積極的に聞くようにしていただければと思います．

小池幸徳
ＭＭＩは，ユーザのレベルに対して考えるものとその目的に対して考えるものがあると思う．ユーザのレベルに対して考えた場合は，開発者，エンドユーザが同じＭＭＩをもつのは，最適ではないと思う．また，目的に対して考えてみると，ポインティングデバイスとしてドット（ピクセル）レベルのポインティングを必要とするかどうかによってマウス，ライトペン，タッチセンサーなどいろいろあると思われる．しかし，これは，人間の生活に置き換えて見るとなんとなくかたわのような感じで，もっと音声等の違ったデバイスに対しても考慮しなくてはならないのではないかと思います．

海尻賢二
User interface とその body となる soft との関係は？

鐘友良
ＭＭＩの中心道は，熊谷さんのおっしゃったとおりで，人間が何を入りたい，どういうふうにこの”入りたい”ものを簡単に表現するか．逆に，マシンが何を出るか．どういうふうに，人間がよく理解できるように表現するか．目には，人間とマシン間の”対話”を人間と人間の”対話”のようにするべきであるか．

盛田政敏
「使う人によって変わる，又，使う人の成長によって変わるインターフェース」は大切である．どうすれば実現できるのか？その為のフレームワーク．

中村眞
ツールの性格付けは必要と思う．

村川貴広
討論の内容が，発表者に対する質問だけで，もう少し，参加者に対する質問があってもいいのではないでしょうか？そんな環境づくりを我々がしなければならないのでしょうけれど！！

近藤康二
・マウスについて
私は，マウス，タブレット etc. これも user 合わせられるものである．

·現実とのギャップ議論はおもしろかった.
·現実には，user 上／下について，
user が変えられるようなものでなければいけないという考えは一致している.
では，具体的に･･･いくつかをプロトタイプしてみる必要を感じる．その意味での青木さんの成果は，すばらしいと思います.

久保宏志
ユーザインターフェース＝ＭＭＩの概念を構造的に理解することが行われて良い．考察の対象の全体モデルについてのコンセンサスを持つ必要がある．といってもよいのかもしれない.
·「レスポンスタイム」は，ＭＭＩの「特性」の一部である.「User friendliness」も特性の一部である.
·マウス，電子ペン etc.は，人がマシンに語りかける為のＭＭＩ言語仕様の一部である.
·マルチウィンドーは，人が知的生産を行うときにつかう仕事場というＭＭＩの一つである.
· 個人の好みにあった MMI というのは，ＭＭＩの implementation に関係する.
· smalltalk の「model ＋ view ＋ controller」＋「Dependent」は，対照世界を人が対象化するためのフレームワークという意味のＭＭＩである.
· smalltalk が，フロッピーのイニシャライズをやってくれないというＭＭＩ上の欠陥は，ＭＭＩのインプリメンテーションに於いて information hiding にかかわる.
·ユーザインターフェースがツールとツールで，整合的になっているかどうか，ＭＭＩの＂特性」の一部である.
以上，全体に包括されるであろう断片をおもいつくままに書いた.

深瀬弘恭
Generic なインターフェースは，一般的ユーザにとっては逆に使いづらいのではないか？ Adaptable な部分と骨格となる部分の切り分け作業に時間を費やすのが良いのでは？

近藤隆彦
ＭＭＩの多様性，困難性が良く分かりました．また，各々の技術者が，第一線で様々な工夫をしておられることに感銘いたしました.

落水浩一郎
おもしろかった.
UI の議論の範囲
操作
らくに操作できる
relation management
視点
object management 関心のあるもの

市川寛
·もう少し，討論を続けたかった.
·ＭＭＩの最新のツールを試用してみたい.

田中一夫
ＭＭＩは，どうやって作るか，非常にむずかしい．でもなんとなく分かったような気がする.

堀川博史
·リレーションの話とモデルの話が概念的で，十分理解できなかった.
·その他は，非常に有意義だった.

佐原伸
(1)「マルチメディア・アーキテクチャのユーザインタフェースは，どうすべきか」という問題を発見できた.
(2)近来稀に見るおもしろいセッションだった.

山浦恒央
いろいろな考え方があり，自分の考えの狭さを思い知りました．今日は，随分勉強になり，少し自分が大きくなったような気がします.

林香
· SUMMIT のようなアプローチでは，UNIX のできることしかできない，とは本当か？ smalltalk の良い所との融合と考えられないか．しかもカンタンにできたのなら，X や GMW より良いアプローチかも？とは言いすぎかな.
·言語の U/I は，シンタックスかな？

岸田孝一
Smalltalk や Mac が好きな人は，これらの system に内在する model が好きなのだろうか？しかし，私には，これ

らの System の持つ User Model や Process Model は，あまりにもナイーブ（原始的）すぎるように思う．

歌代和正

オブジェクト間の関係がキーとなっているというのは，どうもそのようである．でも，関係というものをメタなレベルで定義するというのは，どうであろうか？自然界は，誰かがまとめて管理しているのではなくて，全てのオブジェクトが個々に独特の性質をもっていて，それに基づいて行動している．やはり，オブジェクトが個々にもっている他のオブジェクトとの関係に基づいてコントロールしないと最終的には無理なような気がする．

そこで，中野先生が言っていたウインド・マネージャについて思いついたことがあるが，ウインドの一つ一つに性質を与えて，それぞれのウインドが自立的に行動するようなモデルのウインド・マネージャを考えてみてはどうか？例えば，ウインドAは，Bについて動きたがるとか，Cのウインドは，嫌いなので，Cがくると逃げるといった具合いである．

当然，その時の情勢なども影響するわけで，ウインド間の力関係は，時間とともに変化するのである．

加藤朗

"ユーザインターフェース"が，あまりに広いので，議論が散失してしまいがち．

・プログラム（システム）が，提供する"モデル"

・モデルが固定された時点での入力・出力の技術

この両者の議論（を好む人）は，層が違う．

堀内泰輔

ＭＭＩを考える場合，我々ソフトウェア技術者がソフトウェア開発を行うシーンでのＭＭＩとエンドユーザにとってのＭＭＩの両者を別個に考えなければいけないと思う．（多分，両者が一致することはないだろう）

このセッションでは，前者のＭＭＩについての話が中心であったが，エンドユーザにとってのＭＭＩについても討論する場がほしい．

臼井義美

いいマンマシンインターフェースとは，生産性，信頼性なのか，気持ち良い楽しいことなのか．両方であればいいのは当然だが．

杉田義明

ＭＭＩを討論する場合，よくハードウェア上の問題とソフトウェア上のとが混在して行われるが，今回も一部その混乱があった．キーボード・マウスとそのウインド表現などが，その例である．

私が一番興味があるのは，ソフトウェアの可視的な表現であり，どんな表現方法が，ユーザにとって大事なのか，そのメカニズム，定量的な評価手段などであるが，あまり議論がはずまなかった．

酒匂寛

・マウスは，位置を決め，ボタンイベントを起こし，移動を表現することしかしていない．より進んだ入力デヴァイスとは何か．

・ウインドウが多いと使い難い．仕事の内容に関係ない作業が自動化できないか．

Secretary System

・勝手に整理される手帳があれば，今の大部分の On-line Documentation はいらない．

・ユーザ・インターフェイスの問題は，個々の tool の出来，不出来の問題は，余り問題（重要では）ないのではないか？それらを整理，統合する仕掛けと思想が重要ではないか？

・全ての構造を平たく見せようとするところに問題があるのではないか？紙を漉くところから始めたくはない．私たちの［ペンと紙］の標準は何か？

三者の意見は：

新田：「"ペン"の作り方」を与える方法論

吉村：「新しいペンを作った」

青木：「ペンの作り方」といったところか？

荻生準一

Unix 使い始めたばかりであり，いまいち話がつかめない！！参考に聞いています．

吉村鉄太郎

(1) Keyboard と Mouse は，Kなどの pointing device とは，本質的に異質なもので，その二種を併用することをユーザに強制させるシステムは，嫌いだ．例えば，Touch method でタイプできる人は，［すべての操作］を keyboard から手を離すことなく，行いたいのである．

(2) end user のための user interface と，programmer のための user interface とをハッキリ区別して，

discussion すべきだったと思う．

井川裕基

ユーザ I/F は，ユーザの好みに合わせて--- ユーザのモデリング

抽象的なモデル--- コンピュータ化しやすいが，間違っていたら，もうおしまい．"ちょっと例外処理"というのがうまく表現できない．

多彩な(?)モデル--- コンピュータ化しにくい．(要素が多いから) ユーザは，自分がどうしてよいのか分かりにくい．

では，その間の位置のものを選べばいいのか？選び方がむずかしい．

熊谷章

(1) マルチウインドウとマウスの使い方

それなりにおもしろい．やはり，Mac の影響は大である．

ロボット作成にも Mac が使われている．

(2) マルチウインドウの改善案

それなりにおもしろい．具体策は，連動するメカニズムが必要．

(3) ＭＭＩのモデル

リレーションの操作と表現

・object と relation は，別物と考える．

・what you see is what it is. relation は，object の属性として．

(4) ＭＭＩの方向

・マルチメデイア

・自然指向

小林貞幸

・やはり，自分で使っているツール（ハード，ソフト共に）の話に終始してしまった．

・討論に参加するには，使った経験がなくとも参加出来るはずであるが．

・汎用機の cobol の世界から見た，ＭＭＩやユーザインターフェイスについて，話したり聞きたかった．

中野秀男

・マウスはいいのか？マウスに視点，お絵かきをさせている（今），pointing device がマウス，お絵かきはペン，耳と鼻と皮膚は？

・ＭＭＩの定量評価：各ユーザの能力の差をどう取り込めばよいかなどの問題はともかく，定量評価は，どんどん出てきてほしいけれど．プログラムの complexity の研究のようにならなければよいが．

・ＭＭＩのモデル化 :正直いって良く分かりません．

・ＭＭＩの多様性 ：大賛成です．

新田稔

オブジェクトのさかい目

現実世界の混乱を持ち込む

コンピュータの中へ持ち込むと

人間　　コンピュータ

どちらにアダプトするか

好きなほうを選べばよいのではないか

行番号

古い人たちのことも考えてあげたい．

人が慣れてくれるまで待つか？

インターフェイスの基本は，叫ぶことだと思う．

御喜家貴子

人間すなわちユーザというのは，いろいろと勝手なことを言いだすものだと思った．例えば，マルチウインドウができたことにより，同時にいろいろな情報を入手できて嬉しいと言いながら，ウインドウが多すぎると，整理がつかないといって文句を言いだす．あーいえば，こーいうという感じで対応できない．やはりそうなってくると，なんでもユーザの希望を聞いてあげるユーザ I/F じゃなくて，ユーザの目的とモデルに合わせて，ユーザの我侭を気付かせてやる，強制的に許さないという態度で，マンマシン I/F を提供すべきだ．それは，例えば，車に乗るために教習所で訓練したり，スピードをある程度出し過ぎると「ピューピュー」と注意のブザーが鳴り出すのと似ている．それは，ユーザを守る目的や，安全のために，さらに指導するために考えられている．このような配慮こそが，喜ばれるアンマシン I/F を生みだすのではないかしら？

セッション2
ソフトウェア・データベース化の試み

チェアマン　佐藤千明（長野県協同電算）

プレゼンテータ　岡本隆一（神戸コンピューターサービス）

福田充利（日本電気ソフトウエァ）

堀川博史（三菱電機）

－　内容　－

- 事前アンケート
- プレゼンテータのポジション・ペーパ
- セッション・レポート
- 事後アンケート
- セッション・サマリ

いわゆるソフトウェア資産と呼ばれる各種ドキュメントやソースコード、テスト結果などをオンライン化し、気軽に参照したり再利用したりすることができるようにするための仕掛けとして、ソフトウェア・データベースなるものが必要であるといわれています。しかし、未だにこれぞ決定版というものは現われていません。各所で、実験的な試みはなされているものの、まだ完成にはいたっていないものと思われます。大規摸にソフトウェア資産のすべてをため込もうとするものや、とりあえずドキュメントだけに的をしぼってデータベース化を計るものなど、色々な試行実験が行なわれていることでしょう。このセッションでは、それらの実験結果を持ち寄り、これからの進むべき道を探りたいと思います。

熊谷　章／ＰＦＵ　研究開発センター第三開発室

◆作成済みプロダクトのデータベース

水平分散か集中分散かの形態にかかわらず、大容量のファイル・サーバとして機能するシステムを設け、ここで既存のプロダクトのデータベース・サービスを行う。プロダクトには、ドキュメントとソース・プログラムそれにオブジェクト形式プログラムがある。このプロダクトの検索は、初めは普通のデータベース検索でよいが、使用する中に段々とユーザ・インタフェースを使い易くするように工夫する。例えば、日本文の定形的な Query Language にする。同一プロダクトの世代管理もこのデータベース上で行う。

◆ノウハウのデータベース

ソフトウェア開発には、ツールや方法論を含んだ開発のコツがある。これをノウハウとしてデータベース化する。例えば、「UNIX 下における光ディスク・ドライバの作り方というドキュメントとそのドライバのソース・プログラム」が簡単な例である。考えてみると、日本中至る所で同じ仕事をしているようだ。一企業内は勿論のこと企業外にもこのノウハウを伝達し、便宜を図りたい。これは、多分ドキュメント・ベースというアーキテクチャになるだろう。検索は議題１と同じであるがインプリメンテーションは異なる。

岡本隆一／神戸コンピューターサービス

◆ソフトウェア・データベースの構造と入力の仕掛けについて

どのようなデータベースを構築するときにも問題になるのが、データベース入力の仕掛けである。特に、ソフトウェア・データベースに関する入力の仕掛けについては、開発プロセスやライフ・サイクルを通じて、自然と入力されていくようなメカニズムの実現が必要だと考えている。

◆再利用促進のためのデータベース・マネジメントの仕組みについて

ソフトウェアを開発するプロセス、あるいは、それぞれのプロセスに最適化された開発環境に対し、利用あるいは再利用を活性化させることのできる仕組みについての取り組みが充分でなかったため、これまでのソフトウェア・データベースの試みがうまくいかなかったのではないか。

佐藤千明／長野県協同電算

◆ソフトウェア・データベース、DD/DS を中核に据えた"一貫支援システム"の動向

各メインフレーム・メーカーから発表されている統合開発環境はいずれもデータベース、データ・ディクショナリを中核にしている。

・開発標準手順を規定した方法論を機械化支援する全工程支援の“統合開発環境”
・要求分析、業務分析、システム概要設計フェーズの機械化支援はこれから
・画面、帳票、ファイル、レコード、データ項目をソフトウェア・データベースに登録
・プログラムはパターン化、部品化により半自動生成、又はチャートエディタから自動生成
・進捗管理、履歴管理用データベースがある
・ソフトウェア・データベースから何らかのドキュメントが生成される
・ COBOL、 FORTRAN、 C 等の 4GL 支援環境と "4GL" 支援環境が別々に存在している

あるメーカーは"これからは 4GL の時代"という
あるメーカーは"3GL の資産は大切にしなくては"という
業務に応じて使い分ければよい！

→二つの支援環境の統合化が必要となる（マルチベンダー・ユーザはもっと大変？！）

WS のインテリジェント機能を生かした MML はこれから

◆開発環境としてのメインフレームの存在価値

・水平分散型のデータベース管理は難しいのではないか、全社的なソフトウェア資産は一元管理の方が楽で確実
→メインフレームをソフトウェア・データベース・ M/C として使おう
WS 上のソフトウェア・ データベースの使い方は

Private、Local、In-Process、ホスト内 DB の Instance
・高性能 WS・ミニコンを実行 M/C、運用 M/C として使うにはデータベース管理が弱い
→結合テストは DB/DC の動く実機＝メインフレーム上で行う必要がある
→これにかなりの工数がかかる
高速大容量処理の行えるコンピュータを"メインフレーム"と言ったら、その開発環境は垂直分散？
・データベース上の情報を使って自動生成
システム・フロー、プログラム、JCL、ドキュメント

