SEAMAIL

Newsletter from Software Engineers Association

Volume 6, Number 8-9-10-11 1991

目　次

事務局から　1

第 8 回 SEA 夏のプログラミング・ワークショップ報告　3

- まえがき　4
- 1. はじめに　5
- 2. 方法論の紹介　9
- 3. 課題　55
- 4. 各社の CASE ツールの概要　63
- 5. 中間発表　89
- 6. 最終発表　107
- 7. パネル討論　132
- 8. 各グループの討論記録　139
- 9. グループ・コンサルタントのコメント　169
- 10. まとめ　184
- 11. ポジションペーパーとアンケートの回答　189
- 12. 資料　233
- 13. 参加者名簿　248

Call for Papers

- Wulumuqi International CASE Symposium　250

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

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

その主な活動は，機関誌SEAMAILの発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約200人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，1000余名を越えるメンバーを擁するにいたりました．法人賛助会員も約80社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

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

代表幹事： 熊谷章

常任幹事： 市川寛　落水浩一郎　岸田孝一　中野秀男　深瀬弘恭　松原友夫　山崎朝昭

幹事： 飯沢恒　筏井美枝子　上原憲二　臼井義美　奥山充　片山卓也　亀田繁　窪田芳夫　小林俊明
杉田義明　田中一夫　玉井哲雄　鳥居宏次　中谷多哉子　中山照章　野村敏次　野村行憲　浜野善吉
林香　平尾一浩　藤野晃延　二木厚吉　堀江進　本位田真一　盛田政敏　山崎利治　渡邊雄一

会計監事： 辻淳二　吉村成弘

分科会世話人 環境分科会(SIGENV)：田中慎一郎　渡邊雄一
管理分科会(SIGMAN)：野々下幸治
教育分科会(SIGEDU)：杉田義明　中園順三
ネットワーク分科会(SIGNET)：小林俊明
法的保護分科会(SIGSLP)：能登末之
調査分科会(SIGSURVEY)：岸田孝一　野村敏次

支部世話人 関西支部：臼井義美　中野秀男　盛田政敏
横浜支部：藤野晃延　北條政顕　野中哲　松下和隆
長野支部：市川寛　佐藤千明　細野広水
名古屋支部：筏井美枝子　岩田康　鈴木智　平田淳史
九州支部：平尾一浩　藤本良子

賛助会員会社： 池上通信機　NTTソフトウェア研究所　NTT九州ネットワーク技術センタ　NTT九州技術開発センタ
PFU　SRA　SRA東北　SRA北海道　アスキー　ウチダエスコ　エイ・エス・ティ　エスケーディ
エヌ・ティ・ティ・システム技術　オムロンソフトウェア　カシオ計算機　構造計画研究所　キャノン新川崎事業所
クレオ　クレオアールアンドディ　ケーシーエス　コグノスジャパン　サン・ビルド印刷　システムラボムラタ
ジェーエムエーシステムズ　ジャストシステム　スインク　セイホーソフトウェア
セントラル・コンピュータ・サービス　ソフトウェアコントロール　ダイキン工業　テクノバ　テスク
テックシステムズ　ニコンシステム　ニッセイコンピュータ　ヒラタ・ソフトウェア・テクノロジー
ビーアイテイ　ファコム・ハイタック　フクダ電子　ムラタシステム　リコー　リコーシステム開発
リバテイーシステム　安川電機　近畿日本ツーリスト　古河インフォメーション・テクノロジー
三菱電機セミコンダクタソフトウェア　三菱電機メカトロニクスソフトウェア　三菱電機関西コンピュータシステム
三菱電機東部コンピュータシステム　三菱電機北伊丹製作所　松下ソフトリサーチ　新日鉄情報通信システム
新日本製鉄エレクトロニクス研究所　中央システム　辻システム計画事務所
東芝アドバンストシステム　東電ソフトウェア　東北コンピュータ・サービス　日本NCD
日本エム・アイ・シー　日本データスキル　日本ユニシス　日本ユニシス・ソフトウェア　日本情報システムサービス
日本電気ソフトウェア　日立エンジニアリング　日立ビジネス機器　富士ゼロックス情報システム　富士写真フィルム
富士通　富士通エフ・アイ・ピー　富士通ビー・エス・シー　（以上72社）

SEAMAIL、Vol. 6, No. 8～11　1992年3月30日発行
編集人　岸田孝一
発行人　ソフトウェア技術者協会（SEA）
〒160　東京都新宿区四谷3－12　丸正ビル5F
TEL: 03-3356-1077　FAX: 03-3356-1072
印刷所　サンビルド印刷株式会社　〒162　東京都新宿区築地町8番地
定価　2000円（禁無断転載）

事務局から

☆

花粉症の身にはつらい春がやってきました．3/18 の Forum で一応 1992 年度のイベントは終わりましたが，すぐにまた 4/2 のセミナー受付けで，あいかわらず慌ただしい事務局です．

☆☆

ところで，SEAMAIL のほうは，みなさまのご期待通りに遅れていて，ようやく Vol.6 (1991 年) の第 5 冊目まで辿りつきました．今回は部厚い No.8-9-10-11 合併号ですので，Vol. 6 は 4 月末に仕上がる予定のあと 1 冊でクリヤ．5 月からようやく Vol.7 に入ります．

☆☆☆

この号は，一昨年の 9 月に行われた「若手の会 in 盛岡: Case Study of CASE Tool」の報告です．原稿が落水先生の手から編集部に届けられたのは，昨年 8 月の末でしたから，事務局としては，遅くとも秋の終わりには発行できると予想していたのですが，編集部サイドでの作業（よくわかりませんが，TeX から roff への変換，ページ割り付けのやりなおし，文章の手直しなど）に思いの他時間がかかり，とうとう春を迎えてしまいました．

☆☆☆☆

しかし，門前の小僧よろしく全国縦断フォーラムでの討論や質疑を聞いていると，CASE の一般への普及はまだまだのような感じですので，このレポートも決して時代遅れではなく，十分会員のみなさんのお役に立つものと思います．

☆☆☆☆☆

第8回
SEA 夏のプログラミング・ワークショップ
― Case Study of CASE Tools ―
報 告 書

1990年9月5日(水)～9月8日(土)

於： 岩手県・盛岡市

ソフトウェア技術者協会

まえがき

この報告書，は 1990 年 9 月 5 日（水）から 9 月 8 日（土）まで，盛岡市でおこなわれた第 8 回夏のプログラミング・ワークショップ（若手の会 in 盛岡）の内容をまとめたものである．今回の若手の会は CASE をテーマとして取り上げ，3 泊 4 日にわたり，CASE ツールの実習を交えて，実にさまざまの話題が討論された．

報告書の内容はワークショップの流れにそってまとめられている．

- まず，「はじめに」の章で，このワークショップの目的，構造等を御理解いただきたい．
- 「方法論の紹介」の章には，構造化分析/設計法，オブジェクト指向分析/設計法に関する非常によくまとまった解説がある．
- 「課題」の章に記載された問題は，今回のワークショップで利用されたものである．
- 「各社の CASE ツールの概要」の章には，今回のワークショップに御協力いただいた CASE ベンダ各社のツールの説明が各社の専門家の手によってなされている．
- 「中間報告」および「最終報告」の章は，ワークショップの参加者による CASE ツールの試用経験が報告されている．読者は，CASE ツール導入にあたっての問題点や効果，教育法へのヒント等を，これらの生の声から読みとることができるものと思う．
- 「パネルディスカッション」の章では，ソフトウェア設計方法論の教育法，大規模ソフトウェア開発の支援，CASE ツールの導入法等についての専門家の見解を知ることができる．
- 「各グループの討論記録」の章では，各グループの活動を冷静な立場から観測・報告したものであり，違った角度からワークショップの内容を伺うことができる．
- 各グループにはプログラム委員がアドバイザ，あるいは・コンサルタントとして参加した．その方々の感想を「グループ・コンサルタントのコメント」の章で知ることができる．
- 最後の「まとめ」は筆者が 1990 年 10 月の SEA Forum で報告した内容を掲載したものであり，ワークショップ参加者の背景や参加後の感想等をまとめた資料である．
- ワークショップ参加者には全員ポジションペーパを書いていただいたが，その内容を「ポジションペーパとアンケートの回答」の章にまとめた．

今回のワークショップ開催にあたっては，実行委員長の野村行憲さんをはじめ，プログラム委員の各氏，さらには CASE ベンダ各社の方々に大変お世話になった．ここで，あらためて深く謝意を表したい．

また，この報告書をまとめるにあたり，貴重な勉学の時間をさき連日夜遅くまで精力的に活動してくれた静岡大学工学部情報知識工学科落水研究室のスタッフ各位，特に太田剛，方学芬，下津直武，中尾博司，石川洋介，古川勝也，小澤一彰，関谷和愛の各氏に心より感謝の意を表する．

1991 年 8 月 31 日

落水 浩一郎

1. はじめに

ワークショップ開催報告　実行委員長
野村 行憲（岩手電子計算センター） 6

オリエンテーション　プログラム委員長
落水 浩一郎（静岡大学） 8

開 催 報 告

実行委員長
野村 行憲

1. はじめに

CASE という言葉が，昨年あたりからにわかに脚光を浴びてきている．このことは，わがソフトウェア技術者協会（SEA）の主催するセミナーでも，CASE というキーワードがタイトルについただけで申込が殺到することからも容易にうかがえる．

こうした背景の中で，いま日本で販売されている代表的な CASE ツールを一堂に集め，チュートリアルを受けながら，それらを実際に試用して設計を行ってみるという機会を持つことは，強く希望はしても，なかなか叶えられない夢であった．

いたがって，この夢を現実のものとする企てを，夏のプログラミング・ワークショップで計画した時には，正直なところ多少の不安が心をかすめていたが，SEA メンバーの強力な後押しと，ツール・ベンダ各会社の寛大な御協力によって，結果的には私の心配は杞憂に終り，無事ワークショップが開催できたことを嬉しく思う．

2. 開催の目的

従来，この夏のプログラミング・ワークショップは，（4GL やフォーマル・スペックをとりあげた第 6 回の八ヶ岳ワークショップを除けば），参加者によるグループ討論を中心にプログラムを構成してきたが，今回は，初の試みとして，各種の CASE ツールを開発または販売するいわばライバル会社を一堂に集め，参加者にそれらのツールを実際に使って貰い，学習と評価を行なう形式とした．