中野秀男／大阪大学　工学部通信工学科

◆ソフトウェア・データベース

やりたいけれども、むしろ外でやった成果を取り込みたい。我々の研究のプログラム作りに関しては部品化として WS 上でやるつもりです。

深瀬弘恭／アスキー　システムソフトウェア事業部

◆どのような Software DB が可能なのか？

アルゴリズムなのか、ソースコードなのか、モジュール単位なのか。これは適用分野によって大きく変化するのではないか。既知の技術（事務処理分野等）と新規開発では事情が異なるのでは？

藤野晃延／富士ゼロックス情報システム　技術部

◆ SWDB の基本的なコンセプトは何か？

何が出来ないと SWDB でないのか、何が必要十分条件なのか、論理的な SWDB の Function を明確にしたい。

◆○○ DB や Hypertext は SWDB となりうるか？

○○ DB や Hypertext といった考え方を用いて、SWDB は構築できるか、出来るならその方法論(Frame Work)を明確にしたい。

久保宏志／富士通　システム本部パッケージ企画統括部

◆ SWDB からの要求と RelDB テクノロジーのステータス（現状と展望）

シグマの人から、これまでの検討の経過、到達点と、これからの予定を聞いて、建設的なコメントをクリエイトする。

◆ SWDB と、トランザクション構築処理アプリケーション支援ツールとしてのディクショナリ・システム（概要の整理）

分散化の先進事例の紹介をうける。アプリケーションは、開発環境である必要はない。Stone Automation でも、三次オンラインでもよい。

岸田孝一／ＳＲＡ

◆要求／設計仕様をどうやって作っていくか？

ソフトウェア仕様の作成は基本的にチームワークだと思う。そのための支援をどうすべきか。方向づけをはっきりさせたい。

W.Riddle が試作した JOSEPH システム（SRA でその一部を追試した SPEX システム）のアイデアは、ある程度有効だと思う。

いま MCC で試作されている Plain-Text や Issue-Based システムも興味深い。

これらについては、Workshop までに、簡単なレポートをまとめてみたいと思っている。

◆開発アクティビティと成果オブジェクトとの関連をどうとらえるか？

ERA-Diagram によるモデル化が有効なように思われる。（たとえば、TRW の Penedo たちの試み）が、ことの本質にどこまでせまれるか？効果的な支援システムが作れるのか？少し真剣に議論してみたい。

御喜家貴子／横河ヒューレット・パッカード　マーケティング部門

◆ソフトウェアの再利用性というのはどれぐらいあるのでしょうか？

データベース化する目的は、再利用と統計の２面があると思う。統計情報源としてデータベースを利用するのは、もっともなことと思うが、再利用化のためには、データベースそのものが改善され、自由度の高いものになるとともに、ソフトウェアの設計思想も再利用性を考慮しなければならないと思う。しかし、私はよくわからないのですが、ソフトウェアを再利用しているソフトウェア開発者というのは、一般にどれぐらいいるのでしょうか？又、どんな種類のソフトウェア（アプリケーション）ならばそのようにして開発の工数削減が可能なのでしょうか？

堀川博史／三菱電機　情報電子研究所システム・ソフトウェア開発部

◆データ宣言の方法

ソフトウェア・データベースをスキーマベースの定義方法で行うにはむりがある。コマンド（とライブラリ）を揃える方向が有効だと考える。

◆図形データの扱い

上述（議題１）した方向をふまえて、特に設計情報としての図データを処理するコマンドとライブラリを作成していく必要がある。

伊野　誠／ＳＲＡ

◆知識ベース･システムは本命たりうるか

通常のソフトウェア･データベースは、既存のデータベース･システムを利用して EAR モデルで実現されるものが多い。この場合、システム作りと保守が大変である。

REFINE に見られるようなオブジェクト指向の知識ベース利用のものは、実用システムに近いと思う。

鶴見泰久／東海クリエイト　VAR 推進室

◆開発ツールについて

ハードウェアの進歩について行ける開発ツールを提供されなければならない。

コンパイラー、リンカー、データベースが一体となった開発システムが理想的。

山浦恒央／日立ソフトウェアエンジニアリング　第１技術本部第５設計部第４課

◆再利用性について

ソフトウェア開発環境のねらいの一つは既開発ソフトウェアの再利用であるが、その仕掛けとしてどのようなものが適当であるのか？あまり完全をねらいすぎるとディスク容量、その保守、使用性に問題が生じる。再利用における方法論が必要ではないか？

森　幸一／ＰＦＵ　開発企画部ソフトウェア検査課

◆関連付加情報の蓄積

データベース化された情報は時間の経過とともにその価値が当然変化していく。利用された場合のフィードバック情報を関連づけて蓄積していく必要があろう。こうした付加情報を資産そのものと関連づけておくことにより、そのときどきでの情報の価値の正当な評価が可能になると思われる。

◆ソフトウェア技術情報の管理体系

ソフト資産は、ドキュメントだけでも、ドキュメント、プログラムなど関連する情報一式としてでも流通可能となることが望まれる。そのためには、開発部門のみならず、検査、製品製造、保守部門など関連する部門合体で共通して認識できる情報の管理体系の確率が必要と思われる。

吉村智香子／ HST

◆ドキュメントチームの結成

ソフトウェア開発において、情報の蓄積と検索は重要である。ソフトウェア開発における情報には、次の３種類がある。開発段階のドキュメント、メンテナンス用のドキュメント、管理用のデータである。これらの集中管理がデータベースシステムによって実現されるべきである。

現在担当しているプロジェクトでは、情報を蓄積、検索することのみに専念するドキュメントチームを結成している。現段階では、UNIX のファイルシステムを利用して開発段階でのドキュメント＝メンテナンス用のドキュメントを蓄積している。蓄積するためのツール、検索するためのツール等もドキュメントチームでつぎつぎに構築していっている。各メンバーの評価もまあまあである。

しかし、まだ設計段階であるので情報量は整理できる程度である。これが、テスト段階まで進むとかなりの量になることは予想できる。プロジェクトすべての管理を行うとさらに莫大な量になるといえる。ドキュメントチームに望まれるのは、短発なツール、管理である。

◆ソフトウェア部品の再利用

ソフトウェア部品化はソフトウェア開発の作業を楽にするという考え方は、前々より論じられてきたことである。

FA 業務をやっている弊社でも、今やっとこの考え方が取り入れられようとしている。システムコンポーネントの組み合わせでシステムが構築できる環境が必要であるという考えが、ようやく出てきたのである。

この環境のイメージは出来上がっている。システムコンポーネントを登録するデータベースと、条件に合わせて必要なコンポーネントをデータベースより取り出して組み合わせるツールである。

ただこれだけの環境であるが、実際にこの環境を構築するのは中々困難である。それは、第一段階のシステムコンポーネント化がネックとなっている。再利用できるコンポーネントの調査、つまりノウハウを引き出すことが進まないのである。

また、大量のコンポーネントがデータベースに蓄えられると、それを最大限に活かすツールが必要である。ここで登場するのが AI である。ある条件を与えると知的に検索を行って、最適なシステムを作るツールを構築するのが、現在目標とするところである。しかし、実現方法はまだ見つけていない。（まだまだ、理解不足です。）

村川貴広／ HST

◆実践的データベース化方法論

あるデータベースを構築しようとした場合にそこにどのような情報があり、それをどのように整理していくかを考えなければなりません。大規模にソフトウェア資産のすべてをため込む場合、その整理が重要なポイントだと考えます。このことに関しての討議を行いたい。

西岡健自／横河電機　研究開発４部第１研究室

◆ソフトウェア・データベース構築のためのソフトウェア・データベース

ソフトウェア・データベース上の資産を活用して、再利用の実をあげるのは一朝一夕に成るものではない。まず、空のソフトウェア・データベースにソフトウェア資産を順次つくり込むことが必要である。そのつくり込には必ずしも再利用をあてにできない。ある程度の資産蓄積の後にも、信頼性の高いソフトウェア部品の開発には再利用を活用できない場合は少なくない。つまり、再利用を想定しないで高信頼性ソフトを高生産性で開発できる環境なしには、実際的なソフトウェア・データベースは存在し得ない。空の状態から人の助けを借り、その開発作業を支援しつつ自ら内容を充実させるソフトウェア・データベースが理想である。

斎藤信男／慶応大学　数理工学科

◆データベース管理システムの選択

ソフトウェア・データベースに適合する DBMS は、どんなものが良いか。RDB では駄目であろう。

◆ソフトウェア・データベースの内容

何をデータベースに格納すべきか。どの情報を捨てるべきか。

和田　勉／長野大学　産業社会学部

◆ソフトウェア・データベース

これについてはあいにくほとんど何も知りません。強いて関連する事といえば、JUNET の fj.sources で流されてくる PDS のいくつかを、大変有難く使わせていただいていること位でしょうか。fj.sources に限らず、今まで流れてきたニューズは（初期にセーブに失敗したいくつかを除き）全てフロッピーに保存してあります。SUN が動きだしたので、今まで BSD 系でないため使えずにただ保存してあった PDS を先日いくつか引っ張り出してきました。

新田　稔／ＳＲＡ

◆なにをデータベース化するか

「解答」だけの DB 化ではなく、問題の DB 化、どうやってその解答を得たかという方法の DB 化、なぜその解答を選んだかというデシジョンの DB かも必要。

佐原　伸／ＳＲＡ

◆オブジェクト指向ソフトウェア・データベース

HyperTalk をモデルにした、オブジェクト指向のソフトウェア・データベース定義言語により、ビジュアルな、ソフトウェア・データベースを構築する必要がある。

ソフトウェア・データベース環境より、アドベンチャーゲーム構築環境の方が進歩しているのでは？

◆分散ネットワーク上のソフトウェア・データベースへの移行

既存のシステム（特に汎用大型機）のリソースを、いかにして、分散ネットワーク上のワークステーションのソフトウェア・データベースに移行するか？

そのためのインクリメンタルなソフトウェア・データベースのアーキテクチャを考えたい。

福田充利／日本電気ソフトウェア　OA システム事業部第三開発部

◆ソフトウェア開発ドキュメントのデータベース

マトリクス構造を持ったドキュメント体系→ポジションペーパ参照のこと。

◆ソフトウェア開発情報における「規則←→判例」モデル

仕様を「規則」とすると動作（or テスト結果）は「判例」である。設計という作業はユーザが望む動作を「判例」空間から「規則」空間へ写像することである。この２つの空間を正しく定義し、その写像則を確立できれば、設計自動化に大きく寄与できるものと考える。

歌代和正／ＳＲＡ　環境開発部

◆ソフトウェアの構造

今世の中にあるソフトウェアを１ケ所に集めてもソフトウェア・データベースにはならない。そのためには一つのソフトウェアを構造的に管理しなければ　ならないと思うが、どうやってやるのか？

桜井麻里／ＳＲＡ　環境開発部

◆使いこなしてもらうためには

いわゆるソフトウェア資産をデータベースにいれただけでは十分ではない。その情報をさがし出し、利用するための便利な仕掛けが必要なのであるが、いまは、そのため

のツールがバラバラに存在していてとても使い難い状況である。一定の操作モデルを想定し、それに基づいてガイドしていくような機構が是非とも必要である。また、ソフトウェア・データベースの構築によってソフトウェア開発のライフサイクル自体も変化してくることが考えられるので、そのへんも考慮する必要がある。もちろん、ユーザにカスタマイズしてもらうための機構を用意しておくことも重要である。そしてそれは分かり易いものでなければならない。

◆なにをどのようにバージョン管理していくか

設計情報やプログラム定義情報などは、確かにデータベースに入れてあるけれど、その差分を機械的にとっておいただけでは、人間にとって意味のあるものとはならない。古いバージョンのソースやマニュアルなどをいちいちとっておいてもディスクが一杯になるだけで、余り意味がない。(もちろん、ある程度の短期間においては意味があるということは否定しない。) なぜなら、そのバージョンが動くためのトータルな環境を保存して置かないかぎり結局使えないからである。そして、そのトータルな環境の保存というのはほとんど不可能である。とすると、なにをとっておけば良いのか。結局の所、『開発初期のコンセプト』と、『時と共に要求がどのように変わっていったか、そしてそれにどう対応していったか』ということを保存して置くことが一番大切のように思うのだがどうだろうか。そして、それはやはりドキュメントという形でマシン(外部記憶装置)に置いておくしかないのだろうか。

田中正則／ＳＲＡ　開発第１部

◆インフォーマルな情報の SWDB 化

事務アプリケーションの開発時に、DD の導入を行なってみた。その概要は、ファイル、帳票、画面等の設計書の作成に使用する項目を DD 化し、自動、半自動的に設計書を出力するというものである。このようなフォーマル情報の DD 化は、簡単にシステム化が出来、効果も上がる。しかし、インフォーマル情報の SWDB 化は、まだまだ技術的問題点が多い。これからはインフォーマルな情報の SWDB 化をどうしていくかが SWDB 化にとって大きな課題である。

北野義明／ KCS　ソフト開発部技術第一課

◆情報検索におけるキーワード

ドキュント・データベース管理システムの関係の経験から、DB 検索の際のキーワードについて、キーワードの与え方、分類方法を設定することが、ドキュントそのものの役割を決定していることに大きく関わっていると思った。そこで、キーワードの与え方、分類方法に対する考えを議論したい。

◆ソフトウェア開発者に必要なデータベース

ソフトウェア開発者にとって、どのようなデータベースが必要なのだろう。例えば、オブジェクト・プログラム・コードの D.B は本当に必要なのか？ドキュント DB にはどのようなドキュメントがどのように管理されていればよいのか？設計ドキュメントとソース・コードとの「設計情報」というインターフェースを DB とどう絡めて行くのか？

田中一夫／山一コンピュータ・センター　開発第二部基本システム課

◆保守用 DD/DS の構築

以下のシステムを現在構築中である。
開発済みのソース・プログラムから、使用しているレコード・ファイルの名称やフィールドの名称、桁数、データ形式などの情報を抜き出し、TSX1100 のデータ・ディクショナリに反映させる。従来の DD/DS は、新規プログラムのみ対象としているのが多いが、今回のは現在稼働中のプログラムのデータに的を絞り、保守性の向上を目的としている。具体的な例として、従来だとデータ項目を一つ変更するのに何千本ものプログラムを洗いだし、確認していたが、この DD を使うことにより数秒でインパクトがわかり、　生産性の向上につながるはずである。
参考資料：保守用 DD(VEGA)概要

◆ドキュメント

システム分析～システム設計まで、J-Star で管理している。また、プログラム仕様書なるものは、J-Star では作成せず、ホスト出力を目指している。このようにした理由として、下流工程のドキュメントは保守用にしか使わず、当初作成は手書きでも関係ないはずである。また、工程を考えても手書きの方が早く出来る。しかし、上流工程は現在のホストでは簡単に作成できず、J-Star で作成している。入力は各担当者(SA-SE)ということで進めてきたが、最近、各担当者では入力が間に合わず、専任オペレータを雇った。これは、スケジュール的な問題なのか、　J-Star を使いこなせないレベルの SA-SE、YCC のドキュメントは J-Star には向かない。等色々考えられる。. 現在分析中

田中慎一郎／ＳＲＡ　ソフトウェア工学研究所

◆ソフトウェア･データベースの利用法

もともとは、単に参照したり、また再利用したりする目的で考えられているのだろうと思われますが、これをもっと積極的に教育（というより学習）用として考えている方がいらしたらお話しをお聞きしたい。ソフト開発は「人間の仕事」だから「過去の成果物」ではなく、成果全体、つまり「考えたこと」、「結果としての評価」、「理由」などをうまく活用して、疑似体験学習的なことはできるとすごいと思う。経験は最大の資産であるから。

小池幸徳／日本システムサイエンス　システム部システム技術課

◆部品化によるソフトウェア･データベースの構築

ソフトウェアのライフサイクルにおいて、それぞれの工程は部品の組み合わせによる一連の製造工程と生成物の管理というようにとらえることが出来ると思う。この場合、部品は過去の設計、プログラミングにおける知識を意味する。このような部品を上流工程から下流工程まで有機的に組み合わせ、かつシステム全体の整合性を維持できるような DB について考えたい。

◆最適な情報の検索

議題１に記したような DB について考えてみた場合に、その DB の構成、アクセス方法が問題になってくる。もちろん、利用者が必要な情報(部品)をタイムリーに検索でき、最適化されなくてはならないだろう。このような DB の構成･アクセスについて考えたい。

青木　淳／富士ゼロックス情報システム　技術推進室

◆ソフトウェア･データベースの検索方法

ソフトウェア資産をむやみにため込んでもダメでしょう。必要なときに、必要なことを的確に検索できなければなりません。ODA のようなちゃんとした構造を考えるべきです。また、検索のインタフェースとしては、HyperCard のようなものが適当でしょう。

鎗友　良／富士ゼロックス情報システム　企画課

◆ ADT の概念に基づくソフトウェア･データベース

ソフトウェア･データベースに単にプログラムをため込むのではなくて、ソフトウェアの仕様とその実現としてのプログラムを一緒に保存したほうがいいと思う。ADT（抽象データ型）の概念に基づいて構築したソフトウェアの知識ベースが有望なアプローチの１つであるかも知れません。

野中　哲／日本電気　マイクロ波衛生通信事業部

◆公正なデータベースをどうやって実現するか

資産として残す必要のあるソフト、残す価値のあるソフトは、身近なところをみてもそんなにあるものではないと思う。となると、ある程度広い範囲からソフトを集める必要がある。公立の図書館のようなものがあればよいのではないか？

◆ソースコード以外は絶対信用できない

残すドキュメントの量は、もっと少なくすべきだ。読みもしないドキュメントを一所懸命書いているケースはよく見かける。洗練されたドキュメントを少量残す習慣を広めることが必要。

井上　尚司／ソフトバンク総合研究所

◆データベース登録の採否の問題

採否の程度が問題となる。

甘いと量が増え使いやすそうに見えるが、把握が難しくなり、かつ、使おうとする側が信頼しなくなる。（この点が一番問題では？「信頼できる」という状態をいかに作るか。極端な話、「あいつが作ったソフトは使えねぇぜ」の発言をどうすればよいのだろう。データベース管理者、ライブラリアンがソフトを確実に把握することから始めよう）

からいと、量が少ないので、使うのは楽になり、一見よさそうだが、実は登録しようという気がなくなってしまうので、伸びがなくなる。

河田　亨／シャープ

◆ソフトウェアの評価表

データベース化には、各種付加情報が必要になるが、ソフトウェアの評価も必要となろう。一昔前であれば、オブジェクトサイズなどは評価の中で比重も大きかったであろうが、現在は操作性等が重視されている。評価が可能かといった問題も含む。

近藤康二／ソニースパーマイクロ事業本部ワークステーション事業部

◆既存のベースのドキュメントとの共存

非常に重要な問題であると思う。私の知っている多くの場合、この問題がクリアされず、システム立ち上げ時に気を逸して失敗している。私はこの問題に対しては、イメージファイリング･システムの導入が不可欠であると思う。つまり、既存のドキュメントは、イメージとしてシステムを立ち上げ時までに計画的に専門業者によって高速入力をし、初期の早期安定を計ればよいと思う。しかし問

題は、キャラクタでないイメージデータの取り扱いが現在のところシステム的に完全に解決されていないところに依然問題がある。

井川裕基／日本電子計算開発本部ファイプロジェクト

◆ソフトウェアの評価表

あえてノーコメント

加藤　朗／東工大情報工学部

ソフトウェア自身で自己完結していることは少なく、環境によって意味に影響を受ける場合が多い。従って、いくらデータベース化して、過去のソフトウェア部品を再利用しようとしても、適切な修正が必要となる。意味が自己完結しているようなソフトウェアの部品を開発するための手法·言語·ツール·評価技法が、データベース以前に必要である。これは大変難しいのでこの分野の研究は他の人にお願いする。

中村暢夫／日立ビジネス機器　技術部

◆蓄積と検索をどのように行なうか

·過去の蓄積をいかに現在の仕事に生かすか

·他人の成果·知識をどのように自分の仕事に反映させるか

·作成者だけではなくて、閲覧者にもいかに理解しやすい形で提供するか

が議題だと思っていますが、それ以上に考えが進みません。

◆データベース化の基準について

データベースを作るときに何らかの選定基準を設けるべきかどうか知りたい。広く捉えれば、ソフトウェア部品の作成方法にもつながりうると思っています。

3rd SEA Environment Workshop Position Paper

氏名 岡本隆一	種別 ○会員 no 850177 □一般
所属 株式会社神戸コンピューターサービス（ソフト開発部，技術第一課）	
住所 〒（ 神戸市中央区浪花町64番地 電々ビル4F (〒650)	
TEL 078(391) 8291(代)	FAX 078(392) 0294

仕　事

9課1分室で構成されたソフトウェア開発部門<ソフト開発部>の中で、直接お金儲けに寄与しないセクション<技術第一課>に在籍し、主として、開発環境の高度化と技術者教育に関する諸問題をテーマに掲げ、企画･立案から運用･評価までを行っている....、<メンバー、グループのマネジメントをやっている>。

作業環境

自分の身の回りは、ある意味での実験環境が雑然と置いてあり、全てを一応<最低限物理的には>ネットワーク接続している。それらを若手エンジニアそれぞれが担当しているテーマに沿って、構成を組みかえながら使っている。私の横には、J-Starワークステーションが一台。

(1) メインフレーム
複数のFACOMホストをアクセスしたTSS環境。

(2) UNIX
LAN接続したワークステーションおよびパーソナルコンピュータ。

(3) XINS
部内全体にLAN接続したJ-Starワークステーション。

現在の問題

環境の分散化を推進しているが、マネジメント支援が追いつかなくなってきた<何をやっているのか分からない>。環境の複雑化･高度化が進むにつれて増大してきた維持工数、管理工数の捻出が大変。

(1) 情報コミュニケーション
<ニーズはあるが>情報システム化の芽が育ちにくい。

(2) マネジメントの手薄化
手遅れの評価でなく、ソフトウェア開発プロセスの中に食い込んだマネジメント支援の仕組みを育てたい。

(3) 人的問題
やらなきゃイケナイ<と思われる>課題は山積しているが、人<人材面、工数面>が足りない。

次ページの討論テーマ票にも，必要事項をご記入の上本紙とあわせて 12 月15 日まで（締め切り厳守！）に下記のSEA 事務局までお送りください．もちろん，本式の Position Paper も同封していただくほうがベターです（既発表の論文を添付していただいても構いません）．

ソフトウェア技術者協会 (SEA) 〒102 千代田区単町2-12
TEL: 03(234)9455　FAX: 03(234)9454

3rd SEA Environment Workshop Position Paper

氏名 福田充利	種別 ☑会員 no871154 □一般
所属 日本電気ソフトウェア(株) OAシステム事業部オフィス開発部	
住所 〒(108) 港区高輪 2-17-11	
TEL 03(444)3211	FAX 03(444)6809

仕事

現時点ではプログラム開発部門から離れて品質保証グループの中で生産技術という名の下で仕事をしている。短い期間で成果を上げられることを要求されているが、その期待に応えられずにいる。生産技術という旗印自体が危機に瀕している。今後、企業内ローリング・ストーンという話もある。

作業環境

生産技術を適用するグループの環境はNECのローカルOS PTOS, NTOS, ITOS及び一部のMS-DOS, UNIXである。マシンは16~32bitのパソコン、オフコン。

現在の問題

ソフトウェア・ファクトリという名の下にソフト産業の荒廃が進みつつある。

次ページの討論テーマ票にも，必要事項をご記入の上本紙とあわせて 12 月 15 日まで（締め切り厳守！）に下記の SEA 事務局までお送りください．もちろん，本式の Position Paper も同封していただくほうがベターです（既発表の論文を添付していただいても構いません）．

ソフトウェア技術者協会（SEA）〒102 千代田区隼町2-12 藤和半蔵門コーアビル505
TEL: 03(234)9455 FAX: 03(234)9454

マトリクス構造を持ったドキュメント体系

As a Position Paper for The 3rd Environment Workshop/SEA

日本電気ソフトウェア㈱
ＯＡシステム事業部
第三開発部　福田充利

はじめに

ソフトウェア設計時におけるプログラムのブレークダウンをマトリクス的に（多対多対応で）行うドキュメント体系を提案し、その意味付け、簡単なモデルとその応用、将来への展望について述べる。

１．ソフトウェアドキュメントの現状

ソフトウェアの設計書は多くの場合構造的に、あるいはそうでない場合にも少なくとも階層的に記述されている。そのためにある外部機能（ソフトウェアの内部構造に依存しない、外部から見た機能）は構造的に分解されることによってどのモジュールで実現されているかを容易に特定できるはずである。しかし、実際にはある外部機能がどのモジュールにおいてどういうメカニズムで動作しているのかをこれらの設計書から読み取るのは非常に難しい。つまり外部機能とプログラムモジュールとの対応が明確にはなっていない（あるいは明文化されていない）のが現状である。このために例えばある製造中のプログラム製品について

- あるモジュールの製造の遅れに対応してどの機能のリリースが遅れるのか
- 外部機能を変更したいがどのモジュールを変更すれば良いのか
- １つのモジュールを更新しようとしたときその副作用として何が起こるか

などが設計書から読み取れず、担当者に尋ねないと判断できない

というのが現実である。

これはプログラムが必ずしも末端まで構造的に構成されていないことに起因している。何段階かのブレークダウンのどこかで構造的な（１対Ｎの）細分化規則が破られて、複数の上位機能と複数の下位機能が互いに関係しあい、結果として複数の外部機能と複数のモジュールが関係しあうことになる。

トップダウン的なブレークダウンを正直に行えば階層が下がるに連れて機能モジュールの数が指数的に増加してしまうので適当な階層でモジュールの絞り込みを行う、即ち共通ルーチンを導入するなどして複数の機能を一つのモジュールに対応付けてしまう。

こういう経過を経て一つのプログラムモジュールは複数の外部機能に寄与するのが普通となり、この辺りの情報は文書化されることなく担当者にノウハウとして一時的に蓄積され、そして忘れ去られて行くのである。

２．マトリクス構造を持ったドキュメント

外部機能とプログラムモジュールが１対Ｎではなく Ｍ対Ｎで対応するのであればそれらの間の関係はマトリクスによって表現することができる。ここではまず外部機能項目とプログラムモジュールとを一つのマトリクスで対応付ける一段マトリクスによるドキュメンテーションについて述べる。表１－１に示すようにマトリクスの両辺（左辺と上辺）にそれぞれ外部機能とプログラムモジュールを洩れなく列挙し、関係のある機能とモジュールの交点にそれらの関係を記述することによって先に挙げたような問題は全て解決することが出来る。

３．どう作るか

このマトリクスを作成するにはまず機能を必要なレベルにまでブ

レークダウンし、それぞれの機能が実現される過程においてプログラムモジュールの各々とどういう関わりがあるのかを記述して行けば良い。

このマトリクスは小さなプログラムに関するものでもすぐに１００×２００ぐらいにはなってしまう。マトリクス要素をそれぞれ一枚ずつカードに書き込んで管理する方法も考えられるが、一般的に紙の上で操作するのは得策ではない。必然的に電子化することになる。いわゆる電子シート（表計算プロセッサ）が文章を扱えるようになれば良い。電子シートの各セルが対応するマトリクス要素を記述した文書のファイル名を持ち、それらの文書ファイルの加算（連結）を行う機能を持てばこのマトリクスを電子的に処理することが出来る。この電子化の検討は別途行う

実際には外部機能を分割したものだけではなく、システムを構築、運営、制御するためのモジュールが必要となる（例えばオーバレイ制御モジュール）。これらはプログラムがシステムとして稼働するために一つとして欠くことのできないモジュールであり、個々の機能項目との関係は他のトップダウンなモジュールとは違ったものになるが洩れなく記述しなければならない。

4．どう使うか

このマトリクスのある機能について関連するマトリクス要素（マトリクスの行を形成している）を集めるとその機能が実現される仕組みが読み取れる。一方あるモジュールについて関連するマトリクス要素（マトリクスの列を形成している）を集めるとそのモジュールの果たすべき役割が明らかになる。即ちこのマトリクスは設計時にも意味のある情報を与えることが出来るので一つの設計手法として活用することもできる。この件については後で触れる。

●読解性のよいプログラム構成ドキュメントとして

ある機能がどのようにして実現されているか、またあるいはあるモジュールがどういう機能に関連しているのかを一覧できる。但し、後述する電子化や多段マトリクスを実現することによってより一層の効果を発揮すると思われる。

●テスト項目のチェック

左辺にブレークダウンされた機能項目の一覧を持っているのでこのマトリクスはテスト項目を作る際にも良い基礎データとなる。マトリクス上辺のモジュールをコンパイル単位よりももっと細かい10ステップ程度のプログラムブロック単位で記述しておけば、マトリクス左辺の機能項目に対応したテスト項目に従ってテストを行う事によって、テスト項目の過不足（冗長性、網羅性）をチェックしたり、あるいはテストの合否判定の結果からどのモジュールが怪しいかを推定することが出来る。

このマトリクスは機能とモジュールとの関係をマトリクスの要素として持っているが、その一つ一つの要素についてテストすべき条件が何通りあるかを調べ、ある機能項目についてそれらの積をとるとその機能に関するテスト項目の数を知ることが出来る。この積の総和をとるとプログラム全体で必要なテスト項目の数となる。

●テスト項目の冗長性、網羅性のチェック

また、例えば成功しなかったテスト項目に関係するプログラムモジュールは全て障害を含んでいる可能性があるが他のテストにおいて同じモジュールが同じ機能を果たしていることがわかればそのモジュールは（その問題となっている機能については）正しく動作していることになる。このような論理を適用することによってテスト結果あるいはテスト経過から不良モジュールを摘出することが出来る。

あるテストケースに関係するモジュール群が別のテストケースに関係するモジュール群を同じ機能について含んでいるとき、前者の

テストケースは後者を含んでいることになり前者のテストが成功しているならば後者は行う必要がない。

マトリクス全体を見渡した時、全てのモジュールについてテストすべき条件の全てがテストされているならばホワイトボックステストとしては満足できるものであると言える。この条件を満たしていないときにはこの条件を満足するよう努力すべきである。

5．マトリクス構造ドキュメントの将来への展開

このマトリクスは更に次の方向へと発展する可能性を持っている。

- マトリクスの電子化によるソフトウェアデータベースの構築

先に示唆した通り、マトリクスの各々のセルを小さな文書ファイルとし、そのファイル名をマトリクス状に管理するだけでも有効なツールとして利用することが出来る。さらにそれらの文書の内容をキーワード検索したり、前記のようなさまざまな仕掛を組み込むことによってまずはソフトウェア開発のためのデータベースと呼べるものが構築できる。

- プログラム設計法としての多段階マトリクスによるプログラムブレークダウン。

プログラムを設計して行く過程において外部機能は多くのレベルに階層づけられる沢山の中間的な機能に分解される。上記の例ではそれらの機能階層をすべて無視して同列に扱っており実際には出来上がったプログラムの保守用ドキュメントとしてしか適用できないだろう。実際にはもっとも外部的な機能からプログラムモジュールまでには何段階かのマトリクス形のブレークダウンを経て到達するのが妥当であり、またそういう形においてのみこのマトリクスは設計の手法として利用することが出来る。これらの中間的な機能項目を複数のマトリクスにどう対応付けるか、何段階かのマトリクスを構築して行く作業の中で何が起こるかなど興味ある問題を解決することによって新しい設計手法が誕生する可能性を持っている。

- 基本設計段階における自動的なマトリクス生成。

  プログラム設計の第一段階としての基本設計工程では実現すべき機能項目を列挙し、それらの間の包含／背反関係に注目して機能項目を整理する。この作業を機械化することによって第一段階のマトリクスの作成作業を自動化する。

- 三次元マトリクスへの発展。

  今回提案しているマトリクスを構成している機能項目、プログラムモジュールの二つの次元にデータ／テーブル／ファイルなどのモジュール間インターフェース要素を加えた三次元構造とすることによって、ある機能の実現をどのモジュールがどのインターフェースを使って実現しているかが明確になる。つまり、この三次元立体マトリクスは、機能座標をある機能に固定すればその機能に関するモジュール／データ関連表が得られ、データ座標を固定すればそのデータがどの機能の実現に関してどのモジュールと関わっているのかなどが一目瞭然となるなどプログラム構造に関する全ての情報が整理された形で保持されていることになる。