CASE をめぐる議論の代表的なものとして，使い物になる／ならないとか，役に立つ／立たないといったことが云々されているが，実際の使用経験を持たない人たちが机上の空論を戦わせている場合が多い．そうした不毛の議論ではなく，実際のソフト開発現場で働いている人々が直接ツールに触れてみて，その有用性を探るというのが，今回のワークショップの目的とするところであった．

3. 定員枠の拡大

皆さんへの参加案内では定員を 35 名とし，ポジションペーパの審査を行なって参加者を選定することになっていたが，応募総数 45 名から参加者を選定するための拡大実行委員会において，落水プログラム委員長から，「CASE Tool を 1 人でも多くの人たちにさわって貰いたいので，できれば全員を参加させたい」との意見が出され，委員全員で検討した結果，多少の運営上のリスクを覚悟で申込者全員を参加させることになった．

その結果，一部の参加者から，「グループのメンバーが多すぎて Tool に触れる時間が少なく残念だ」といった不満の声も聞かれたが，この趣旨をご理解いただいて，ご容赦いただきたいと思う．

4. WS 開催

会場に，SA ／ SD 系の Tool 3 種，OOA ／ OOD ／ OOP 系の Tool 2 種をそれぞれ組み込んだワークステーションが設置され，ワークショップがスタートした．内容の詳しい報告は，プログラム委員長の落水先生を始めとする各プログラム委員の報告に委ね，ここでは省略するが，終始熱心な討論やチュートリアルが行なわれ，熱気に溢れた 4 日間であった．

また，実際に CASE Tool を使用するのに先立って行なわれた SA ／ SD および OOA ／ OOD ／ OOP の開発方法論の講演は，タイムリーであり，その後に行われた各ツール説明の理解を助けたことと思う．

5. おわりに

今回のワークショップ運営は，新しい試みであったため，多少ギクシャクした点もあって，参加者の皆さんにはご迷惑をお掛け致したことと思う．ここでもう一度お詫び申し上げる．

ところで，皆さんのワークショップの内容に対する感想はどうだったろうか？　このワークショップできっと

何かを掴んだことだと思う．また，全国各地から集まった同業の人々との語らいの中で，いろいろ見聞を深め，交流も深められたことと思う．今回のワークショップでの経験が，皆さんのこれからの仕事や生活の色々な局面で，何等かの役に立つことを希望して止まない．

最後に，落水プログラム委員長を始めとするプログラム委員の方々，記録や準備などを協力して戴いた静岡大学の学生の方々，SEA 事務局の方々の御協力に謝意を表する．

また，これなくしては，今回のワークショップが開催できなかった CASE Tool およびワークステーションを快く提供していただき，親切なインストラクタ役をつとめてくださった下記ベンダ各社の皆さんにあらためて厚くお礼を申し上げたい．

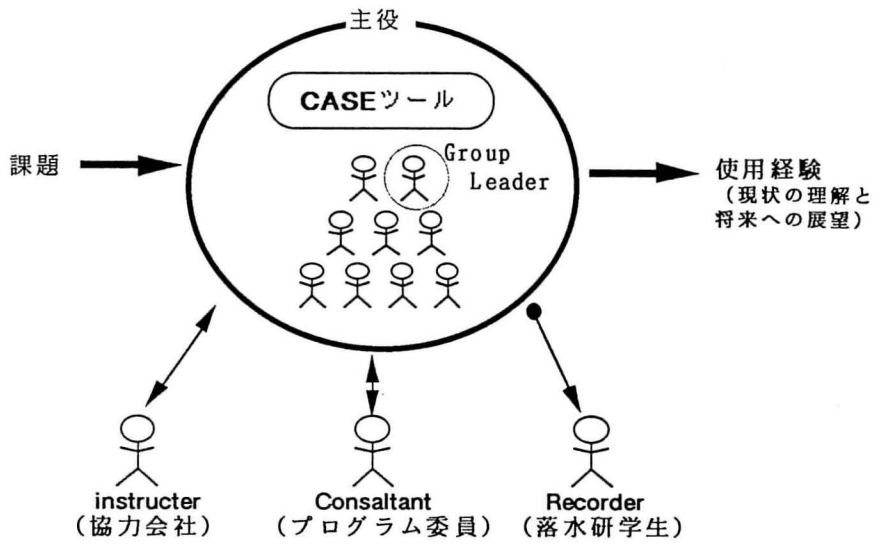
CASE Bench:	渡部健次（ソニーテクトロニクス）
PWB & HyperBook:	足立太郎（PFU）
Smalltalk-80 & Grapher Gear:	青木淳（富士ゼロックス情報システム）
StP:	桜井麻里（SRA）
Teamwork:	田中耕市（東陽テクニカ）

オリエンテーション

プログラム委員長
落水 浩一郎

1. ワークショップの目的

下図に示すようなチームを 5 つ作り，具体的な設計課題解決の過程を通じて，CASE ツールの現状に対する理解と将来への展望を持つ．



2. ワークショップの構造

起承転結の構造を次のように定める

2.1 起 (9 月 5 日)

CASE を理解するには，(1) ツールの背後にある方法論のポイントの理解，および (2) 各 CASE ツールの記法・操作法の理解，が必要である．(1) は，荻原・青木両氏にチュートリアルをお願いし，(2) は各ツール・ベンダのインストラクタに解説していただくこととする．

2.2 承および転 (9 月 6 ～ 7 日)

まず，とにかく使ってみて，中間報告でアイデアを交換しつつ，最後に何かを掴む．

2.3 結 (9 月 8 日)

(1) CASE ツールの理解を前提としてお互いに意見を交換し，(2) 将来の技術課題等についての期待と提言を建設的な立場からまとめる．

3. そして最後に

成果を SEAMAIL 誌上で公開しよう！

2. 方法論の紹介

構造化分析・構造化設計・リアルタイム構造化設計
荻原 剛志（大阪大学） 10

オブジェクト指向ソフトウェア開発
青木 淳（富士ゼロックス情報システム） 20

構造化分析，構造化設計，リアルタイム構造化設計の紹介

大阪大学 情報処理教育センター 荻原剛志
(mail: ogihara@ics.osaka-u.ac.jp)

1. はじめに

構造化分析／構造化設計 (SA/SD: Structured Analysis/Structured Design) とは，システムを入力と出力を結ぶデータ変換のネットワークととらえ，このネットワーク構造から，システムを構成するモジュール構造を作り上げるための技法である．

本稿で紹介する構造化分析法，構造化設計法は，それぞれ主に DeMarco[1], Yourdon & Constantine[2] に基づく．本稿では例を用いながら，これらの方法の概要を説明する．これらの詳細，および SA/SD 法とほぼ同等な Myers の複合設計法 (Composite Design)[3][4]，あるいは Ross らの SADT(Structured Analysis and Design Technique)[5] については各文献にあたっていただきたい．なお，SA/SD 法は各「流派」によって適用方法や用語がかなりまちまちであり，注意する必要がある．

SA/SD 法はどちらかというと事務処理などの分野に適しており，制御やタイミングについての要求が不可欠な分野には対応できなかった．最近，SA 法にこれらの要素を取り入れた，リアルタイム構造化設計法と呼ばれるものがいくつか提案されている [6][7]．そこで本稿では，それらの中から，Ward[6] の方法についても概略を説明する．

2. 要求分析 ～ 構造化分析法

2.1 準備

構造化分析法は，EDP システムを構築するための仕様書を作成することを目的とする．この仕様書は単なる文書ではなく，データフローダイアグラム (Data Flow Diagram: 以下，DFD と略記する) を用いた機能的，構造的なものである．なお，DeMarco の著書 [1] では，この構造化仕様書の書き方以上に，いかに顧客の要求を引き出し，それを仕様書に反映させるかということに多大の労力が注がれている．

本稿では，例題として「大学附属図書館における図書購入業務問題」を用いる．これは文献 [8] の問題の一部を変更し，簡単化したものである．

> 大学附属図書館は，各研究室からの図書の購入依頼に応じて図書を発注，納品の受け入れなどの業務を行う．
>
> 図書館は各研究室から図書の購入依頼書を受け取る．購入依頼書には書名（著者名，発行所等を含む），価格，研究室名が記入されている．購入依頼書を受け取ったら，記入もれがないことを確認し，その研究室の予算でその図書を購入できるかどうかを調べる．これは，研究室の予算残高と，支出予定額（発注したがまだ納品されていない図書の価格の合計）から調べることができる．
>
> 購入できることが確かめられたら，購入依頼書から発注票とその控えを作り，発注票は書店へ送る．発注票には，書名，価格，発注者名，発注番号，発注日が記入される．発注者名には研究室名が記入される．発注番号は発注票を識別するための記号列である．同時に，発注した研究室の支出予定額を増やしておく．
>
> 書店から図書が納品書とともに届けられたら，納品書に記載された発注番号から発注票の控えを調べ，その図書が本当に発注されたものかどうかを確認し，そうであれば受け入れる．納品書には発注番号の他に，書名と図書の実際の価格が記載されている．受け入れの際は，その図書を購入する研究室の予算残高と支出予定額を変更する．図書は配本票とともに研究室に届けられる．配本票には，書名，実際の価格，研究室名が記載されている．

以下，この問題について，DFD，データディクショナリ，およびミニ仕様書を作成してゆく．

2.2 コンテキストダイアグラム

DFD は階層的に構成されるが，作成すべきシステムと外部要素を区別し，システムが何をすべきかを明確にするため，コンテキストダイアグラムと呼ばれるダイアグラムを記述する．

図 2-1 は図書購入問題のコンテキストダイアグラムである．DFD では，名前付けされたバブル（円）でプロセスを表す．プロセスとは入力を出力に変換する処理を指す．実世界での，人間や機械による作業を抽象化したものと言える．コンテキストダイアグラムとは，逆に言えば，システム全体をひとつのプロセスとして記述したも

のである．

矢印はデータの流れと要素間のインタフェースを表し，これをデータフローと呼ぶ．データフローにはそこを流れるデータの名前を記述する．データフローにはコントロールの流れを記述することはできない．

四角はデータ源泉／データ吸収(あるいはエンティティ)といい，システムの外部の組織などを表す．

2.3 データディクショナリ

DFDの記述を進めて行くなかで新たなデータが現れたら，そのデータの内容の定義をデータディクショナリに記入していく．データディクショナリとは，データフローを流れるデータ，ファイル（後述）の内部のデータの構造の定義を記述したものである．データディクショナリの記述中の記号の意味は表1-1に示す通りである．例えば，ある作業グループを表すデータの形式は

作業グループ ＝ グループ名 ＋ {構成員}
構成員 ＝ 氏名 ＋ 性別 ＋ (年齢)
性別 ＝ [男 / 女]

というように表せる．構成員が1～8名であることを表したければ，$^{1}_{8}${構成員} あるいは 1{構成員}8 というように記述すればよい．上限，下限のどちらかだけを記述することもできる．

データの定義は，定義が自明であるデータ項目か，まだ定義が決まっていないデータ項目に至るまで記述を行う．例えば，「人数」というデータ項目は整数であることが自明である．また，「氏名」という項目の詳しい定義（最大n個の文字の連続である，など）は定義する必要が生じるまではそのままにしておいてよい．

図2-1のコンテキストダイアグラムの段階では，次のようにデータを定義できる．

購入依頼書 ＝ 書名 ＋ 価格 ＋ 研究室名
発注票 ＝ 書名 ＋ 価格 ＋ 発注者名
　　＋ 発注番号 ＋ 発注日
発注者名 ＝ 研究室名
納品書 ＝ 発注番号 ＋ 書名 ＋ 実価格
配本票 ＝ 書名 ＋ 実価格 ＋ 研究室名

2.4 データフローダイアグラムの記述方法

図2-2は，コンテキストダイアグラムの内容をDFDとして記述したものである．

2本の平行線（1本のこともある）で表されているのはファイルである．ファイルはデータの一時的な保存場所を表す．帳簿，書類の束，テープ，データベース等々を抽象化したものである．ファイルはプロセスとのみデータフローで結合される．単一のファイルとプロセスの間のデータフローには名前は記述しない．ファイルの内容がデータフローの内容をも指すと考えるためである．

ファイルからプロセスへのデータフローは，そのプロセスがファイルからデータを読み込むことを表す．プロセスからファイルへのデータフローは，そのプロセスがファイルを読み書きする，あるいは単に書き込むことを表す．アクセスの結果，ファイルの内容を変更する場合には，ファイルからのデータ読み込みが伴っていても，ファイル側へ矢の向いたデータフローで表現する．ひとつのプロセスがデータの読み込み，書き込みの両方を行う時は，両端に矢のついたデータフローで表現する．

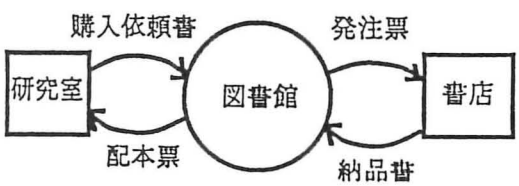


図2-1　図書購入問題：コンテキストダイアグラム

表1-1　データディクショナリの記号

記号	意　味
＝	.....に等しい
＋	.....と.....
[ ]	.....または.....
｜ ｜	.....の繰り返し
( )	.....は任意選択

ダイアグラム0

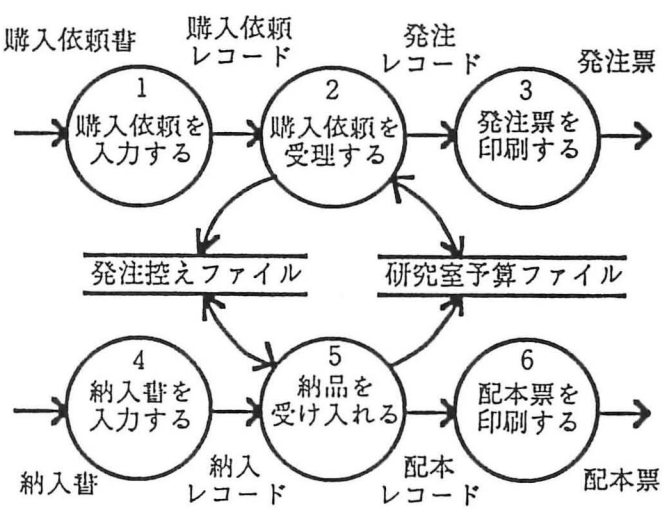


図2-2　図書購入問題：ダイアグラム0

DFDを記述するには，まずそのダイアグラムの最終的な入出力（外部，あるいは上位プロセスとの入出力）を記述し，次に作業の単位を想定したバブルとその間を結ぶデータフローを記述してゆく．記述は入力側からでも出力側からでもよい．データフロー，ファイルにはデータの内容を適切に表す名前（名詞句）を付ける．「データ」，「帳簿」のような，内容を明確に表さない名前は良くない．次にバブルに入力と出力を表現する名前を付ける．バブルの名前は基本的には動詞で，「○○を××する」というようにする．「処理する」とか「扱う」のような漠然とした言葉は良くない．バブルにはまた，1, 2, ... と通し番号を付ける．

データフローが記述できたら，新しいデータフローはデータディクショナリに記述して行く．また，あるバブルに流れ込むデータフローだけから，その出力がすべて構成できることを確認する．また，入力だけ，出力だけしか持たないバブルがないかをチェックする．書き込まれるだけのファイルも間違いであることが多い．DFDは何回も書き直しを繰り返して，適切なものに仕上げてゆかなければならない．

DFDは，システムの定常的な稼働状況を記述したものであり，初期設定や終了時処理は考慮しなくてよい．また，プロセスの処理の時間的関係やコントロールの流れは記述しない（DFDはフローチャートやオートマトンではない）．また，はじめのうちは，小さなエラー処理には目をつぶっておく．

図2-2のデータフローの定義を以下に示す．

発注控えファイル ＝ { 発注票 }
研究室予算ファイル
　＝ { 研究室名 ＋ 予算残高 ＋ 支出予定額 }
購入依頼レコード ＝ 購入依頼書
発注レコード ＝ 発注票
配本レコード ＝ 配本票
納入レコード ＝ 発注番号 ＋ 実価格

2.5 ダイアグラムの階層化

あるDFDのプロセスを，さらに分割して下層のDFD（つまりサブシステム）を記述することができる．こうしてシステム全体は，階層化された一連のDFDで構成することができる．

あるプロセスnを分割してできたDFDは「ダイアグラムn」と呼ばれ，その中のプロセスには，n.1, n.2, ... と番号が付けられる．プロセスnを含むダイアグラムとダイアグラムnを，相対的に見て，互いに親ダイアグラム，子ダイアグラムと呼ぶことがある．コンテキストダイアグラムの直下（最上層と呼ぶ）のDFDは，特別にダイアグラム0と呼ぶ．

各ダイアグラムのプロセスは，一度に全体を把握できる程度の個数で，しかもほぼ等しい記述のレベルで表現できていなければならない．プロセスの個数の目安が必要ならば 7±2 程度がよい．

ダイアグラムnの最終的な入出力は，プロセスnの入出力と一致していなければならない．一致がとれていることを「バランスがとれている」と言う．ただし，出力に関する重要でないリジェクト処理は親子のバランスをとる必要はない．

図2-3は図2-2のプロセス5に対応する，ダイアグラム5である．ここで，「発注番号」と「実価格」はプロセス5への入力である「納入レコード」の構成要素である．このような場合もバランスがとれていると見なせる．ダイアグラムの外にあるファイルもこのように記述する．

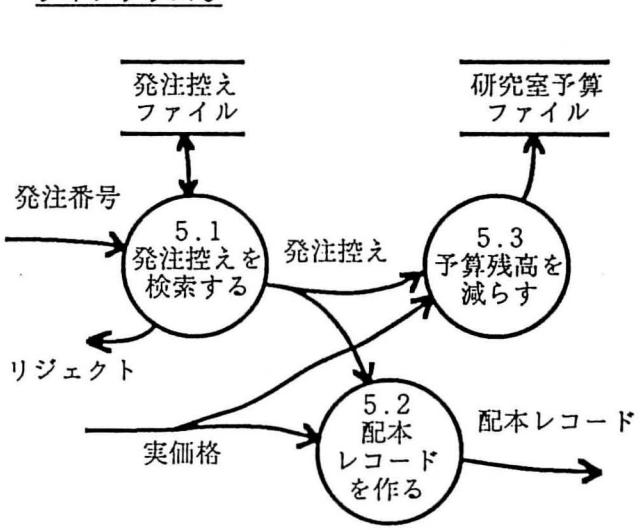


図2-3　図書購入問題：ダイアグラム5

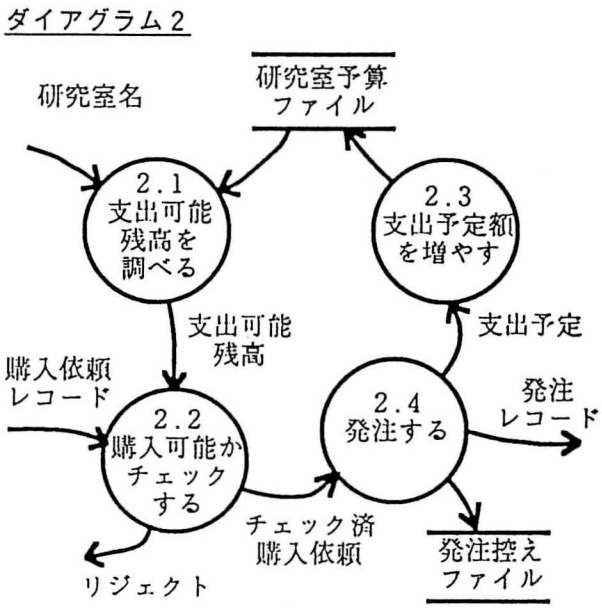


図2-4　図書購入問題：ダイアグラム2

ダイアグラム2.4

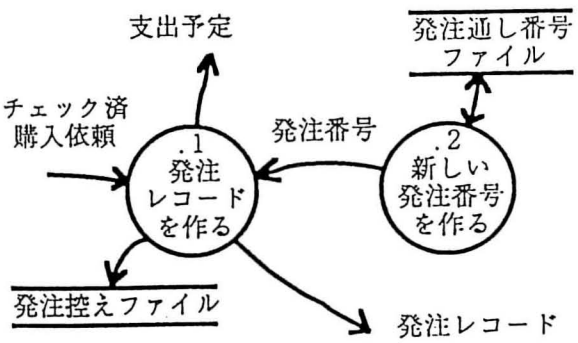


図2-5　図書購入問題：ダイアグラム2.4

プロセス：　2.4.1 プロセス名：　発注レコードを作る
入力された「チェック済購入依頼」について以下の動作を行う。 「発注番号」を得る。 「発注日」を得る。/* マシンから */ 「発注者名」と「価格」から「支出予定」を作成する。 「書名」,「価格」,「発注者名」,「発注番号」,「発注日」から「発注レコード」を作成する。 「発注控えファイル」に「発注レコード」を書き込む。 「支出予定」,「発注レコード」を出力する。