表1－1機能／モジュールマトリクス、及びその応用例

		XXXモジュール（コンパイル単位）			YYYモジュール（コンパイル単位）		必要項目数	関連テスト項目	テスト結果
		XX1ブロック	XX2ブロック	IX1ルーチン（内部モジュール）	IY4ルーチン（内部モジュール）	IY5ルーチン（内部モジュール）			
○○○○コマンド	パラメータ省略	～		～					
	パラメータ個数1	～	～						
	パラメータ個数2	～		～					
			～	～					
△△△△デバイス □□□□情報ポーリング	手順状態#D3					～			
	手順状態#F6				～	～			
	手順状態#G2				～				
	手順状態#K8				～				
プログラム規模［Step］		25	17	35	24	28			
言語		COBOL	COBOL	C	C	ASM			
担当者		山田	山田	川村	川村	海野			
日程	設計完	11／12	11／12	10／5	10／5	12／12			
	コーディング完	12／2	12／2	10／15	10／15	12／22			
	単体テスト完	12／16	12／16	11／15	11／15	1／14			
	結合テスト完	12／25	12／25	11／26	11／26	1／21			

3rd SEA Environment Workshop Position Paper

氏名	堀川博史　　種別 □会員 no　☑一般
所属	三菱電機(株) 情報電子研究所 システム・ソフトウェア開発部
住所	〒(247) 鎌倉市大船5の1の1
TEL	0467(44) 9084　　FAX 0467(44) 9106

仕事　ソフトウェア開発時，時に設計作業をワークステーション上で行える様にワークステーション上に設計支援ツールを開発している。現在は，主として仕様書を作成するためのエディタ(仕様書エディタ spec)とデータフロー図を編集するためのエディタ(データフロー図エディタ spider)の開発を行っている。

作業環境

MELCOM ME 1000シリーズ(cpuはモトローラ 68020，OSは Unix system V)をターゲットにして開発している。机の横に1台MEを置いて雪有して使っている。

ネットワーク関係ではバス型LANを用いてTCP-IPを利用している。

TCP-IPを用いて，プリントサーバ，他の計算機がアクセスできる。

現在の問題

ソフトウェアのデータベース化は目標の一つであるが，色々と難しいことが多い。

例えば①ツールが単独で動く，と，データベースを介して動くことを同時に目標としてかかげるのは無駄が多い

②データベースの設計を行う時，ツール設計と同時に行うとコンフリクトする。

③図形データの扱いに従来型データベースでは実現困難。

④スキーマ定義のデータベースではソフトウェアの内容が扱いづらい。

次ページの討論テーマ票にも，必要事項をご記入の上本紙とあわせて 12 月 15 日まで（締め切り厳守！）に下記の SEA 事務局までお送りください．もちろん，本式の Position Paper も同封していただくほうがベターです（既発表の論文を添付していただいても構いません）．

ソフトウェア・データベースに対する一考察

三菱電機㈱情報電子研究所
システム・ソフトウェア開発部
堀川博史

1．はじめに

ソフトウェアの設計作業は設計情報の編集、解析、生成の繰り返しである。設計情報はソフトウェア特有の図式を含んでいるため、従来は計算機による支援が難かしかった。そこで、分散型ソフトウェア開発支援システムSolon（Software Engineers Land on Network）〔1〕においては、設計の効率向上をねらい編集系を中心とした各種ツールを、ソフトウェアエンジニアリングワークステーションのビットマップディスプレイ等の特性を活用して開発した（ハードウェアはMELCOM ME1000 シリーズ、ＯＳはｕｎｉｘ　ＳｙｓｔｅｍＶ）。　次に設計支援ツールの概要を示す。

①仕様書エディタ（Ｓｐｅｃ）〔2〕

仕様書の標準構成を仕様書定義として与えておくことで、規則通りの仕様書を作ることができる日本語エディタ（図形処理を含む）である。文書の論理構造（章、節、項単位）の編集ができる。

②データフロー図エディタ（Ｓｐｉｄｅｒ）〔3〕

設計の上流で機能分析を行うときに、データの流れに着目しながら機能の詳細化を行なう為の図形エディタである。

③プログラム図エディタ／コンパイラ（Ｐｅｃ）〔4〕

外部設計の最後の段階から内部設計全般にわたって使用されるエディタである。モジュール構成図、モジュール仕様シート、プログラム図を編集し、Ｃ言語のソースコードとの間の変換ができる。

現在、設計の一貫支援を目差して、ツールの統合化を行なっている。ツールの統合はデータ変換器を用いれば、技術的には可能である。しかし、それでは美しくない。そこでの問題点はソフトウェアデータベースの在り方である。

2．ソフトウェアデータベースの問題

一般のデータベースはユーザに、データを検索し表示する機能と、データの変更／追加機能を提供する。一般に、データの宣言のし方は、スキーマを用いた定義的な方法が取られる。データベースの基本的役割は、データの意味付け（関係付け）と統一的データアクセスメソッドを提供する汎用的パッケージであろう。

ソフトウェアデータベースの難かしさは、幾つかあるが、ここでは、スキーマを用いた定義的な方法の問題と開発段階の難かしさとメディアの問題について述べる。

①スキーマを用いた定義的な方法の問題

ソフトウェアデータの一つとしてソースコードがある。ソースコードを１アイテムとするデータベースを考えたとき、アイテムのデータ長が可変であり、かなり長い場合がある

ことからあまり例をみないデータベースとなるであろう。そこで、ソースコードを細分化することが考えられる。一つの実現例は、プログラムの構文要素をアイテムと考えることであり、Ａｄａ言語の中間語Ｄｉａｎａベースの環境があげられる〔5〕。此のような環境が、正しい方向にあるかどうかは、わからないが、私はあまり好きではない。最も、好きでない理由は、データが構文木として、保存されていることによって、実際のデータの構造とエディタを通してテキストとして見ているユーザの見方との間にギャップがあるのではないかということである。一つの解に構造エディタがあるが、どうもユーザにコンパイラの１フェーズを肩代りさせているような気がする。

②開発段階の難かしさ

ソフトウェアデータベースとツールの開発のし方には３通りある。同時に作るか、データベースを先に作るか、ツールを先に作るかである。

同時に作ると、作業のフィードバックが大きいので開発が進行しない。

データベースを先に作ると、環境の統一は取れるが、データベース設計に失敗すると全体がだめになる。

ツールを先に作ると、美しくはないが（統一的アクセスメソッドを持たない）、単独ツールで使用できるし、データベース側で各種データの扱いのバラエティをもうけると、既存ツールの取り込みも可能である。

ツールを先に作っても、美しく環境が構築できる方法はないか。

③メディアの問題

製造段階で扱われるデータの形態は、ＡＳＣＩＩ文字である。しかし、設計段階で使用されるデータの形態は、日本語と図形である。既存のデータベースで各種メディアがあつかえるか。

３．考察

ここで、想定したいソフトウェアデータベースは、単に設計の一貫支援を美しく開発するためのライブラリではなく、ソフトウェア開発時のアウトプットを一元的に管理する為のものである。例えば、ソースコードに関する設計情報、テストデータ、テスト設計情報がマルチウインドゥで同期を取りながら、同時に見える環境のベースとなるものである（丁度、ｃｔａｇｓ〔6〕の拡張のようなもの）。この前提に立ち、上述した問題を考察したい。

①スキーマを用いた定義的な方法の問題

データベースのもう一つのデータの宣言のし方にメソッドを規定する関数的な方法がある。ｕｎｉｘのａｗｋ（パターン走査及び処理用言語）を用いたフラット型データベースという論文〔7〕がある。その類推で、保存するデータはプログラムテキストのまま、ｕｎｉｘのテキスト処理コマンド或はプログラムテキストコマンドを組み合わせることによって、プログラムデータベースを構築できると考える。

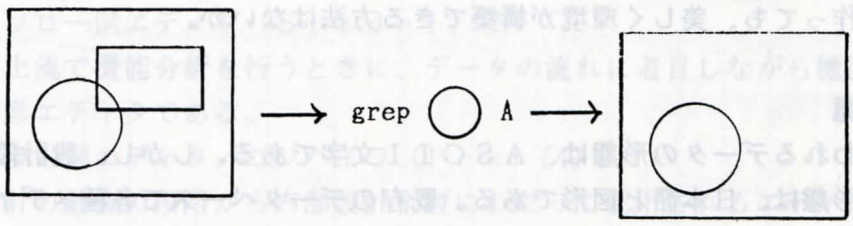
②開発段階の難かしさ

ツールを先に作っても、美しく環境が構築できる方法の一つは、ｕｎｉｘのツールがパイプインタフェースとして統一しているバイトストリームをベースにすることである。そして、図形等のデータに関しても同様の思想で拡張していく。図形に対しても、単機能のデータ変換ツールを豊富に揃えていけば良い。

③メディアの問題

ｕｎｉｘでは、通常のファイルはユーザからみたとき、すべてバイトストリームとみなせる。大型の計算機によくあるｘｘｘアクセスファイルといった豊富なファイルカテゴリを持つものではなく、簡略化がはかられている。単一構造であるにも関わらず、使用していて不自由しないのは、テキストファイルを扱うコマンド（とライブラリ）の多さによる。（例：ｃａｔ，ｍｏｒｅ，ｐｒ，ｆｉｌｅ，ｔａｉｌ，ｓｅｄ，ａｗｋ，ｄｉｆｆ，ｇｒｅｐ，ｓｏｒｔ，ｏｄ，ｒｃｓ）〔6〕。そこで、例えば、図形ファイルに対して次の様なコマンドを用意していくことが、ソフトウェアデータベースの第一歩ではないかと思う。

例：



４．おわりに

内容がｕｎｉｘ絶賛になってしまったが、決してｕｎｉｘが最高のＯＳであると思っているわけではない。ｕｎｉｘの是非については、いくつかの批判もあるが、代替するＯＳがない以上、私の立場は是である。

参考文献

〔1〕高野他：分散型開発システムで仕様書作成からプログラム生成までを支援する、日経エレクトロニクス、１９８７．７．１３。

〔2〕堀川他：仕様書エディタｓｐｅｃの評価、情処全体３５回、３Ｚ－２。

〔3〕寿原他：統合化ソフトウェア仕様書エディタ（４）、情処全体３４回、４Ｔ－４。

〔4〕北畠他：統合化ソフトウェア仕様書エディタ（５）、情処全体３４回、４Ｔ－５。

〔5〕Tartan Lab. ：DIANA REFERENCE MANUAL,Tartan Lab., １９８３．２。

〔6〕ｕｎｉｘのプログラマーズ・リファレンス・マニュアル等。

〔7〕D.Comer:The flat file system FFG,S-P&E, １９８２．１１。

第３回　SEA環境ワークショップ

セッション２

ソフトウェア・データベース化の試み

チェアマン　：佐藤　千明（長野県協同電算）
プレゼンテータ：岡本　隆一（神戸コンピュータサービス）
　　：福田　充利（日本電気ソフトウェア）
　　：堀川　博史（三菱電機）
レポータ　：塚田　道明（電算）

１．はじめに

佐藤： 事前に提出いただいたポジションペーパより３名のプレゼンテータを選ばせていただきました．プレゼンテーションが終了した後中間整理を行ない，続いて次のような切口で討論を進め，さらに共通する別の課題を合計３時間の中で討論したいと思います．

◇何のためにＳＷＤＢを構築するのか
◇何を蓄えるのか
◇どうやって蓄えるのか
◇どうやって検索するのか
◇いつ蓄えるのか
◇どこに蓄えるのか
◇誰が蓄えるのか

２．プレゼンテーション

２．１．使えるソフトウェア・データベース

（システムディクショナリの高度化からのアプローチと分散開発環境への展開）

岡本： ソフトウェア・データベースに絡まるかどうかわかりませんが，それをつくるカーペンタのつもりでお話します．

２．１．１．はじめに

ちまたにソフトウェア・データベースの話はよくでて来るが，いちばん肝心な"使える"といったものがなかなか見あたらないのが現状である．これに対して我々は７～８年前にさかのぼるが，コンピュータを動かすといった面での様々な問題に対して，コンピュータ・リソースの管理といったところからアプローチし，システム・ディクショナリーへの高度化を試みてきた．ソフトウェア・データベースを作るつもりは無かったが，結果としてソフトウェアを開発するグループにも貢献して行こうと考えている．これまでは富士通の８シリーズ，Ｍシリーズ上で集中型システムとして進化させてきたが，これからは分散型としてどう展開させて行くか，考えてみたい．

世の中でデータベースというのはこんなものだと言われ始めた頃（昭和５７年）ソフトウェアに関する社会背景は

◇ソフトウェア需要の増大とバックログの増加
◇世の中に大規模システムが増え，信頼性への要求が高まる　→　言語命令を基本部品として手作りしてきたが，抽象度を高めた"部品化"が言われ始めた
◇ツールの利用が叫ばれたくさんのツールはでてきたが，それぞれの現場にマッチしたものが少ない．またテクノロジー・トランスファも進まない，等があった．

これからのソフトウェア開発のやり方としては，ＣＡＤ／ＣＡＭ，オートメ化，プログラムレス化と言ったようなアプローチになると思う．しかしオートメ化，プログラムレス化があまり進むと，自分たちの仕事が無くなってしまうと言う心配がある．そこで，まずＣＡＤ／ＣＡＭを真剣に進めてみようと言うことになり，必然的にシステム・ディクショナリと言ったところに話が行き，さらに将来はソフトウェア・データベースへと発展させて行こうと言うことになった．

２．１．２．構築事例から

私はホスト・コンピュータの運用マネージをやっていた時期があり，コンピュータをうまく動かそうと言った所を中心にやってきた．ホスト集中環境での泥臭い話として，開発側のリソース要求に対する運用側の対応のむずかしさ（葛藤）がある．運用に携わっているとコンピュータを使ういろいろな人種が集まってきて，それらの全体を垣間

みる機会に恵まれる．新規開発をするグループ，システム保守を担当するグループ，コンピュータ運用をするグループ，のそれぞれを通過して来るシステムのライフサイクルを管理する立場から，システムディクショナリあるいはソフトウェア･データベースへのアプローチが始まったわけである．

新規開発あるいは保守と言ったグループに新規に参入して来るエンジニアには，コンピュータ環境がよく解らない．従って，とりあえず身近な環境または方法論で開発を進めてしまう．また運用側から見ると，その都度自分たちのコンピュータ環境マップを説明する惑ろこしさがある．従ってそうしたことをせず，かつ新たな開発方法論が生きるような形の環境を提供出来ないだろうかと言うことになる．そこで出て来るのが，コンピュータ･リソースのマネージをコンピュータ自体にうまくやらせられないだろうかと言うことである．

もう１つ，コンピュータ･リソースをマネージする立場での大きな悩みがある．ソフトウェアをつくる話はよくするが，"いつ捨てるか"と言う肝心な話があまり議論されない．毎年何十キロ･ステップもの新しいソフトウェアが追加され；またそれを稼動させる環境もますます複雑化しているが，どの部分が生きているか解らない．いつ捨てられるかを知るためには，コンピュータ･リソース（ソフトウェアの部品にいたるまでのリレーションも）全てがうまく管理されていないと確実な判断は出来ない．そう言った意味で，リソースの管理を真剣にやって行かなければならないと考えている．

これまでの話はコンピュータをどううまく運用し動かすかと言う事であったが，これから話をソフトウェアを開発するためのシステム･ディクショナリに移したいと思う．

ソフトウェアの新規開発や保守グループには，どういった形で仕事をしてもらうか．新規開発グループには，環境マップをよく知って実行環境にうまくアダプト出来るようなシステムをつくってほしい．また保守グループには，動的に変わるソフトウェア実行環境の情報を流し，うまく対応してもらう．その為には，いままで系統性なくバラバラに作られてきたツール群を，システム･ディクショナリをフレームとして統合化していけないか．さらに開発，保守，運用と言ったそれぞれのステージで持っている代表的プロセスをディクショナリにもって，実際の仕事上のプロセスを支援する事が出来ないか．また環境全体の管理と操作を一貫してうまく出来ないか，といった考えでディクショナリの高度化を進めてきた．

その結果，環境のマッピングから稼動実態のモニタリングまでをコンピュータに支援させるところまで来た．この間約７年間，そのうちこれらを実現するためのデータを入力するに５－６年を要しており，やはり長期間に渡って内容を充実させていかないとうまくいかないのではないかと思う．これらのコンピュータ支援システムは，どういった機能を中心として動いているかと言うと