図2-6　図書購入問題：プロセス2.4.1のミニ仕様書

図2-4はダイアグラム2である．このプロセス2.4をさらに分割したのが図2-5である．このように，あるプロセスの内部にだけ見えるファイル（内部ファイル）を使うことができる．また，バブルの名をこのように省略して記述してもよい．

新たなデータの定義は以下の通り．

支出予定 ＝ 研究室名 ＋ 価格
チェック済購入依頼 ＝ 購入依頼レコード
発注控え ＝ 発注レコード

2.6 ミニ仕様書

プロセスの分割を進めていって，それ以上分割できなくなった，あるいは分割の必要がないと決めたら，そのプロセスを基本機能と呼ぶ．システムの基本機能を見つけて，その相互の関係を調べることがDFD作成の目標である．

基本機能には，そのプロセスが何を行うか記述したミニ仕様書を対応させなければならない．ミニ仕様書は自然言語，あるいは擬似コードなどで簡明に記述する．ミニ仕様書には，対応するバブルの番号を付ける．

プロセスが基本機能であるかどうかの明確な判断基準は存在しないが，入出力が煩雑でなく，1ページ以内のミニ仕様書に完全に記述できると思えた時，という条件が目安になろう．

プロセス2.4.1のミニ仕様書は図2-6のようになる．

3. モジュール構造の作成 ～ 構造化設計法

構造化設計法は，与えられた仕様からシステムの機能の分析を行い，適切なモジュール構成を作り上げることを主な目的としている．要求仕様はDFDに展開し，これに対して変換分析法，あるいはトランザクション分析法という方法を適用することにより，ソフトウェアの構成を表す構造図を作成する．この時，モジュール分割の目安として，モジュール間の結合度，凝結度という尺度が用いられる．

3.1 構造図で用いる記号

図3-1に構造図で用いる記号を挙げる．(a)はモジュールAから1回ずつ活性化される（呼ばれる）モジュールB,C,Dを表す．(b)は2重になったループである．(c)は条件付活性化である．モジュールBは条件が真の時に活性化される(if-then)．モジュールC,Dはif-then-elseの例である．(d)において，B,C,Dもif-then-elseの例であるが，条件が真の時はB,Cが活性化される．また，D,E,Fのように多分岐も記述できる．

図3-2中の，尾部に丸のついた小さな矢印はモジュール間のデータの移動，参照を表す．白抜きの丸はデータであることを，黒い丸は制御要素が含まれていることを示す．このようにデータや制御情報を構造図中に記述す

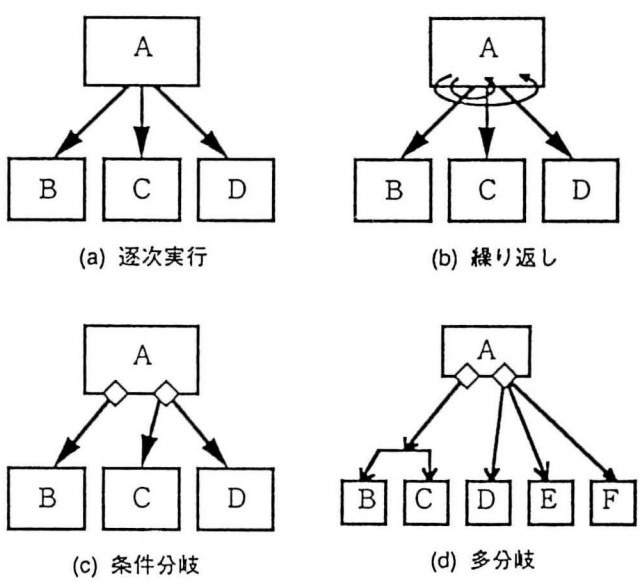


図3-1　構造図の記述方法

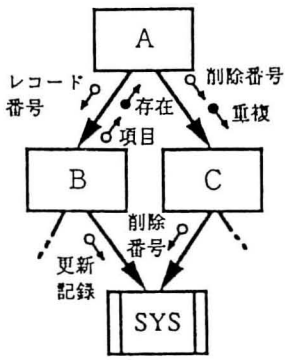


図3-2　データフローの記述

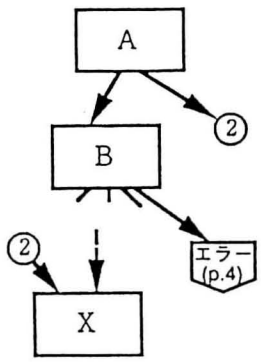


図3-3　連結子による記述

る方法と，データの内容を一覧表にして作成する方法があり，場合によってどちらか，あるいは組み合わせて使用される．モジュールSYSは既定義，あるいはシステム提供のモジュールである．

図3-3はページ内の連結子，ページ外への連結子を表している．

3.2 モジュールの独立性の尺度

モジュール構造の設計を進めて行く上で留意すべき，モジュール間の関係についての尺度として，結合度(coupling)と凝結度(cohesion)が提案されている．ここでは，これらの尺度のレベルなどの詳細な議論は避け，考え方の概略について述べる．

結合度は構成図に現れる2つのモジュール間の情報交換の方法について注目したものである．基本的な考え方は，あるモジュールの変更が他のモジュールに影響しないこと，あるモジュールの動作をできるだけ他のモジュールからの情報に依存しないようにすることである．モジュールが強く影響しあう程，結合度は高くなる．

情報交換の方法は，制御情報を伴わない，データのみによるのが好ましい．交換されるデータの量も少ない方がよい．制御情報を含む場合にも，その影響が広い範囲のモジュールに及ばないようにする．特に，上層から下層へのスイッチの委譲は好ましくない．情報はサブルーチンコールの際のパラメータとして受渡しするのが良い．

凝結度はモジュールの内部のまとまりの良さを表す．モジュールとしての分かりやすさとともに，変更の容易さ，変更の影響がモジュールの外部に及びにくいことが求められる．

一番望ましいとされるのは，1つの明確に定義された機能のみを果たすモジュール，あるいは抽象データ型（またはオブジェクト）として情報隠蔽されたモジュールである．パイプライン的な処理をモジュールとしたもの，同じ入出力データを扱う機能をまとめたもの，手順としてまとめて行われる機能をまとめたものが，この順番で望ましいとされている．互いに関係が希薄な機能を集めたモジュールは好ましくない．

3.3 変換分析

変換分析は，DFDから構造図を作成するための方法のひとつであるが，それは完全な構造図を自動的に作成してくれるものではない．むしろ，構造図を作成する上での便利なガイドラインであると捉えるべきである．

まず，モジュールの4つのタイプについて説明する．最初のタイプは求心型モジュールと呼ばれるもので，図3-4(a)のように，下位モジュールから渡された情報を上位に渡すものであり，典型的には入力装置から処理部分へ渡すためのデータ変換部分などがこれにあたる．

これと逆に，出力装置へのデータの変換部分など，上位モジュールから下位モジュールへ渡すもの（図3-4(b)）は遠心型モジュールと呼ぶ．

図3-4(c)のように，上位モジュールから渡されたデータを変換して再び上位モジュールへ渡すものを変換型モジュールと呼ぶ．計算を行うモジュールなどが典型的な例として挙げられる．

図3-4(d)は，下位モジュールの管理・調整を行うモジュールを表す．このようなモジュールを調整型モジュールと呼ぶ．

実際のモジュールはさまざまな要素を含み，これら4つのタイプに明確に分けることができないものも多いが，変換分析ではシステム全体の機能を把握する上で，これらの概念を積極的に使用する．

以下，変換分析の作業手順を説明する．

得られたDFDのうち，中心的な処理を行っていると考えられる部分，あるいは入出力から最も遠く離れた部分を中央変換部と呼ぶことにする．中央変換部は1つまたは複数のプロセスから成る．中央変換部以外の部分は，中央変換部で処理するためのデータを提供するか，処理された結果を出力するための部分である．中央変換部の位置および範囲の決定は設計者自身が十分考えて行わなければならない．

中央変換部を決めたら，以下のような規則に従って構造図への変換を行う．

(1) 中央変換部全体を表すモジュールを一番上位に置く．
(2) その下に，中央変換部への入力フロー（求心型フロー）に対応して求心型モジュールを置く．
(3) 同様に，出力フロー（遠心型フロー）に対応して遠

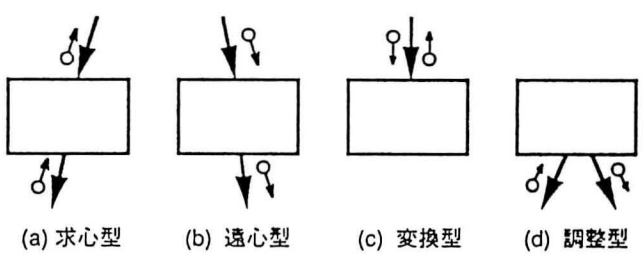


図3-4　モジュールの4つのタイプ

心型モジュールを置く.

(4) 中央変換部のプロセスに対応して変換型モジュールを置く.

(5) それぞれの下位のモジュールをさらに分解してゆく.

下位のモジュールの分割では，原則として，求心型モジュールは求心型と変換型のモジュールで，遠心型モジュールは変換型と遠心型のモジュールで分割する．適宜，調整型モジュールを作ってよい．変換型モジュールの下位モジュールへの分割には，特に決められた方法はない．

例として，図3-5のダイアグラムで，バブル4,5,6を中央変換部と見なすことにしたとする．これに対応する構造図は，例えば図3-6のように構成することができる．モジュールの箱の中の数字はバブルに対応させてある．バブル5,6のそれぞれからの遠心型フローは異なった方法で分解を試みている．

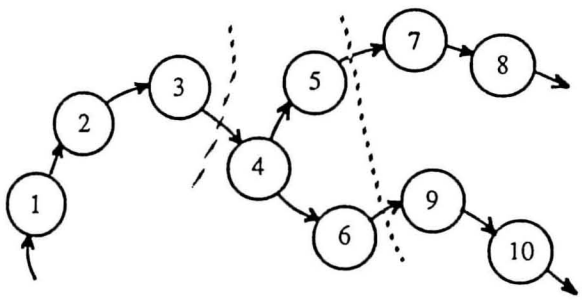


図3-5　中央変換部の決定

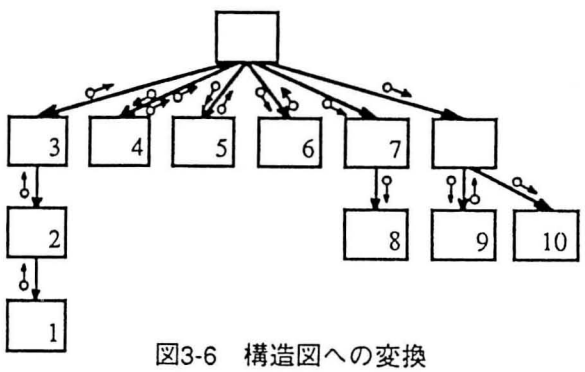


図3-6　構造図への変換

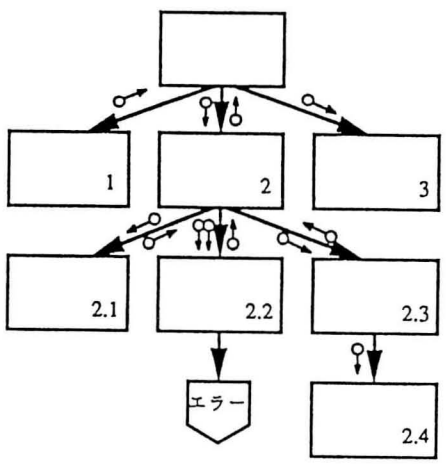


図3-7　図書購入問題：ダイアグラム0，ダイアグラム2の構造図

図書購入問題の例の，最上層のバブル1,2,3を構造図に変換し，さらにバブル2の下層についても変換を行ったのが図3-7である．ただし，DFDのすべての層をひとつの構造図に変換する必要はなく，DFDの階層を利用して構造図も階層的に作成しておくことができる．なお，本稿の図では省略しているが，各モジュールにはそのはたらきを表す適切な名前を記入しておく．

3.4 トランザクション分析

トランザクション(transaction)とは，システムに対するひとまとまりの処理要求であると言える．システムはトランザクションの内容を調べて，適切な処理を行わなければならない．

トランザクション分析は，DFDの記述からトランザクションを識別して，トランザクションを処理するのに適したモジュール構造を作成する方法である．ただし，変換分析同様，トランザクション分析の適用によって得られる構造図は，最終的なモジュール構造作成のためのたたき台であると考えるべきである．

トランザクション分析による変換の対象となるDFDは，図3-8のように，あるバブルにトランザクションが入力され，その内容に応じてどれかの処理を行うといった形態を持つものである．トランザクションを受け取り，分析し，適切な処理先へ送り出すモジュール（群）をト

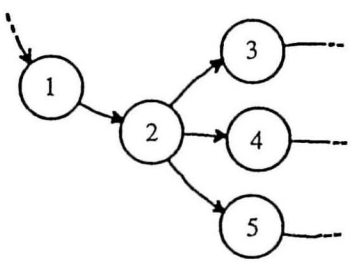


図3-8　典型的なトランザクションのパターン

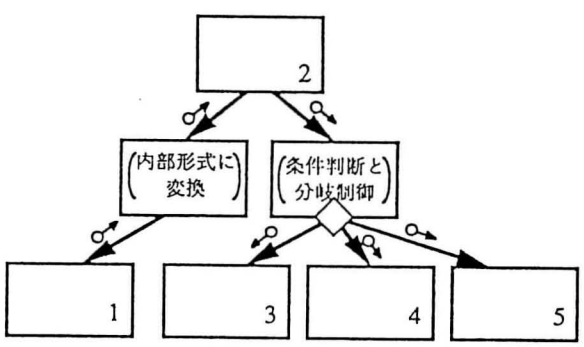


図3-9　トランザクションを構造図に変換するパターン

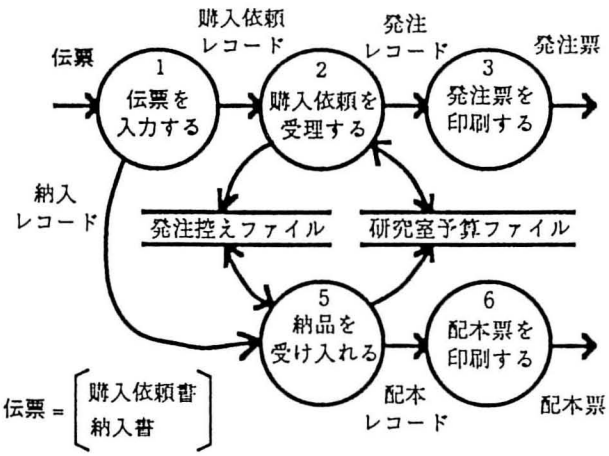


図3-10　図書購入問題：入力部分の変更

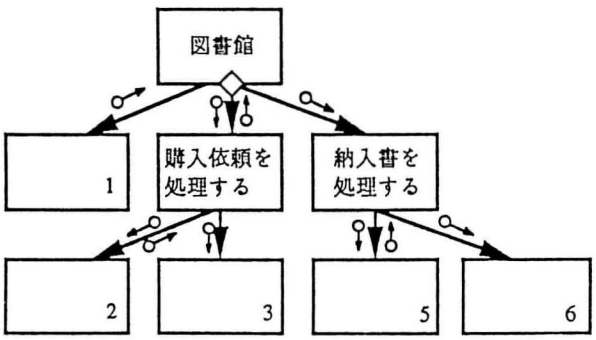


図3-11　図書購入問題：ダイアグラム0の構造図

ランザクションセンターと呼ぶ．図3-8では，バブル2がトランザクションセンターである．

図3-8は，図3-9のような構造図に変換できる．ただし，トランザクションを内部形式に変換する部分，あるいはそれぞれの処理を行うモジュールへの分岐の部分を上位のモジュールに埋め込んでもよい．

トランザクションセンターを識別し，図3-9のような構造図を得たら，変換分析で用いた方法を利用して下位モジュールを分解して行く．トランザクションセンターは必ずしもシステムの上部のモジュールとする必要はない．どこに置くのがよいかはモジュールの独立性の尺度によって判断される．トランザクション分析と変換分析は別のものではなく，それぞれを適切に使い分けながら構造図を構成してゆくことが必要である．

図書購入問題では，購入依頼書と納品書の扱いをまったく別にしていた（図2-2）．実際に1つのシステムとして実現する場合には，図3-10のように入力元を1つにまとめるのが良いであろう．この場合の構造図を図3-11に示す．

3.5 構造図の手直し

変換分析，トランザクション分析によって作成した構造図は，まず3.2で述べた尺度を適用して妥当性を調査してみる必要がある．また，実際は何もしないモジュールや，ひとつであまりに多くの作業をしているモジュールがないか，などを検討すべきである．

インプリメントする上での都合は，構造図がほぼ完成した時点か，どちらも妥当と思われる選択を決定する時点に持ち込むことができる．構造的にまとめ上げた要求が実現段階まで反映されていることが理想的である．

構造図を手直しする場合，上位モジュールの機能の一部を下位モジュールとして切り放す，あるいは逆に下位モジュールを上位モジュール内部に取り込む，複数の下位モジュールをまとめたり，切り放したりする方法が考えられる．また，共通の機能をサブルーチンやマクロとして使うこともある．さまざまな方法が利用可能であるが，これらはむしろ「構造化プログラミング」の範疇の問題であろう．

4. リアルタイム構造化設計法

ここでは，リアルタイムSA手法の中から，Wardの方法について説明を行なう．内容は文献[6]のうちの基本的な部分に限り，詳しい動作原理の説明，実行計画，拡張機能などは省略した．

4.1 変換図で用いる記号

Wardの方法では，DFDにシグナルなどのイベントやそれらを制御する特別なバブルを導入し，これを変換図と呼ぶ．変換図で用いられる記号を図4-1に示す．

バブルは変換部分を表す．実線のバブルはデータ変換部分で，主にデータの変換を行うが，シグナルを発生したりもする．数値処理や装置の制御，保存データへのアクセスなどの抽象化である．破線のバブルは制御変換部分で，各種のイベントやデータ変換部分の処理を制御する．システム制御ロジックの部分の抽象化である．

実線のアーク（弧）はデータフローを表す．このうち，矢が1つだけ付いたものは不連続なデータフロー，矢が2つ付いたものは連続したデータフローを表す．不連続なデータフローとは，離散的な別個の時間でのみ捉えられる値やその集合であり，トランザクションで受信されるデータの集まりなどを抽象化したものである．連続的なデータフローとは，時間的に連続して変化するデータの流れであり，温度や電圧などの量を抽象化したものである．

破線のアークはイベントフローを表す．イベントは何か起こったことを連絡したり，何かの処理の開始／中止を指示するために使用される．値は持たない．システムの割込やパルスを抽象化したものである．このうち，矢が1つだけ付いたものはシグナルである．シグナルは何かが起こったことを知らせる．矢が2つ付いたものは活性化，矢が2つ逆向きに付いたイベントは非活性化と呼ばれる．どちらも矢がついている側が受信側である．活性化は受信側に出力の生成を行なわせようとするもので