(1)コンピュータ･リソース全ての関連を網羅しトレースできる．

(2)開発者の要求する環境にマッチ出来るプロセス，リソース，ツールを即座に提供出来る．

(3)不要，陳腐化したソフトウェアを確実に破棄しゴミを残さない．

(4)最終的にはコンピュータ運用の効率化に寄与出来る．

と言ったところである．

当初はシステム･ディクショナリとしても字引の域を越えておらず，単にそれぞれの語句のリレーションを表現するにとどまっていたが，ＤＢＭＳ機能の自由度がでてきた事もあって，付加価値情報を補いながら本格的な環境管理データベースへと進化させてきた．

次に開発と環境管理データベースの関わりについて，プロセスあるいは環境を提供すると言った仕組みを例に挙げお話する．

◇まず開発環境の創成破棄の例について

それぞれの開発プロジェクトから開発計画書が出されると，環境管理者は必要な定義体を環境ジェネレータに入れる．それにより環境管理データベースから必要情報を取り出し開発者に提供するといった一連のしくみになっている．

これの開発プロジョクトでの運用は図１－１のようになっている．

またシステム構造は，概ね図１－２の様になっている．

リソース管理機構とリソース管理情報は別階層にあるが，これが将来の分散型への展開のＫＥＹとなるものと思う．このような構造をセットで開発者に提供し，また情報については必要な部分を切り出して開発者に提供している．プロジェクトが完了した場合には，開発者に提供した環境からツールやオブジェクト等を，環境管理データ･ベースへフィードバックさせるしくみが必要となる．ドキュメントとオブジェクトの整合性チェックは，この再吸収の段階で行なわれる．再吸収が完了すると，開発者に提

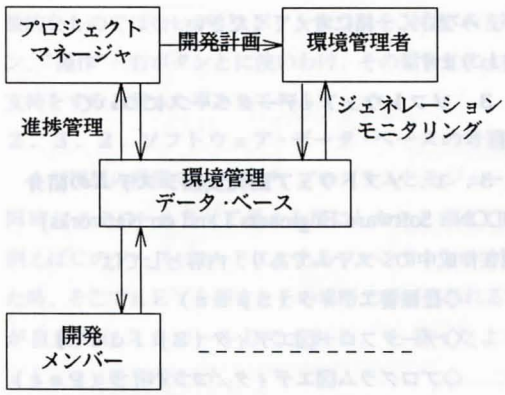


図1－1

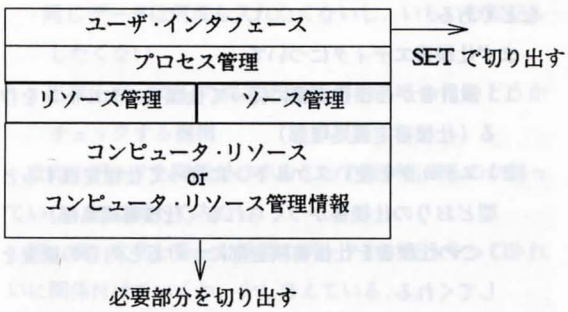


図1－2

供した環境は破棄される．またアクセス権も消滅する．

2．1．3．最後に

ソフトウェア・データベースを考えるときどうしても開発ステージという一部分にのみ言及してしまいがちであるが，関係するコンピュータ全体でのリソースとそのリレーションが，全て網羅されていないとうまくいかない．これがソフトウェア・データベースについて考えるとき私自身の信念としている"リソース管理ができていないまたは不十分なものは成功しない"と言うことである。

以上．

2．2．マトリクス構造を持ったドキュメント体系

福田： ソフトウェア・データベースと直接どう関わるのかはよく解りませんが，発表する機会に恵まれましたのでお話します．

ソフトウェアは作ってしまった．それをどう保守するの，誰がそれを見てくれるの，と言う話から．

ソフトウェアの設計書はあまり保守の役に立たない．だってつソフトウェアの設計書が武器の設計書と同じように，取り合う(盗む)なんて言うことが考えられるだろうか．(笑い)

なぜならば，現実のソフトウェア設計書では

◇仕組みが解らない

◇どこをいじればよいか解らない

◇いじるとどうなるか（副作用）解らない

からである．

ではどうしたら良いか．その前に技術者の頭の中では，開発課程の中で何が起こっているのか考えてみたい．概ね図2－1のように情報量を捉える事が出来る．

まず，リクワイヤメントの段階から構造的，階層的に設計しようという話がある．

要求定義を順次細分化の方向に分割して行くと，なんとなく機能はどう有るべきか解って来る筈であるが，実際はそううまくはいかない．なぜならば共通モジュールまたは部品と呼ばれるもので縮退化され，階層的にはツリー構造からはずれてくるからである．この辺の事情が解っている人は"全体が本当に解っている人"と言う事になる．そこで，それをドキュメント化するにはどうしたら良いか．

階層的にブレークダウンされた機能とモジュール（プログラム構造）の関係をマトリクス化することによって，機能はどのような方法で実現されているかとか，モジュールはどんな機能を持っているかと言うことを，明確化して表現する事が出来る．(図2－2)

私の今回の主張は，これを電子的にデータ・ベース化することによって価値が得られるのではないか，と言う事である．

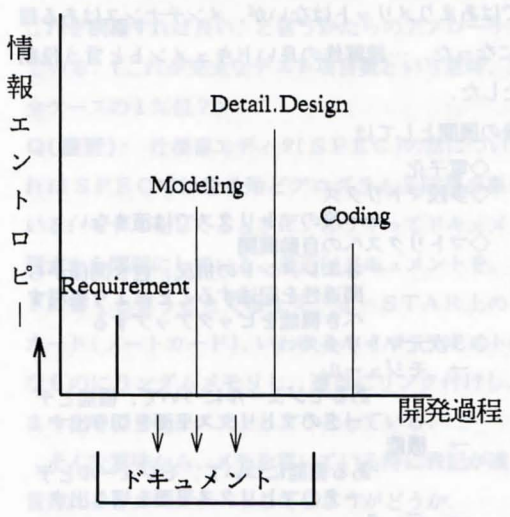


図2－1

機能↓	構造→	記述文	Mod. 1 1.1	1.2	1.3	Mod. 2 2.1	2.2	・・・	・・・・・
Fnc 1	1.1								
	1.2								
	1.3								
Fnc 2	2.1								
	2.2								
・・・	・・・								

図2－2

さて，それではこれ（マトリクス構造を持ったドキュメント）をどう使うか．

◇読解性の良いドキュメントとして

◇テスト項目のチェック

・機能ごとの完全テスト項目の把握

・テスト過程の管理

・テストカバレージの検査

と言った事である．

マトリクス構造ドキュメントの実際例としては，リアルタイム・アプリケーション・ソフトウェア，約３００Ｋステップの保守用ドキュメントを目的として行なった．機能細目毎，３８７枚のカードにそれがどう実現されているか（モジュール名）と言うことを記述した．これは１８３モジュールに対応していた訳であるが，この段階ではモジュールからの逆引きは出来なかった．この様な部分電子化ではあまりメリットはないが，メンテナンスはある程度楽になった．…読解性の良いドキュメントと言う役割は果たした．

今後の展開としては

◇電子化

◇多段マトリクス

←一段のマトリクスでは済まない

◇マトリクスへの自動展開

←各エレメントの排反，包含関係から関連性を記述することにより実現すべき機能をピックアップする

◇３次元マトリクス

→ モジュール

・あるモジュールについて，機能とデータのマトリクス平面を切り出す

→ 機能

・ある機能について，モジュールとデータのマトリクス平面を切り出す

→ データ

・あるデータについて，機能とモジュールのマトリクス平面を切り出す

更に，３次元マトリクスを多段化する事も可能と思いますが，みなさん一緒に考えてください．

終わります．

２．３．ソフトウェア・データベースについて

堀川：

２．３．１．ソフトウェア開発支援システムの紹介

(SOLON：Software Engineers Land on Networks)

現在作成中のシステムであり，内容としては

◇仕様書エディタ（Ｓｐｅｃ）

◇データフロー図エディタ（Ｓｐｉｄｅｒ）

◇プログラム図エディタ／コンパイラ（Ｐｅｃ）

◇ソフトウェア生産インタフェース（Ｓｉｆ）

◇ウインドゥシンボリックデバッカ（Ｗｓｄｂ）

などである．

まず仕様書エディタについて

１）設計者が仕様書定義に従って仕様書スケルトンを作る（仕様書定義処理部）

２）エディタを使いスケルトンに従って仕様定義すると型どおりの仕様書がつくられる（仕様書編集部）

３）この仕様書を仕様書検査部にかけると内容の検査をしてくれる．

と言った３つの大きな構成（ツール）で成り立っている．

仕様書定義部では章－節と言った項立てをし定義を行なうと，同時にその項目毎に仕様書エディタが動くときの仕掛を属性部として与えることができる．例えば，ガイドを出すとか，コピーしてくるとか，またはシェルを動かすとかの指定，と言ったものである．

機能を書くときはホール＆ピースで書いていくが，このとき既に記述された文書の自動複写機能やガイダンス機能が働くようになっている．また，チェック機能としては記述不整合の検出（同一性の違反検査）や作成状況（書き残し文書名や記述／ノードの比率）の表示と言ったものがある．

データフロー図エディタは，ＳＡ／ＳＥ用のツールと言ったもので，箱に機能を書き矢印でデータの流れを画面に向かって作っていくためのものである．

プログラム図エディタ（コンパイラ）は，モジュール構成を木構造で記述し，かつモジュールの仕様をシートに埋めていく．更にモジュール内の処理についてフローチャート（ＨＣＰ図）で定義していくと言ったものである．これをソースレベルまでブレークダウンして行くと，Ｃのソースを作り出すことができる．

エディタ上でのマンマシン・インタフェースについて特徴的なものではないが，２ボタンマウスを"対象"－左ボタン，"操作"－右ボタンとに使いわけ，その組合せによって支持をすると言った事をしている．

２．３．２．ソフトウェア・データ・ベースの考察

まだ構想の段階ではあるが，プログラムとモジュールを同時にスクロールさせて見える様にしたいと考えている．例えばＣのソース・コードに，あるファンクションがあった時，そこでＫＥＹを押すとその場所で呼び出される場所が自動的にスクロールしてくれる，と言ったようなｃｔａｇｓを作りたい．

その前にデータ自身の事をもっと整理してみる必要がある．データに関しては２つの事をやりたい．

・同じデータは何度も入れたくないし，いちいち転記もしたくない
・出来上がったもの同志の対応がとれているかどうかチェックする機構

また，データを関係付けすることで何が出来るのか解っていないので調べてみたい．

図，表，文字と言った異質のデータを，どうやってきれいに関係付けていくか，少し考えている．

従来型のＵＮＩＸはＫｅｙＢｏａｒｄの世界があって作られたものである．その後ビットマップ・ディスプレーやマウスが登場し，これは使いやすいと言うことで幾何図形，表，木構造データ，グラフ構造データが簡単に扱えるようになってきた．しかし，ＵＮＩＸはバイト・ストリームをベースとして作られているので，これらのデータは扱いにくい．

ＵＮＩＸのＡＷＫを使ったデータベースにフラットファイル・システムがある．すべてコマンドで成り立っているが，なぜデータベースかと言うと

・一貫性を保証している
　プリミティブへのデータ型チェックはＬａｔｅ・Ｂｉｎｄｉｎｇと言う方法（誤った操作の入力は許すがチェックイン時に排除する）をとっている
・データベース的操作が出来る

からである．

この発想で今考えている事を図示すると，図３－１のようになる．

ポイントとしては

・図形を文字として編集可能な形式にする
・型チェックはＬａｔｅ・Ｂｉｎｄｉｎｇ方式
・概念を高めたプリミティブ（データ構造毎の操作機能）の提供
・形態の一緒のもの（例：モジュール構造図もＨＣＰ図も木構造）はプリミティブも統一する

ＵＮＩＸで取り扱いにくい図形データをすべて文字データに変換し，ソフトウェア・データベースと言ったものが作れないか．利用者はパイプのメカニズムとプリミティブだけを知っていればＤ／Ｂ操作が出来るようにしたい．

ただ最終的姿での図形データ操作は，もっとグラフィカルな指示としたい．

以上．

３．討論

３．１．プレゼンテータへの質問

Q（藤野）：マトリクス構造について．１つのモジュールが，なぜあれほどたくさんあるファンクションと，交差するのか．この場合のモジュールとは何か．

A（福田）：リアルタイム・システムの例だからか．ただ，機能と１つのモジュールを１対１に対応させる事など出来ないと考えている．

Q（藤野）：テストの時，マトリクス表現によりテスト可能な個数が解ると言う話だが，モジュールを細かく分ける必要が生じた場合は，コンプレクスティ・エクスプロージョンを起こす事になるのではないか．

A（福田）：基本的に全ケースのテストは出来ないと思っているので，全体のテスト項目のマッピングを見てこれとこれを網羅すれば良い，と言うかたちのアプローチで考えている．（これが完全なテスト項目数という意味，実際は全ケースの１％位？）

Q（藤野）：仕様書エディタ（ＳＰＥＣ）の話について．あれはＳＰＥＣではなく殆どプログラムと同様の事をしている．モデル化してるときに，どうやってドキュメントを残すかを問題にしている．最近はドキュメントを，フラットに書くと言うことではなく，Ｊ－ＳＴＡＲ上のビューカード（ノートカード），いわゆるハイパーテキストのようなものにランダムメモリし，適当にリンク付けし，まとまった時にドキュメントしようとしている．

そんな意味から，メモを書いている時に表記が違うとか言われると，ヘジテートしてしまうがどうか．

A（堀川）：確かに，骨組みに併せてドキュメントするという事では設計出来ない．設計した情報を，きれいな形に

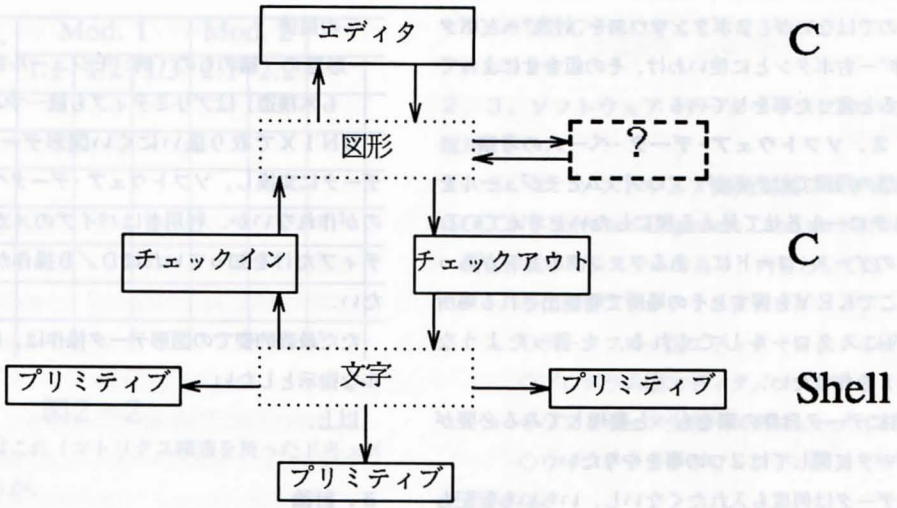


図3－1

まとめあげているに過ぎないと思う．

・設計する　Ｎｏｔ＝　設計の情報をきれいにまとめる（文書化）

設計の方法としてカード・ベースの方法は有効かと思う．嫌われるということについては，設計文書をまとめる段階で管理部門の決めたドキュメント体系があって，何も考えずに自然に文書が出来上がると言う"無駄の排除"に意味があると思う．

・強制への反発←→標準化としての意味

Q（市川）：リソースを管理するデータ・ベースに関して．別のソフト開発環境（M/C）での，実機を意識したテストはどう行なえうるか．

A（岡本）：先ほどの話で漏れていたデータ・ベースの構造について補足コメントすると，目的のデータ・ベース相互間には，それらのリレーションを取るためのマトリクス構造を持った格納場所のポインタ・テーブルを置いて，間接的にリンクさせている．