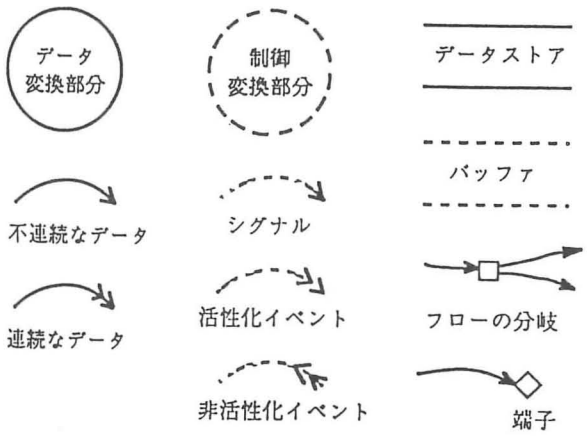


図4-1　変換図で用いる記号

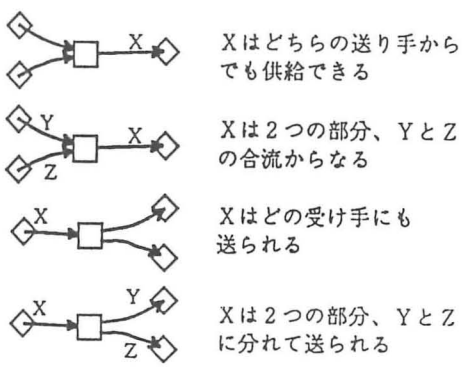


図4-2　フローの分岐と合流

あり，非活性化は逆に出力の生成を中止させようとするものである．

平行な2本の実線はデータストアであり，ファイルやメモリを抽象化したものである．破線の平行線はバッファである．バッファは不連続なデータあるいはシグナルの流れに対してキュー（待ち行列）の役割を果たす．データストアからの読み出しは非破壊的（内容を破壊しない）だが，バッファから読み出したデータ（あるいはシグナル）は取り除かれる．

小さなひし形（◇）はシステムの外部，あるいは上位の階層とのデータ／イベントの入出力端子である．

小さな正方形（□）はデータフロー，イベントフローが分岐，あるいは合流する箇所を表す．正方形は省略されることもある．分岐／合流の様子を図4-2にまとめる．また，不連続なデータフローからはシグナルが分岐できる場合がある．

4.2 記述の規則

変換図を記述する上での規則のいくつかを説明する．ただし，ここでは特に説明しない限り，「データ変換部分」として，それ以上下層の変換図に分解されないものを指すこととする．

(1) 図の構成要素と入出力フロー

データ変換部分はすべての種類のデータフロー，イベントフローを入力にできる．また，連続／不連続なデータフロー，シグナルを出力できる．ただし，不連続な入力（不連続なデータフロー，シグナル）を複数持つことはできない．

制御変換部分はシグナルのみを入力でき，シグナル，活性化／非活性化イベントを出力できる．

データストアへの入力／出力は不連続なデータフローだけであり，最低1つのデータ変換部分とつながっていなければならない．

バッファへの入力／出力は不連続なデータフロー，あるいはシグナルである．入力／出力はそれぞれ少なくとも1つのデータ変換部分とつながっていなければならない．また，ひとつのバッファはデータ専用，あるいはシグナル専用として使用しなければならない．

システムの外部（上位の変換図ではなく）からは，連続／不連続なデータフロー，シグナルを受け取り，また送ることができる．

(2) アクティヴな入力／出力

アクティヴな入力とは，不連続なデータフロー，あるいはシグナルであって，それを受信することによってデータ変換部分の実行が行われるものを言う．データストアからのデータフロー，連続なデータフロー，活性化／非活性化イベントはアクティヴな入力ではない．データ変換部分は最大1つのアクティヴな入力を持つ．

アクティヴな出力とは，不連続なデータフロー，あるいはシグナルであって，入力側へのアクティヴな入力となるものを言う．データ変換部分はアクティヴな出力をいくつも持ったり，持たないことができる．ただし，複数のアクティヴな出力を持つデータ変換部分は，ある時点で1つのアクティヴな出力しかできない．

データ変換部分への活性化／非活性化イベントは，1つの活性化イベントのみ（ただし，そのデータ変換部分が，アクティヴな入力，連続なデータ出力を一切持たないこと）を持つか，一対の活性化／非活性化イベントを持つかに限られ，そうでなければ一切のイベント入力を持たない．

不連続な出力（不連続なデータフロー，シグナル）を持ち，アクティヴな入力を持たないデータ変換部分は，対になっていない活性化イベントを受けた時，連続な入力の値がある特定の値になった時，あるいはデータストアに予め保存しておいた時刻になった時に出力を生成する．

(3) 変換部分の下層への分解

データ変換部分はさらに下層の変換図に分割できる．下層の変換図は制御変換部分を含んでいてもよい．それ以上分割しないデータ変換部分は基本機能としてミニ仕様書を対応させる．

制御変換部分もさらに下層の変換図に分割できるが，そこにはデータ変換部分を含めることはできない．制御変換部分は内部にステート（状態）を持つ．それ以上分割しない制御変換部分には，ステートの状態遷移を表すために，ステートマシン（Mealy型のオートマトン）の記述を対応させる．ステートマシンの記述方法は例題の説明のところで示す．

4.3 例題

ここで，以下のような簡単な例題について変換図の記述を試みる．

保温機能付きの電気ポットの制御部分を設計する．

このポットには沸騰ランプ，保温ランプ，沸騰ボタン，ヒーター，サーミスタ温度計が付けられている．

ポットに電源を入れると湯を沸かし始め，湯が沸騰すると保温に移る．保温中に沸騰ボタンを押すと，再び湯を沸かすことができる．湯を沸かしている時に沸騰ボタンを押しても何もしない．湯を沸かしている間は沸騰ランプ，保温中は保温ランプが点灯している．

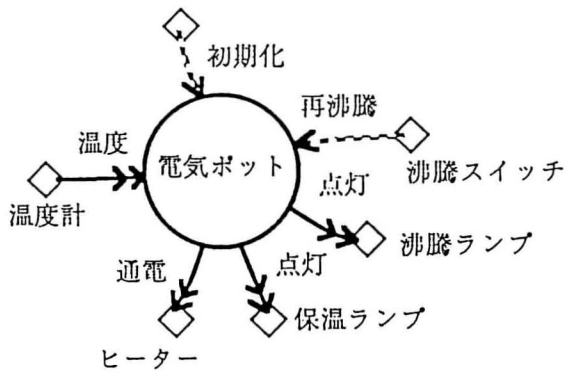


図4-3　電気ポット：コンテキストダイアグラム

湯を沸かしている間は1分毎に湯の温度を調べ，1分前の温度と変化がなくなった場合，あるいはその時の温度が100℃を越えた場合に沸騰したと見なす．

保温中は，湯の温度が80℃より低くなったらヒーターに通電し，温度が90℃以上になったら止める．

(1) コンテキストダイアグラムの記述

まず，コンテキストダイアグラムを記述する．システム外部から，およびシステムから外部へのシグナルやデータフローにどんなものがあるかをまとめ，これをダイアグラムに直接反映させる（図4-3）．変換図では，システムの起動を「初期化」のようなシグナルとして表現できる．また，各階層ではデータフローについてデータディクショナリを記述して行く（この例題では省略した）．

(2) 最上層の分割

最上層として図4-4を得たとする．制御変換部分「ポッ

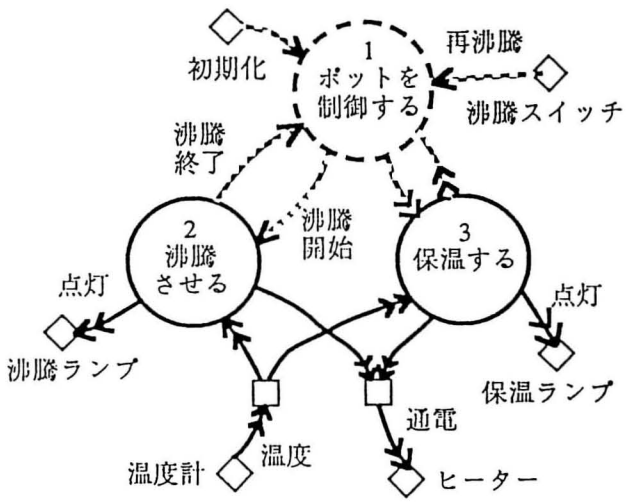


図4-4　電気ポット：最上層の変換図

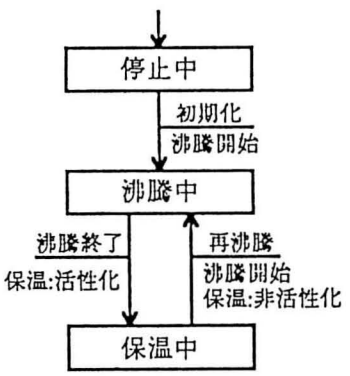


図4-5　ステートマシンの記述

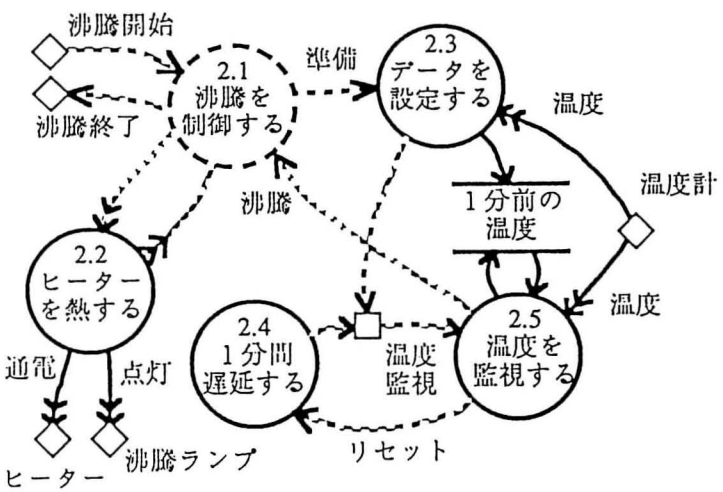


図4-6　電気ポット：変換図2（沸騰させる）

トを制御する」に対応するステートマシンは図4-5のようになる．Wardの方法では，ステートの表記に長方形を使用する．ステートマシンは入力されるシグナルによって現在のステートを変化させ，同時にイベントを出力する．なお，ステートマシンは表形式で記述することもできる．