質問に関しては，作業終了後の情報入力段階で，バグやテスト・メンテナンスの状況や理由の記録が文章形式でＤ／Ｂ化され，その後の分析に使用している－と言った機能がテストと言うものに貢献している程度である．

またユニークな部分では，作業者と人事Ｄ／Ｂのリンケージがとられていることで，必要な要員の所在が常につかめるようになっている．すなわち，ドキュメントを残すと言うより，その作業者(の頭に入っている知識情報)を管理すると言う発想である．（笑い）

３．２．関連討論

佐藤：何をどのように貯えるかと言う点については，開発行程のうちの上流か下流かと言う話がありますが，過程そのもののドキュメントも残すか結果だけ残すか，と言う点についての意見はありますか．

熊谷：過程のメモやラフスケッチこそ，設計ＫｎｏｗＨｏｗだと思う．光ディスク等にスキャナーから直接読み込んだ情報を貯え，そのマンマシーン・インタフェースを確立させたらどうか．

佐原：ソフトウェア・データベースと似たようなシステムとして，アドベンチャゲームがある．そのような探す喜びを，ソフトウェア・データベースの利用者にも与える仕掛を考えてはどうか．

熊谷：今のＤ／Ｂは，最初に決めたスキーマに従ったＯｎｅＷａｙ機構であるが，人によってＫｎｏｗＨｏｗや，やり方が違うので，自分の希望に併せてカスタマイズ出来る機構（ＴｗｏＷａｙのコミニュケーション）が必要だと思う．

落水：ＳＤＡの中でこんな事をやりたいと考えている．(図3－2)

西岡：ユーザは，負担なくどのようにＤ／Ｂに情報を貯めて行けば良いのか．すぐ見える効果をユーザは求めるが，お考えは．

落水：開発側と利用者側の接点の部分で，協力して進めるしかない．大学側だけでは無力だ．

佐藤：では，実際にＤ／Ｂを構築されていて，経験から話

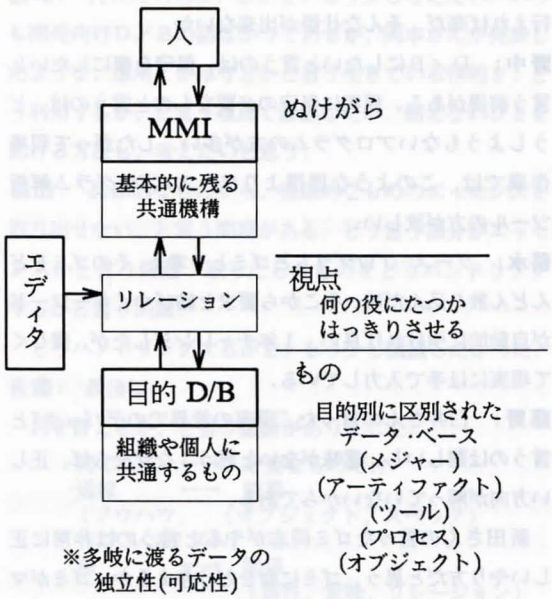


図3－2

していただける方．ＨＳＴではドキュメントＤ／Ｂグループで，専門にツール開発しながら進めていると言うことですが，いかがですか．

吉村：　チームを作って３ヶ月．要求定義からテストまでの，全てのドキュメントを貯えていこうと言う考えで進めてきたが，要求定義のところは未完．

基本設計から詳細設計に移る時に，詳細設計テンプレートと言うのを用意し，それに従って個人々入力している．Ｄ／ＢはＵＮＩＸのディレクトリ構造のまま．この内容を，全員がオンラインで見ることができるようになっている．またツールを通す事によって，テスト仕様書が出力される．この程度である．

佐藤：　ドキュメントや他のオブジェクト等，大規模に管理されている田中さんの方からいかがですか．

田中・正：　設計のドキュメントはいらんと言う話が進んでいるが，それはおかしい．今は紙で保持されているから使い物にならないだけ．どんな情報もＤ／Ｂに入れて，いろいろな方向から自由に検索できれば使える．コンピータとの対話で開発を進め，そのプロセスの結果として自然に情報が貯えられるようにすれば，入力負荷はかからない．

佐藤：　熊谷さん．前にマルチメディアの問題でしたか，イメージをＣ／Ｄ－ＲＯＭや光ディスクに入れてしまうと言うお話でしたが，検索の仕掛についてはどうお考えですか．

佐原：　イメージとテキストを別に管理すると言うのは，あまり良い方向ではない．イメージで持っているけどテキストの抽出も出来ると言うことが必要で，それが可能になってから，ソフトウェアＤ／Ｂに取り入れるべきだ．

今までの話しでは，ドキュメントとプログラムを離して考えられている．本来いちばん良いのは，プログラムの中にドキュメントが見えていることだ．例えばＷＡＶＥと言うシステム(ＰＡＳＣＡＬ)では，ＳＰＥＣを書いて行くとプログラムやドキュメントがジェネレートされてくる．こうならないと，ドキュメントとプログラムの関係は見にくい．

熊谷：　１人ずつの端末が与えられていない現在の状況では，紙で書いたものをスキャナでちょっと読ませた方が早い．ＫＥＹがコード系でないから面倒で駄目と言う話については，

・ペーパーとペーパーのリレーションを色分けで行なう

・イメージの一部を圧縮しＫＥＹとする

と言う方法が，かなり現実的な解だと思う．但しコード系でしか入らないものだけをコンピュータに入れて，ソフトウェアＤ／Ｂ云々と言うのは現実的ではないと思う．

佐藤：　インデックスとドキュメントとは別に管理し，インデックスは目で探すと言ったような発想は，ＳＲＡの田中さんも同じではないかと思いますが，いかがですか．

田中：　情報と言うもの全てを貯えなければ，ＳＷＤＢではないと言っているだけ．

落水：　佐原さんの，プログラムだけ入れておけば良い，と言うのは反対．作った物だけではなく，背景，原因，効果と言った情報が入っていないと駄目．

新田：　アドベンチャ・ゲームは，作成者の意図した方向に沿って検索して行くのでおもしろい．ソフトウェア・データベースはメンテナンスに役立たせるがために，作成者の方向と直角にばかり見ているからおもしろくない．

落水：　いや，ツールなのだから直角に見て効率を上げれば良い．

臼井：　何を入れようかと言う点については，必要なドキュメントは入れるべきである．但し現実には，本当にうまくドキュメント管理していると言う報告がなかなか無い．ただ電子化しようとしているだけである．

入れればなんとかなるでしょうでは，クズカゴの大きいのを作っているだけだと思う．

佐藤：　入れなければ何も出発しないと言う議論もあると思いますが．

臼井： 作るより現実の世界でデバックしてみるべきである．

佐原： ソース以外は入れないと言う事ではなく，メインのドキュメントとプログラムがあり，それに関連したメモ情報がＰｏｓｔ・Ｉｔ的に貼ついていて，それをたどると関連した所を見に行ける，と言った感じに考えている．これで，今までの環境より整理できるし，いろいろなものを入れようとすれば出来る．ただ，Ｐｏｓｔ・Ｉｔの部分に声とか絵とかを入れる仕掛は，現在のアーキテクチャでは限界がある．

岸田： これまでの，議論がかみ合わないのは，皆がいや１人の中でも違うことを言っているからだ．臼井さん流に言えば"構造化されたゴミ箱論"と言う事になる．それはＥＲＡモデルで日常の全ての行動を記述できるか，と言う質問に通じる．

- 作業と成果物の関係付けは，どういうモデルだったら出来るか（良いのか）
- そのモデルを，ユーザにどうプレゼンテーションするか
- コンピュータ・システムの上でどう具体化－インプリメンテーションするか

と言う話が，ゴチャゴチャになっている．

今われわれの持っているモデル化の手法で，大規模なソフトウェア開発が本当に出来るか．出来ると言う人もいるだろうが，私は半分以上は疑問ではないかと思っている．

近藤： 電子ファイルの現状についてお話する．

キーワードの付設負荷や，イメージとキーワードの分離と言った問題について，ある事例（新聞の切抜き）では記事をイメージで入れ，見出しを指示（マウス等でクリック）する事により，文字認識させコードでのＫＥＹ付設を行なっている．

電子ファイルが一般の世の中に広がらない（ユーザは興味を持っているが）のは，現状のドキュメントが整理されていないのをまずどうやって整理しようか，と言うところに話がいってしまうためだ．

電子ファイルとプログラム環境の結合についての，フレンドリーなマンマシン・インタフェースは，特化したシステムでまず立ち上がるのではないかと見ている．

新田： ソフトウェア・データベースがゴミ箱で，そこに入っているドキュメントをゴミとすると，ゴミとゴミとの関係を人間が繋げようとしているのが，いけないのではないか．そのために，大変である→実用かされないと言うことになる．ゴミ同志がキーワードの接続関係を認識して行えれば楽だ．そんな仕掛が出来ないか．

野中： Ｄ／Ｂにしたいと言うのは，保守を楽にしたいと言う前提がある．現実に保守の必要なものと言うのは，どうしようもないプログラムの方が多い．したがって現場作業では，このような機構よりソース・プログラム解析ツールの方が欲しい．

落水： ソース・プログラムとゴミとを考え，そのゴミをどんどん放り込んだ時，そこから要求と結びつくキーワードが自動的に引ければ良い．１年チャレンジしたが，難しくて現実には手で入力している．

藤野： 臼井さんの言った，"現実の世界でのデバック"と言うのは難しいし，意味がないと思う．なぜならば，正しい方向が解っていないからである．

新田さんの言ったゴミ同志がやると言うのは非常に正しいやり方だと思う．ゴミに命をどう与えるか，ゴミがマルチメディアになったらどうなるか，が今後の問題である．これに，まじめに取り組もうとしているのが熊谷さんの所ではないか．

なぜＴｏｓＦｉｌｅが売れないのか．文書だと，簡単にクリップでのまとめが出来る．その便利さが無い．紙の操作はリニアであり，検索の過程からの判断で，この次あたりには何があるかが解る．直すことも簡単である．即ち，現状の紙文書でやっている操作が同じように出来ないからである．

岡本： 定型的にまとまった世界と，ゴミばかり貯められる世界の両方を，１つのＤ／Ｂと見ないと駄目である．ゴミの世界がＫｅｙ．これを，構造的に拘束力の低いＤ／Ｂで繋ぐ事が，必要だと思う．

臼井： ゴミの山と言ったが，何がゴミか，はっきりしないのが問題である．この管理が現実の世界ではやりにくいので，Ｄ／Ｂ化すると言うのだと，これでは実現出来ない．機械は計算が早い検索が早い等，なにがしかの能力しか無い．その辺が楽になるだけで，ほかは何も楽にならない．それ以外のコンセプトが，現実の世界で出来ていなければ絶対に使えない．

北野： 現実にゴミをたくさん作っている．ユーザの要求に従って作ったが，後で使ってみると使いにくいからゴミと言うことになる．目的の違うものを，一緒のドキュメントに書くからゴミとなる．目的別にツリー状の整理が行なわれ，それが重ね合わされて１つの仕様書になるのである．

酒匂: 何のためのD／Bかを，もう少し考えたい．いつも開発向けD／Bの話ばかりであるが，岡本さんが発表したような，運用とか保守とかと言う生きている情報を，どう利用するか，と言う視点で論議したい．燃えないゴミを助ける方法も，考えたいと思う．

福田: 具体的なものから，抽象的なもののエッセンスを取り出せない，と言う問題がある．どう言う部分がエッセンスかと言う問題．取りだしたものをどうハンドリングするかと言う問題．

どうハンドリングするかを，もう少し議論したかった．

佐藤: 最後に．

何を貯えるか，と言う議論がありました．

- 何でも ←→ 必要なものだけ
- 過程 ←→ 結果
  （ノウハウ，アルゴリズム） （オブジェクト，スペック）
- 物 ←→ 関係
  （属性，意味，リレーション）
- フォーマル ←→ インフォーマル
  （イメージ，図形，音声，文章）

全体的にはちょっとはっきりしませんでしたが，対象システム全体の情報，実現する構成とそのモジュール間の機能，ソフトウェア開発の進捗・統計管理情報，等の検討が必要と言う事でした．

どこで，どういうモデルで，だれが作るか，と言うことも議論したかった訳ですが，時間が過ぎていますので終わりにします．

中村眞

自分の机の上でもそうであるが、物を「捨てる」ことのむずかしさ、忘れることの大切さと共にそのむずかしさを感じる．

加藤朗

はっきり言って、企業のマネージャー向きの話．地に足が着いていないので、面白くない．もっと、具体的な話がいい．

小林貞幸

ソフトウェアデータベースは、電子化を抜きに語れないと思うが、福田さんのマトリックス構想は、非常に共感を覚えたので、是非、電子化について、突っ込んだ方法論を展開したい．

小池幸徳

福田さんの言われたマトリックスは、大変興味がありますが、複雑化し過ぎて、実用面でどの程度期待できるのでしょうか？

市川寛

十分なカルチャーショックを得た．

桜井麻里

福田氏(NES) のマトリクス体系を持ったドキュメントというのは、面白いアイディアだと思います．

問題は、分かりやすい表(マトリクス)にするためには、書く人のスキルが要求されるということである．めんどうだという意見もあるだろうし、挫折してしまう危険は多分にある．でも、頑張ってほしい．

岡本さんの話もよかった．運用面での悩みを解決するべく、努力されてきたのだなあ、と．

田中一夫

KCS 岡本氏

現在、我々がやろうとしている事を 7,8 年前に実施していたとは、ショックであった．但し、具体的資料が見たかった．

NEC 福田氏

s/w DB とはいうものの設計手法的な内容であった．が、詳細設計以降は使えるが、要求定義では使えないと思う．

臼井義美

単なるドキュメントデータベースは、巨大な電子ゴミ箱となる．関連事項の検索方法、変更に対する矛盾の発生防止、削除の規則、管理方法の確立などが必要．

中野秀男

(1) 大学における SWDB：そんなに大規模なものはできないので、既存の DB を使ってやろうかな．

(理由) 皆が簡単に使えるように．LAN と WS があるが、それを使いたいし、Informix もあるので．

(2) プログラム開発(WS based)：いい Tool があれば使いたい．そこまでやっている暇がないので．

(3) 企業における SWDB：(okamoto@kcs) 発表者の言う事は良く分かるが、実際には、そのような環境にいないので、コメントできません．

(4) マトリックス：関数の階層化：実際には、階層化が、tree になっていなくてややこしいと言う意見だが、私は、構造化された関数 (木構造) を小物関数の形で実現していて、うまくいっていると思う．小物関数のグループを木構造の対応する部分木の root にはりつければ、もっとうまくいくなあと今日思いました．

(5) 図形データベース中身：ファイルの中身が知りたい．

西岡健自

(1) ディクショナリと環境ジェネレーションの関係が分からない．

どのような仕組みで多様な特性を備えている筈の環境が生成できるのか．

データベースへの吸収時の取捨選択はどういう基準か？

(2) 前に品質機能展開表を作ったが、ついに、A1 サイズに戻って、書くだけで終わり、未だに何が書いてあるのか分からない．

酒匂寛

岡本さん：開発以外の運用、保守フェーズに光を当てる努力に感心しました．もう少し具体的なお話が聞きたいと思います．

福田さん： requirement --- module マトリクスのアイディアは、よくある考えだが、階層の表現、表の大規模な変更 (保守) は、どうすれば良いだろう？？

例えば (たとえ話)、マトリクスからは、モジュール·テストのチェック表には使えるが･･･システムテストに対して

は･･･??

盛田政敏

マトリックス構造（３次元）を持ったドキュメント構造の話、大変おもしろいと思います.
現在、ドキュメントの DB 化を考えていますが、これの論理的な一つの切り口として、私たちも考えてみたいと思います.
岡田さん、頑張って下さい.

伊野誠

(1) 岡本さんの発表は、もう少し、焦点を絞られると良かったと思う.
管理用のデータベースの効用は良く分かるが、実現する際の苦労話や、データベースの大きさ、データベースそのものの管理について、話があるとよかった.
(2) 福田さんの話は、ソフトウェア･データベースとの関わりが、よく分からなかった.
(3) 堀川さんの話は、大変興味深かった.（具体的なツールの話として）内容的にもレベルの高いものと思う.

三浦あさ子

「コンピュータ」をうまく運用するという目的から始まって、開発環境のデータベースができた、というところに納得した.
ソフトウェアデータベースって何かしら？もし、自分がそれを使うとしたら、どんなものがほしいのだろうか？

新田

単なる結果の DB 化よりも、なぜそうしたのかという過程の DB 化が大切と思う.
マトリックスドキュメントは、すぐにでも実験してみたい.

鶴見泰久

NEC 福田さん、言語は何ですか？

寺井孝

データベース化する際、「環境全体」を考える必要があるのではないかとする岡本氏の意見には全く同感である.

井川裕基

KCS：ホスト中心の、という事だが、一つのマシンだけで全てをまかなえるのだろうか？開発と実行は同環境？

荻生準一

福田氏の＂マトリクス構造を持ったドキュメント体系＂に賛同しました.
大規模システムになると、そのメンテナンスが常に問題となります. マトリクス構造のドキュメントにて各モジュールの関係付けがドキュメント化されていれば、メンテナンスに役に立つと思います.（設計時の検討資料にもなる？）

小林明彦

岡本氏の「開発環境の generate」は、大変興味を持った.
少々、対象は異なると思われるが、ワークステーションの環境を set 又は、改変する際に使用できると思われるが、環境のモニタリング、その他、情報の D/B 更新問題で、当方はやや悩みがあります.
福田氏のマトリックスの考え方、即に･･･ということはないでしょうが、「電子化は」実現性のあるものだと思いました.

深瀬弘恭

福田さん: マトリックスによるドキュメント管理は、可能性があるように思えた. エンジニアは、ドキュメント書きには、興味を示さないのが一般的である.
多段マトリックスが大規模開発では必要となると思われるが、その「ストリックス間の関連」の記述が問題となると思う.

井上尚司

３つともソフトやドキュメントをデータベース化するソフトウェアについての発表であったが、政治的、思想的の点から，ソフトウェアやドキュメントをデータベース化することについての意見も欲しかった. データベース化した後の運用とか･･･.

村川貴広

･一口に DBMS を作るといっても大変な時間と労力を要するようだが、なんとか、この労力を減らすうまい方法はないものだろうか？
･我が社では、MAP (Matrix Arange Plan)というマトリックスを使った行程管理を行っていますが、福田氏のマ

トリックスを用いたドキュメント体系も手法として似ています.
私はまだ、この MAP について良く知らないのですが、帰って、このマトリックスを用いた、いろいろな応用を考えてみたいと思いました.

堀川博史

岡本さん：イメージがつかめなかった.
福田さん：おもしろい.

近藤康二

自分として、あまり大規模な Application の作成経験がないため、岡本さん、福田さんの話は、実感はなかった.
確かに、耳学問的には、重要であると思われる.
最後の、堀川さんの話はとても興味がある. できたら（完成したら）使ってみたい.
自分でも document を書く時に苦労しているので・・・.

森幸一

(2-1) ・かなり現実的なアプローチで興味深い. ・開発プロジェクトの管理単位の最小は、プロジェクトなのか？プロジェクト内の情報の切り出しは実現可能か. %DB の利用は、保守担当も含むか？
(2-2) ・テスト項目洗い出しで module 単体レベルでは、かなり有効と思われる. ・現実のテストは、機能間の相互作用を無視できない.
多段マトリックスで解決可能か？
(2-3) ・仕様書エディタで、用語の同一性違反チェックは、どんなメカニズムで行っているのか.
・仕様書エディタの利用価値、十分にあると思う.

塚田道明

自分自身、３万本のプログラム資産管理ができておらず、今後、データベース化を進めなければ、と考えていますので、大変参考になり、興味深く聞く事ができました.

鐘友良

要求仕様か、設計仕様をできるだけ形式的に、(要求仕様は外部から見ると、非形式でもいいです) 記述する必要がある.

田中正則

岡本氏の話の中の、開発時ドキュメントは、保守用ドキュメントにはならないという意見に賛成です.

歌代和正

福田氏：ううむ・・・結論があああなるとは・・・. マトリクスにすると、
項目が、ｎx (x=1,2,・・・) のオーダーで増えちゃうので、オーダーを減らす工夫をしないと大変じゃないかなあ・・・.

田中慎一郎

ちょっと忙しくて聞けなかった. 討論に期待します.

北野義明

福田さんのプレゼンテーションの内容は、あまりに身近であり、仕様等の情報をマトリックスで表現しようというのは、通常そういう意識なく、そのような思考を行っている筈である.
普段、思考の中で何が行われているのかという事を整理して、目に見える形で、ある程度、標準的にとらえてゆくという事は、大切で、道を開く第一歩だと思う.

吉村智香子

(2-1) 具体的な話で、非常に興味深かった. リソース管理が不充分なソフトウェアデータベースは、ソフトウェアデータベースといえるのか.
(2-2) とても楽しく聞く事ができた. 多段マトリクス化、三次元マトリクス化などの理論をもっと考え、電子化にとりくむと、なかなかおもしろいと思う.
(2-3) まず、プロトタイプとして、図形データを文字で表現するという試みは、今、考えられる方法の一つであろう. UNIX を考えると、やはりバイトストリームを考えざるをえないから---.

野中哲

ドキュメントのマトリウス化は、おもしろいと思った.
ポインタの考えをいれて、重複した記述をなるべく減らす事ができると良いのではないか.
保守も楽になると思う.

藤野晃延

岡本(KCS)： J-Star で作った OHP は美しい！また、実際に DB の中味（情報）の入力に５、６年費やしたという

のは、正しい選択だろう.「入れ物」と「中味」を同時に構築・更新していくのが必要だと再認識をした.
福田: Matrix として、Func x module としていたが、例えば、Mutti-Function を持つ一つの Module という設計は、それ自身が余り良い設計ではない(?)のではないか? Object 指向では、Function が非常に単純な Object で、Mapping されている為、組み合わせテスト等は、Conplexity-Explosion を起こしてしまう.
また、Test で、完全性を証明する事もできないと思う. 併し、「現時点では、実用的な 1 step」なのかも知れない. 試されるべきかもしれない.
堀川 : 仕様書エディタで記述する内容は、何か殆ど「How」を記述させるように Guide しているような気がするが、Doc を作っているときは、What を試行錯誤している段階が多く、メモ程度を入力して、考えが纏まったら、初めて、Doc の形式になるのだと思う. ちょっと、Doc の意味合いが違うようだ. ソフトを工業製品の様に大量生産するために必要な「悪」なのかも知れない.

熊谷章
福田 : 誰が、matrix を作り、使用者は誰か. マネジャなのか作業者なのか. 機械とモジュールの table ----- 書くのが大変そう!
堀川 : check in/out がミソのようであるが、「グラフを文字に変えた必然性」が余りよくわからなかった.

久保宏志
(1) いわゆる DP 一般の DB と、ソフトウェア開発という特殊アプリケーションのための DB (=ソフトウェア DB) を分けて議論することにしないと、フォーカスがぼける.
(2) おそらく、DB テクノロジーの中で、もっとも遅れているのは、ソフトウェア DB のテクノロジーであろう. この仮説を確認する事をやってはどうか. 遅れざるをえない事情、つまり、解かねばならない問題のむずかしさをきちんと理解しておくのは、有効そうである.
その上、それぞれの問題へのソルーションサーチの state-of-arts を確認する. もう一つ、state-of-praction を知っておくことも software DB の position を確認するので有効である.
「[CAD/CAM] -{生産管理と一体]--[販売管理・物流管理と一体]」の順で、DB テクノロジーの practice を追録するのに、一つのアプローチでありうる.

???????
岡本氏 : 開発計画から環境をジェネレートするということですが、予定の遅れ、進みなど予定外のイベントをどう救いますか?

鐘友良

ソフトウェア DB を議論するときに、
(1) 現存的なソフトウェアの管理
(2) これから新パラダイム形態ソフトウェア開発手法に関連して考えた方がいいじゃないかなと思っています.

森幸一

・データベースの利用は、作成した人が再利用ではなく、他人、他部門が有効に再利用できるものであるべきだ.
・イメージの検索の可能性をもう少し追求したい.
・情報のデータベースへの入力が、責任部門で行えるシステム構築が必要.

落水浩一郎

大変参考になった.
ゴミ key 付きゴミ自動クラスタ化はよかった

桜井麻里

何についてはなしをしたかったのだろうか. 根底にあるものが見えてこない人には、辛かったでしょうね. 私は、楽しんでいましたけれど.

堀川博史

ゴミの話がよかった. (共感)

伊野誠

(1) この問題は、いろいろな切り口からの見方で意見が違ってくる. 午前中の討論は、その切り口をはっきりしないで討論したので、発散したきらいがある.

村川貴広

・なかなか討論が白熱していますが、自分が参加できないのがくやしいですね. やはり、常日頃、いろいろなことを発展的に考え、実行し、反省するという過程を踏んでいないといけないなと感じました.
・DB 化として、ドキュメントの電子化について討論されていますが、近藤さんの話にもあったように、ユーザは、いま目の前にあるドキュメントをどう整理していくかということが案外重要ではないでしょうか?
電子化という、インプリメントのところの話だけでなく、現実世界のモデル化というか、どのようにドキュメントを蓄え、整理していくのか、その実際例をもう少し討論できたらよかったと思う.
ちなみに、私は、現在、物事の整理に、「KJ 法」を用いています.

鶴見泰久

ソフトウェアデータベースにおけるコンセプトが、開発/保守の立場が違いすぎる.
ソフトデータベースを作る事において、時間差を考えたデータ構造が必要である.

加藤朗

佐藤さんの presentation を「最初に」していただければ、もっと良いまとまりが得られたような気がします.

田中慎一郎

・設計するという事、設計をきれいにまとめあげるという事は、私も別ものだと思う.
だから、設計の output を分析しても設計 process は、明らかにならないと思うのだが・・・.
(今回の話とは関係ありません. すみません.)
・ソフトウェア DB は、何の為のものなのか大事.
・メンテ用か
・再利用か
・教育 (学習) か --- 私はこれに使えたら嬉しい.
が、現実はまず、メンテ用でしょう.
これをはっきりさせないと、もともとむずかしいものがもっとむずかしくなると思う.
これは、設計用のドキュメントが、メンテの役に立たないのと結局は同じことになるのだろうと思う.
データが入っていればいいというものではない. ここにもユーザインターフェイスの話が出てくると思う.

盛田政敏

もう少し、実践的に、どうあれば、又、どの様に、データベースを構築し、使うのか? といったアプローチについて、議論したかった.
「ゴミを入れてもしようがない」と言うだけでは答えにならない. それでは、何をどう入れるのか? もう少し考えたい.

塚田道明

(1) 開発プロジェクト員間の情報伝達を目的とした D/B

必要な検索キーは無数にある．
書き上がった資料を全て D/B 化する必要はない．標準的なもののみで良い？
必要な情報が一意にできないから、とりあえず、登録するという発想が問題ではないか．
(2) メンテナンスを目的とした D/B
・システム構成、思想、方式を記述したドキュメントとソースリスト及び、I/O イメージが分かれば十分？
・設計段階で出力されたメモこそ、整理されてイメージ D/B として、参照可能にしておくべき．
・成果プログラムの持つ属性が、クロスに見る事ができる D/B は必要．・・・不要プログラムの整理のためにも．
(3) 納品物としてのドキュメントが多すぎるのではないか．
・活字で出すために電子化 D/B 化が本当に必要か？

岸田孝一

(1) ソフトウェア開発プロセスとそこから出てくるプロダクト（もちろん中間のメモに近いドキュメントを含む）の関係を表すモデルをきちんとしなければ、議論がまとまらないような気がする．
Entity-Relationship-Attribute で行けるか??
(2) 分数環境での Software Database は？
MMC の gIBIS のように、電子掲示坂にある構造を持ち込めば良いのか？

近藤康二

・カード型データベースからのドキュメントの作成、スケルトン型のデータベースからのドキュメントの作成
・電子 File
・ゴミの世界
とくにありません．

市川寛

・やはり、ソフトウェア DB 管理マシンがいるのだろうか？
・既にあるシステムのデータベースをどう作り上げていくか．

藤野晃延

・再利用を支援できなければ、SWDB ではない．本当か？
・検索のキーは何か．又、そのキーはどうユーザに specify させるか．
・「入れ物」としての機能は？
・「内容」は、なにを入れるか？
Media は変わってくる、その影響は？
・兎に角、いろいろヤル事が大切！

中野秀男

(1) 後半盛り上がって良かった．
(2) ゴミが勝手に集まる idea は、面白い．
(3) 帰ってゆっくり考えよう．

野中哲

ソフトウェアのデータベース化というタイトルからは、ソースコードの部品化とか、再利用とかの話が中心になるのかと思っていたが、そうではなくて良かった．
データベースは、個人でカストマイズできるべきだという話に賛成．

三浦あさ子

討論の的を絞ることに賛成．
ソフトウェアデータベースは、「何のため」に作られるのか．
「作る」が良く分からなくなってしまった．

新田

ソフトウェア DB の話は、すごく重い．

西岡健自

ソフトウェアデータベース
あらかじめ入れておくもの開発途上に蓄積されるもの開発後に蓄積されるもの
各々分けて考えるべきです．それに意味を持たせるのは、ソフトウェア開発における効果と、入力コストとのかねあいで、後者の負ける場合．
こういう観点から入力コストの意義大．蓄積後のヴィジョン必要．
データベース化することの生産性向上に対するヴィジョンあいまい．

田中正則

SWDB（ソフトウェアデータベース）
・入れる情報は何でもかんでもすべて入れる．これなくし

ては、SWDB ではない.
・SWDB では、検索が生命だと思う. 自由度が高く、レスポンスも速い検索機能. 熊谷さんが言った、イメージ化したインデックス方式が、有効だと思う.

小池幸徳

ソフトウェア DB は、何のために必要かと考えると、ソフトウェアの開発面、運用面によりそれに対する価値が異なるのではないかと思う.
最初にシステムを設計するときには、むしろ過去のシステム構築事例(ソフトウェア DB)に頼るよりは、SE 、デザイナーが独自に設計を始めるように思う.
ということは、その時に要求されるのは、プログラム開発段階において、ソフトウェア DB によりシステム全体の整合性を保証することではないかと思う.
しかるに、ソフトウェア DB というよりプログラム%ディクショナリー、システム%ディクショナリーというものと、データ%ディクショナリーとの関わりについて、もっと話し合う方がいいと思う.

田中一夫

現在の H/W 、S/W では、SWDB 実現は無理?!

酒匂寛

何が、object か?
何が、Relation か?
どのようにそれらを操作するのか?

井川裕基

自分の仕事場でもドキュメントを電子化、DB 化したいと思っているけれど、まだちょっと大変なようですね.

荻生準一

ソフトウェアデータベースを論じるにあたって、" 新しいメディア(光ディスク)を使用して、保存する"又、"どの様に検索する" という議論以前に、ソフト開発にあたり、ある作業をする時に、どの様な情報(データベース)を必要とするのかを検討すべきと思う.

岸田孝一

ゴミ論議は面白かった.

海尻賢二

私も定型的な仕様化と、プログラミングを統合した環境を試作しましたが、結局、仕様が生かせず、失敗でした.
やはり、仕様(又は、document)は動的で、static に定型的に作ってもダメだと思います.

小林貞幸

ユーザニーズをまとめて、最終的にシステム稼働までのステップを一つの作業としてとらえると、ソフトウェアデータベースの必要性は、あくまでも生産性(より早く、より正しく、より楽しく、より美しく)から出発しているはずであり、「かくあるべき」(存在)論を抜きにして、今回は、方法論だけが論じられてしまった感がある.

中村眞

ラフスケッチの重要性を感じ、いかに、DB 化するかを再認識した.

井上尚司

ゴミ(ゴキブリ)の入力方法ゴキブリホイホイの中での欲しいゴキブリの見つけ方のきわめつけが見つからないと、机上の空論になってしまいそう.

林香

従来のドキュメントの DB 化には、夢がない. 後ろ向きだからか.
マルチ%メディアの方向で考えていくと、夢があって面白い.
目的が不明確で、実行が大変なことは、つまらない. SDB は、そんなもの.