(3) 下層への分割

データ変換部分「沸騰させる」はさらに，例えば，図4-6のように分解できる．「1分間遅延する」は，シグナル「リセット」を受信してから1分経った後，シグナル「温度監視」を出力する．

これ以上分解しないデータ変換部分は，基本機能としてミニ仕様書を記述しておく．

5. おわりに

SA/SD法はそれほど厳密に定義されたものではない．特に，データフローダイアグラムの作成やパッケージ化の部分は，未だ工学よりもアート的であると言われている．つまり，誰が分析／設計しても同じ設計が得られるということはなく，分析／設計者が要所要所で適切な判断を下せるかどうかが，やはり重要である．

また，実際に人手あるいは機械によって動いているシステムを分析するのは容易だが，全くはじめて作るシステムには適用しにくいという点も指摘されている．

しかしこの方法は，適切にモジュール化され，文書化された「よりよい」分析／設計を手続き的に行うためのひとつの明確なガイドラインを与えるものであると言える．また，利用者がそれぞれの経験や知識を活かして，その人なりにSA/SD法の有効な利用を考えることも重要であろう．

CASEツールの利用によって，SA/SD法がより便利な道具として発展していくことを期待したい．

参考文献

[1] T. DeMarco (高梨, 黒田訳): “構造化分析とシステム仕様”, 日経マグロウヒル, 1986 (T. DeMarco: “Structured Analysis and System Specification”, Prentice-Hall, 1979).

[2] E. Yourdon and L. L. Constantine(原田, 久保訳): “ソフトウェアの構造化設計法”, 日本コンピュータ協会, 1986 (E. Yourdon and L. L. Constantine: “Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design”, Prentice-Hall, 1979).

[3] G. J. Myers (久保, 国友訳): “高信頼性ソフトウェア —複合設計”, 近代科学社, 1976 (G. J. Myers : “Reliable Software through Composite Design”, Mason/Charter, 1975).

[4] 久保未沙: “複合設計”, 情報処理, Vol.25, No.9, pp.935–945, 1984.

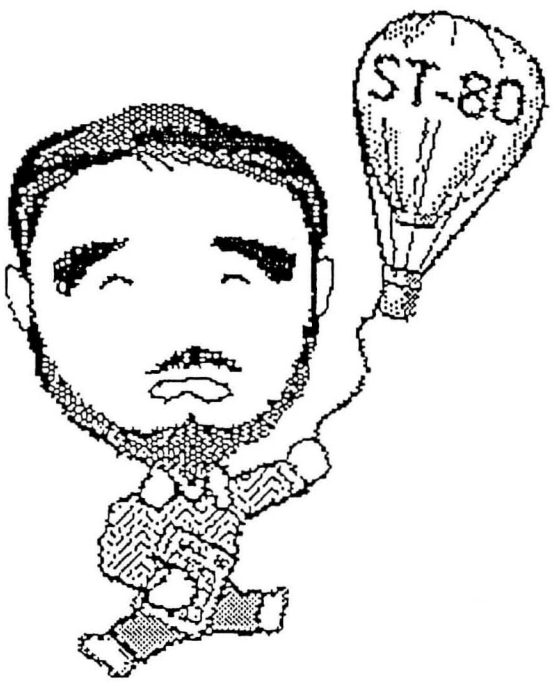
[5] D. T. Ross and K. E. Schoman Jr.(前川, 大筆, 松村訳): “要求定義のための構造的分析”, 国井利泰監修, “ソフトウェア工学 要求仕様技術”, pp.216–230, 共立出版, 1988 (D. T. Ross and K. E. Schoman Jr.: “Structured Analysis for Requirements Definition”, IEEE, Transactions on Software Engineering, Vol.SE-3, No.1, pp.6–15, 1977).

[6] P. T. Ward (立田種宏訳): “ワード氏のリアルタイムSA手法”, bit, Vol.20, No.5, pp.64–82, 1988 (P. T. Ward: “The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing”, IEEE, Transactions on Software Engineering, Vol.SE-12, No.2, pp.198–210, 1986).

[7] D. J. Hatley (立田種宏訳): “ハトレー氏のリアルタイムSA手法”, bit, Vol.19, No.14, pp.35–50, 1987 (D. J. Hatley: “The use of structured methods in the development of large software-based avionics systems”, American Institute of Aeronautics and Astronautics Inc., Paper No.84-2595, 1986).

[8] 梶谷, 伊藤, 松浦, 谷口, 嵩 : “オフィスワークの代数的記述と検証 — 図書購入業務の記述 —”, 電子通信学会技術報告, AL84-38 (1984).

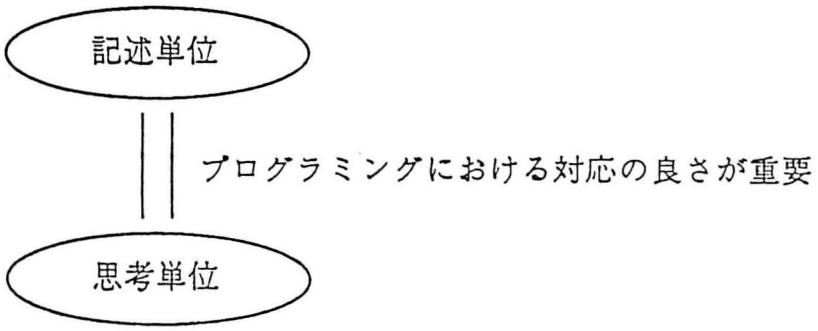
オブジェクト指向ソフトウェア開発
(OOA, OOD, OOP)



富士ゼロックス情報システム (株)
企画部付 Smalltalk-80特別プロジェクト (Blizzard-90)
青木　淳
160 東京都 新宿区 西新宿 3-16-6
TEL: 03 - 378 - 8011 (ex. 224)
FAX: 03 - 378 - 8145
MAIL: r2d2@bz90.fxis.fujixerox.co.jp

1. オブジェクト指向への潮流

ソフトウェアの開発のための多くの方法論は、思考単位(ThinkingUnit)と記述単位(ProgrammingUnit)の良好な対応を求めて変遷してきた。その中で、オブジェクト指向は、思考単位と記述単位の溝を埋める有力な方法論であるとの認識が深まっている。インプリメント言語も、データベースも、システム分析設計方法も、そして並列分散協調系も、オブジェクト指向の匂いがするようになってきている。

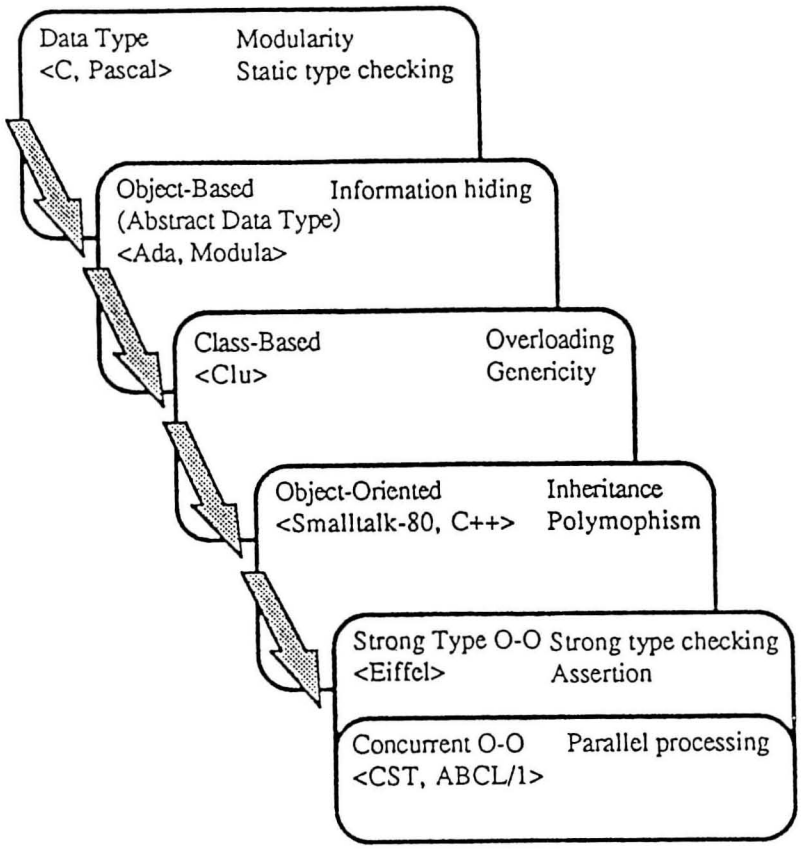


1.1. インプリメント言語

インプリメント言語の世界では、データ型(DataType)、データ抽象(ObjectBased)、抽象データ型(ClassBased)を経て、オブジェクト指向(ObjectOriented)と変化している。データ型としては、CやPascalなど。データ抽象や抽象データ型としては、AdaやModula-2そしてMesaなどがある。オブジェクト指向としては、Smalltalk-80があまりにも有名である。最近では、データ型の言語にオブジェクト指向のふりかけをかけたC++やObjective-Cなどのハイブリッド言語が登場していきている。1989年にCODASYLで、COBOLをオブジェクト指向に対応させるサブコミッティが創設され、1992年ぐらいには、COBOL++やOO-COBOLが出現するだろうとまで言われている。

オブジェクト指向言語の特徴は二つある。一つは、データとアルゴリズムそしてメッセージ送受信機能をパックして実現される「情報隠蔽」や「カプセル化」であり、いわゆる「抽象データ型(AbstractDataType)」である。もう一つは、再利用可能なソフトウェア要素の整理機構を実現する「委譲機構(delegation)」や「部分クラス(subclass)」であり、すなわち「インヘリタンス(inheritance)」である。簡単に表せば、「ObjectOriented = AbstractDataType + Inheritance」である。