北野義明

SWDB 開発設計テスト
　運用保守次元-1 各々について
　業務ハードユーザ個別の要件レスポンススピード操作性 etc.
次元-2 各々上流下流
次元-3 何次元もの座標で定義情報

熊谷章

ゴミの話がおもしろい.
非マシン現在のマシン電子ファイルを入り込んだマシン

SDB
なかなかロジカルな情報
評価する人が、DB Manager だから、一意的になってしまう.
有効内容は、ノウハウ(設計書、プログラム、その他)ゴミ
大勢の人がいると、ゴミは、ゴミでなくなる.
したがって、SDB のキーは、自分で自由に操作できる flexible な DB が必要となる. このとき、relation の書き方と操作が重要だ.

久保宏志
・ソフトウェア%プロセス一般における人と software DB のからみを問題にする前に、もっと易しい問題で予習するのが利口そうである.
・易しい問題として何を選んだらよいのであろうか. 例えば、
・情報システム全体の Hardware/Software Confifuretion の lutegrity を maintain するプロセス.
・税制改革を稼動中システムに反映させるプロセス.
・Design Phase の終了時点で代表的なトランザクション処理を実行するのに必要な、path を推定してみる、あるいは、目標とする path length に近づけるために、Design を改良するプロセス.
・再利用部品を探索するプロセス.
・パッケージプロダクトにたいするカストマゼーション仕様を導くプロセス.
・研究方法は、現実に遂行されたプロセス事例を題材にするのが良い.

『ソフトウェア･データベースの試み』 サマリー

長野県協同電算　　佐藤千明

ソフトウェア･データベースという古そうで新しいテーマに関して、実践的見地から理路整然と(?)討論するつもりで意気込んでいったところ、見事に打ち取られてカオスの世界へ入ってしまった。事前のプログラム委員長との打合せで『今までのワークショップでChair.が取りしきった例はない』と言われていたので覚悟はしていたが... 私が描いていた筋書きは

1. 討論にあたっての論点の洗い出し(5W2H)
2. プレゼンテーション
3. 論点の分析整理
4. 論点ごとの討論
5. 言いたい放題

そして、出されたポジションペーパをもとに以下の論点を用意しておいた。

(1) WHY ……何のためにSWDBを構築するのか

管理、保守、環境構築･提供、開発支援、文書化、再利用、検索、教育･技術移転、標準化

(2) WHAT ……何を蓄えるか

方針: とにかく詰め込め(電子化が先)　**VS.** 是非管理したい情報(見返りを考えろ)

内容: 対象システム　の分析結果、の振る舞い、を実現させるソフトウェアの機能
ソフトウェアの　構成、振る舞い、ロジック、インターフェイス、性能
個々の部品の　機能、ロジック、インターフェイス、関係、テストデータ
進捗管理情報、統計管理情報、リソース管理情報

属性: オブジェクト、スペック、成果物、結論　**VS.** ノウハウ、アルゴリズム、概念、判断、過程、メモ。
実体、プログラム、データ　**VS.** 属性、意味、関係、スペック、メタデータ。
formal、スキーマ定義、言語表現、コード、テーブル　**VS.** informal、イメージ、絵、音声、図形、文章(文字の集まり)。

(3) HOW ……どうやって蓄えるか、捨てるか

どんなモデルか　階層型、ネットワーク型、リレーショナル型、ERA型、知識ベース、ドキュメントベース、オブジェクト指向DB、抽象データ型、HyperText、ポインター型、マトリックス構造、ODA、ODIF、自己増殖可能モデル、概念モデル、外部モデル、内部モデル

どうインプリメントするか、
どう捨てるか　ガーベッジ･コレクション

(4) HOW ……どうやって検索するか (検索支援)

ユーザ･インターフェイスの改良方法

日本語のQuery･Language、ビジュアル･インターフェイス、HyperCard的

WYSIWYGは本当に使いやすいか、速さが解決したとしたら?

アドベンチャーゲーム的探索　　知的検索、AI指向
一定の操作モデルに基づくガイド、シナリオ　　キーワードの設定、分類基準
人により異なるインターフェイス･視点･Viewの提供　　コマンド、ライブラリの組合せ

(5) WHEN ……いつ蓄えるか、いつ捨てるか

① 開発プロセス＝＝SWDBとのアクセス……物を創りだす

Zodiac、Solon、Spiderモデル

メインフレーマの一貫支援環境

SEA/I、SDAS、EAGLE2,TSX-1100,ADSSBX

データ中心設計　　DD/DS

② 開発プロセスと並行して……プロセス･アイデア･リソース･環境の管理

文章、図形、イメージ等のドキュメント

概念、アイデア、判断の履歴、環境管理情報

③ 保守用に後から……物の管理

現行資産をどう移行するか、ソースプログラムとの整合性確保方法は?
手入力の労力、信頼性は?
自動連動の仕掛け、サイクルは?

(6) WHERE ……どこに蓄えるか

① 一元管理 ON ホスト
他に比べて構築、管理は容易
ホスト上で動くプロダクトの半自動生成に結び付けやすい
ホスト上で動くプロダクトの管理データを取り込みやすい
TSSではビジュアル表現が難しい

② 水平分散管理 ON EWS
一般的にユーザ・インターフェイスは良い
全体を管理するスーパー・ファイル・サーバは必要ないか
ネットワーク、分散ファイル・システムが必要

③ 垂直分散管理 BY MML
サーバ＝ホストコンピュータ　　クライアント＝ワークステーション
ホスト＋EWS＋PC も考えられる
どう機能分散させるか、繋ぐか、同期をとるか
メインフレーマの協力が必要だが...

④ ドキュメント専用マシーン
JStar (ネットワークあり)
ワープロ (ネットワークなし)
光ディスク、CD-ROM (加工困難、ゴミ箱がわり?)

(7) WHO ……誰が誰のために蓄えるか

① 個人が個人のために …… Mac、Personal、Local
② 個人がチーム、組織のために …… 一般的
③ 専任者がチーム、組織のために …… ドキュメント・チーム、専任パンチャー
④ 専任者が個人のために …… こういう身分になってみたいが

(8) 再利用、部品化

部品登録の判定基準、部品の評価基準
パブリック・データベース　　Sigmaでの考え方は?
再利用促進方法　　利用後のフィードバック情報

(9) 版管理システムの有効性

履歴管理にとどめるか、実行環境の再現まで可能とするか・できるか

(10) SWDBはどこまで必要なのか

ツールとの関係は　　構築にかかる手間と効果はどうか

(11) マルチ・メディア化

以上の論点についてきれいに話題を移そう、などという浅はかな考えをもって椅子にすわったのがまちがいであった。混乱の原因はChair.にあるが、入口でのWHYの議論が空振りとなり、田中姓の人まちがいがきっかけでペースが狂い、ゴミ箱論で極みに達したようだ。
まずはプレゼンテータの主張を整理してみる。

I　神戸コンピューターサービス 岡本さん

- 7年前から運用の効率化、リソース管理と情報の集約化の必要に迫られてシステム・ディクショナリの構築・整備をしてきており、それの延長でSWDBを考えたい。即ち、リソース管理⇒ツールの統合化⇒全プロセス支援⇒環境全体の管理、操作というアプローチである。
- その効果はリソース関係管理、開発に必要な環境の提供、不要SWの破棄、運用の効率化。

- SWDBは単に開発部門のみの支援にあらず。運用･保守部門まで含めてとらえ、リソース管理、開発環境管理、プロセス･モニタリングを行う環境管理DBとして考えるべきである。
- 環境ジェネレータがその環境管理DBを参考に開発者に必要な環境･情報を提供してくれる。
- 環境管理DBにはリソース(メモリ、ディスク、チャネル)、稼働状況、環境マップ、標準開発プロセスのひな型、プロジェクト管理情報等が入っている。
- DB構造はネットワーク型とマトリックス型(格納されている位置を押さえている)の併用
- ホスト集中管理から分散環境へと展開したいが、それにはSWDBを階層的に構築し基礎レイヤでネットワークを吸収する必要がある。今後の課題。
- » もう少し具体例が欲しかったが、確かに環境DBという捉え方は盲点だったかもしれない。

II　日本電気ソフトウェア 福田さん

- 今のSW設計書は保守の役にたたない。現に誰も盗もうとしないではないか!
- 機能とモジュールを軸にしたマトリックス構造を持ったドキュメント管理向きDBを作ろう。
- このドキュメントにより機能の実現方法とモジュールの役割が明確となり、さらには、テスト項目の洗い出し、網羅性チェック、怪しいモジュールの推定等テスト支援もできる。ただし、テストケースは膨大になるので全部の組合せを行うのは不可能。
- トップダウン設計を支援するには多段階マトリックスという概念が必要になる。
- 電子化したい。各セルで文書ファイル名を管理し、文書内容はキーワード検索可能とする。
- 機能、モジュールにデータ/テーブル/ファイル等の要素を加えた三次元マトリックスへと発展させたい。
- » 保守用ドキュメントから出発しているが、インプリされれば設計支援、テスト支援にも有効となろう。ただこのモデルでモジュール間の関係をきれいに表現できるのでしょうか。

III　三菱電機 堀川さん

- 設計工程支援として分散型ソフトウェア開発支援システムSolonを開発。
- まずは編集系を中心とした各種ツール群をEWS上に開発。
  - 仕様書エディタ(Spec):仕様書のスケルトン生成⇒日本語エディタで記述⇒内容の検証分析。これは設計作業の本質ではなく、後工程(文書化)の省力化、標準化をねらっている。
  - データフロー図エディタ(Spider):
  - プログラム図エディタ/コンパイラ(Pec):モジュールツリー、HCPエディタ、Cソース生成。
- 次にSWDB構築を考えている。
  - ソースとドキュメント(モジュールトリー、仕様書、HCP図、クロス･リファレンス)とテスト項目･テストデータを関係付けし、同時に見れる仕掛けをつくりたい。
  - そのためには、まず個々のデータの自動転記機能、一貫性チェックの仕掛けをつくると同時に、文字/表/図形という異質データの関係付けから始めたい。即ち.
- 元来がバイトストリーム･オリエンテッドなUNIXで図形/グラフ/表等のマルチメディアを扱うために“フラット型データベース”の考え方を導入する。一旦全データを文字に変換し(CHKOUT)、その文字データベースにアクセスするプリミティブ･コマンドをいくつか用意してバイトストリームでつなげていき、概念を高めたプリミティブを構築したい。
- » これからは文字だけでなく図形ハンドリングが大きなキーポイントになる気がします。“フラット型データベース”という考え方を文字、図形混在データベースの実現に取り入れるのは現実的なアプローチなのですか。

以上迄で11時となり、討論時間はあと1時間しか残されていない。これではChair.の論点分析整理時間は取れない。ならば言いたい放題から始めて、後で整理しようと方針変更。(これが甘かった!)

討論で出たテーマ、意見はおおよそ次のとおり。☆は意見の分かれたもの。

<目的>

- リソース管理、環境管理といった運用面を支援する環境データベース
- 設計書、保守用資料等の検索を支援するドキュメント･データベース

<中身>

- ☆ 単に設計文書だけでなく、開発プロセスの全情報が互いに関係を持って蓄えられなければならない。しかもその入力は特別な負荷なく、開発時に自然と入るのがいい
- できあがったソース･コードだけでなく.その背景、履歴、原因、効果も必要だ
- 設計時にはカードベースのメモ(設計の本質がある)作成支援と、それをきれいにまとめ上げる文書化ツール･仕様書エディタの両方が必要だ

- ラフスケッチ等のノウハウはスキーマ表現では難しいのでそのままイメージとして光ディスクに入れたら。その際検索のためのキイワード作成、インデックス作成に工夫がいる。即ち
  ①コード系で別に入力②イメージの一部をパターン化してキイとする③イメージの一部を文字認識してコード系のキイとする
- ☆ しかし、単に電子化しただけではそれはゴミ箱でしかなく、混乱の電子化にすぎない。まずは現実世界でのモデル化、ドキュメント整理が先。今これができていないのに電子化しても意味がない。(TOSファイルが何故普及しないのかと関係がありそう)
- ☆ いや、ゴミでも整理すれば役にたつものが見つかる。いかに命を与えるかを探ろう
- 頭のいいゴミは自分でデータに進化したり、仲間をみつけて燃えてくれるかもしれない

＜モデル＞

- プログラムとドキュメント(文字、図、表、絵)は一緒に管理し、同時に参照したい
- SWDB用モデルはひとつのスキーマで固定して考えないほうがいい
- MMI層(共通機構)－REL層(複数モデル化実現)－OBJ層(単なる物の集まり)に分け、使用目的にあわせてREL層を記述するのはどうか
- 今のモデル化手法で現実のソフトウェア開発を本当に記述できるのか
- 現実世界を概念的に表現する概念モデル、それをインプリメントする内部モデル、それをユーザにプレゼンテーションする外部モデルを別々に考えよ

＜ユーザ･インターフェイス＞

- アドベンチャーゲームのように探す喜びを与える仕掛けが必要
- 人ごとに異なるインターフェイス、視点を提供できるような仕掛けが必要
- 単なるオブジェクトの集まりでも、切り出し方によって異なる価値が出てくるようにしたい
- 検索時の視点を作成者と同じ(水平)方向にしたい (アドベンチャーゲーム的な疑似体験)
- ☆ 但し、保守時には垂直方向でもいいではないか。その方が実用的だ

『設計工程のドキュメント蓄積はSWDBの一種とも考えられるが、型にはまったワークシートへの電子的記入は不要ではないまでも設計作業の本質ではなく、過程･メモ等の情報の方が貴重でありそれを蓄積すべきだ。SWDBは幅広い概念を持っており単なるドキュメント管理、データ･ディクショナリ、リソース管理としてではなく全プロセス支援の環境として考えるべきだ(かなり難しいが)。ユーザ･インターフェイスは人によって可変とすべきだ。何のモデルもなくただ闇雲に詰め込んでもだめだ。』という考え方は大方の意見の一致をみた。

最後は、コード系からマルチメディア系データベースへの展開及びそのためには光ディスク、HyperTextなどの仕掛けが必要だという理想論と、いやまずは現実世界を整理したうえで必要な情報をきれいにモデル化して入力すべきだという現実論の対決であった。ゴミ箱は役に立たないので入れるだけ無駄と考えるか(SWDBゴミ箱論)、ゴミとして入れておいてもいつか昇華させてその有効活用を探ろう、とするかの違いでありともに一理ある。不十分なモデルのまま電子化しても役に立たないという完璧主義か、部分的な効果でも良しとするかの違いであり、ベクトルの力は違っても方向は同じ気がする。　なお蛇足ながらここでいう『ゴミ箱』には二つの意味があったようだ。即ち、スキーマ定義できないゴミ(必然的にイメージのまま蓄えられる)と、仮にスキーマ定義できても役にたたないゴミ(モデル化の手法がアヤシイはずだから)である。

あれはいいこれはダメという議論の背景には『どの場合には』という前提があるはずで、そこをお互いに認め合わないと『JStarかUNIXか』『文字かイメージか』『メインフレームかEWSか』という平行線に終始してしまう。これからの環境を考えたとき両方の存在を無視できないので、両者をうまく結び付けてみんなが楽しくなる仕掛けを作る必要があろう。

裏に隠された高尚な理論(?)に基づいた手軽に使えるツール(例えばJStar,HyperText,4GL,RDB,光ディスク等)を必要に応じて使い分け、結合していくのはどうだろうか。その資産の蓄積がいつか整理されてSWDBになるかもしれない。モデル化がいい加減なまま電子化しても十分な効果は得られないだろうが、電子化作業を契機にモデル化への真摯な取り組みが開始されるならそれもよし。一気に完成品を狙わずできるところから徐々に蓄えてゆく手もあろう。システム作りと同じように。

人によってその背景となる環境、技術レベル、SWDBの構築目的、目標時期が異なり、しかも華々しい成功例が少なく概念だけが先走った感のあるテーマを議論することの難しさを味わった。誰がいつ頃何のために利用するかという前提を設けて討論できればよかったが、Chair.の力不足でそこまでの討論ができなかったことをお詫びしたい。メタな討論に終始し各論ができなかったが、もう少し地道な努力、実践を続けた後、いつの日にか各モデル化手法を実例をもとに分析できるような討論をしてみたい。