オブジェクト指向言語以外で、データ抽象に対する支援を、こんなに包括的に与えてくれる言語は皆無である。さまざまな形式の抽象化をサポートしたと言われるAdaさえ、オブジェクト指向言語のサブセットである。Adaのパッケージ機能を利用すれば、情報隠蔽やカプセル化は実現できるが、データをプロセデュアに渡すという基本姿勢は変わらず、その上、パッケージをパラメータとして渡す機能を持っていないため、ソフトウェア要素の整理機構(インヘリタンス)が提供されない。オブジェクト指向言語は、抽象データ型を値として扱う以上の機能を持っているのである。



オブジェクト指向言語では、オブジェクト(データ)にメッセージ(プロセデュアの名前)を渡すことが基本である。これにより「ポリモフィズム(多相)」が実現できる。問題を解決するのは、関数やプロセデュアではなく、オブジェクト(データ)それ自身である。この「ポリモフィズム(多相)」を実現するためには、メッセージが送られるオブジェクトに従って、対応するプロセデュアを捜さねばならない。そのため、プロセデュアの名前とプロセデュア本体は、コンパイル時に束縛することができず、実行時に動的に束縛(遅延束縛)するのが普通である。Smalltalk-80では、これを「メソッドサーチ」と呼び、C++などでは、「オーバローディング」などと呼ばれている。

また、再利用を前提としたソフトウェア要素の整理機構(インヘリタンス)を有効に利用するためには、構造や機能の抽象化プロセスが重要であり、そこに登場するオブジェクトをはっきりとしたチャンクに分割し、一つの概念カテゴリを形成していかねばならない。それは認知的経済性と共有可能性の制約を受ける。認知的経済性とは、作成されたクラスライブラリの階層構造や組立構造が、問題ドメインをもっとも経済的に表現しているかということである。分析し設計し開発するプログラムには、認知的努力が最小になるような要請が常に課せられているはずである。共有可能性とは、作成されたクラスライブラリの階層構造や組立構造が、他のプログラム開発でも共有されなければならないことである。一個人よりも多くの開発者にとって意味を持つべきだという要請によって制約されている。

これは、インプリメント言語の範疇を越えている。むしろ、システム分析設計方法のところで議論されねばならないことであろう。オブジェクト指向言語は、従来の言語と異なり、システム分析設計方法までを云々しなければならない「高級なインプリメント言語」とみなすこともできる。思考単位と記述単位の溝が明らかに埋められている証拠である。筆者は、Smalltalk-80で6年の経験を持つが、自分が作成したプログラムのクラス階層構造(インヘリタンス)を他人に見せるということに恥ずかしさを覚えた経験がある。これは、自分の能力(概念カテゴリや抽象化プロセス)があからさまになるためであろう。

<参考文献>

E. W. Dijkstra, C. A. R. Hoare, O.J.Dahl 著, 野下, 川合, 武市 共訳 : 構造化プログラミング, サイエンス社, 1975.

B. H. Liskov, S. N. Zilles, "Programming with Abstruct Data Type", Proc. ACM SIGPLAN Conf. on Very High Level Languages, SIGPLAN Notices 9, 4, pp50-59, 1974.

A. Goldberg, D. Robson 共著, 相磯秀夫 監訳, "Smalltalk-80言語詳解", オーム社, 1987.

B. Cox, Object-Oriented Programming. Addison-Wesley, 1986.

D.Ingalls, "Design Principles Behind Smalltalk," Byte, August 1981.

P.Wegner, "Dimensions of Object-Based Language Design," ACM OOPSLA '87 Proceedings, October 1987.

P.Wegner, "Object-Orented Software Engineering", International Conference on Computer Languages '88, Tutorial Notes, October 1988.

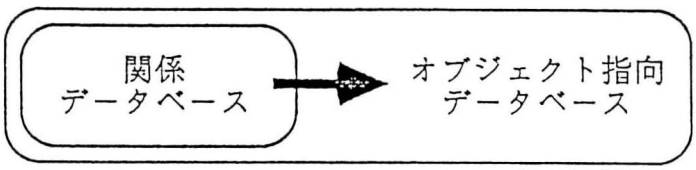
Scott Danforth, Chris Tomlinson, "Type Theories and Object-Oriented Programming," ACM Computing Surveys, Vol.20, No.1, pp.29-72, 1988.

青木淳, "オブジェクト指向プログラミング環境におけるインテグレーション過程," Proceedings of 10th Software Symposium, June 1990. または Proceedings of WOOC, 箱根 1990.

Iona Roth, John P. Frisby 著, 長町三生 監訳, "知覚と表象", 海文堂, 1989.

1.2. データベース

データベースの世界では、リレーショナル型が主流であるが、それが平面的(表)であり、複合オブジェクトが扱えないとか、汎化階層が利用できないとか、レコードそれ自身が識別子を持っていないなどの不満から、オブジェクト指向データベースへの期待が集まっている。1989年に京都でDOOD'89が開催され、オブジェクト指向データベース宣言が行なわれた。



このオブジェクト指向データベース宣言によれば、関係データベースのように、問題領域の実体を他と区別するために、属性値を用いる必要がない。これは「オブジェクト識別性(ObjectIdentity)」と呼ばれる。また、データとそれに対する操作が一体化されているために操作インタフェースの抽象化が可能となる。そして、オブジェクトをネストして「複合オブジェクト(ComplexObject)」を作り、問題領域の複雑な実体をもモデル化できること、汎化階層による部品化と差分プログラミングができるという利点がある。

オブジェクト指向データベースを開発するには、オブジェクト指向言語で扱われているオブジェクトに「永続性(Persistency)」を加える方法と、オブジェクト指向に沿ってデータベースそのものを構築していく方法がある。まだ、どちらも大変な作業を必要としている。オブジェクト指向データベースが、現在のリレーショナルデータベースに替わって利用されるのは、2000年代に入ってからではなかろうか。

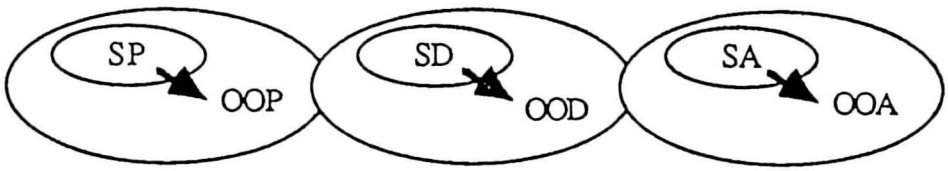
<参考文献>

Proceedings of DOOD (Deductive and Object-Oriented Database) 1989 Kyoto, Japan.

西尾章治郎, 田中克巳, 「オブジェクト指向データベース宣言」とその意義, bit, Vol.22 No.8, 1989.

1.3. システム分析設計方法

構造化手法やCASEの識者(Yourdon, DeMarco, Wassermanなど)が、オブジェクト指向手法へシフトしている現象が見受けられる。また、オブジェクト指向のソフトウェア開発手法を提唱する人(Coad, Booch, Mellorなど)も増加している。



構造化プログラミング(SP)、構造化設計(SD)、構造化分析(SA)に代わって、オブジェクト指向プログラミング(OOP)、オブジェクト指向設計(OOD)、オブジェクト指向分析(OOA)が台頭してきている。

今までのシステム分析設計の仕様化技法は、次の4つの観点に大別できる。

[1] 機能の構造に着目
[2] 情報の流れに着目
[3] 実体と関係に着目
[4] 事象の構造に着目

機能の構造に着目する仕様化技術の根本は、「機能分割法」である。機能は、数学的な関数として表され、上位の関数が、下位の関数を呼び出す形で、階層構造を用いてモデル化する。その目的は、多くの共通となる機能の割り出しを行なうことである。

情報の流れに着目する仕様化技術としては、データフロー図やデータ辞書そして決定表などに代表される「構造化分析法(DeMarco法)」がもっともポピュラーである。現在のCASEツールのほとんどは、この方法を基にしている。後に、リアルタイム用の機能が付加された「リアルタイム構造化分析法(Hatley法やWard法)」もある。DeMarco法とは記法が異なるが、やはり情報の流れに着目したRossの「SADT(Structured Analysis and Design Technique)」もある。

実体と関係に着目する仕様化技術は、データベースの概念設計方法の一つである。もともとCoddによる関係データベースのモデルから発想されている。問題領域の事物や事象などを表す「実体」と、それらの実体間を規定する「関係」を用いて、問題領域を分析する。その際には、実体を「箱」、関係を「菱形」で表し、それらを実線で結んだ「実体-関係図(ER図)」を使用する。この実体と関係に着目する仕様化技術は、HullとKingの「意味データモデリング」や、ShlaerとMellorの「インフォメーションモデリング(ShlaerとMellorはオブジェクト指向システム分析と呼んでいる)」へと発展する。

事象の構造に着目する仕様化技術は、制御系やOSなどの事象駆動型のプログラムの開発ともに発展してきた。事象発生により内部の状態を変化させるため、その記述に「状態遷移図」や「ペトリネット」などが利用された。HoareのCSP(通信逐次プロセス)に基づく「ジャクソン法(JSD)」も、事象の構造に着目する仕様化技術の一つとみなせる。

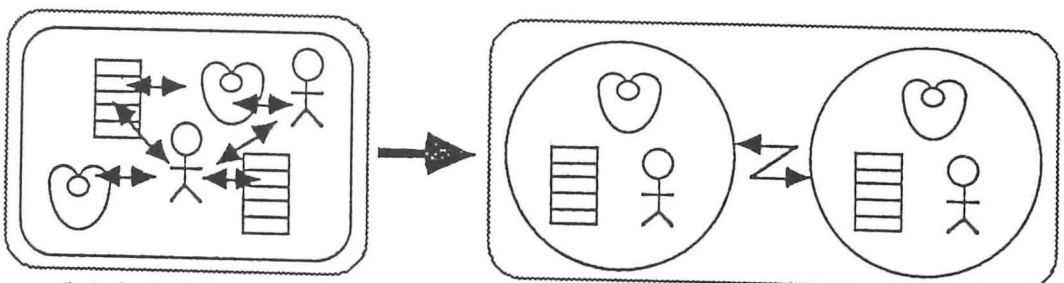
仕様化技術とは、一言で言ってしまえば、「複雑性を扱うための方法」である。オブジェクト指向の仕様化技術は、以下に示す観点に着目して複雑性を整理する。

[1] 抽象化
  [1.1] 手続きの抽象化
  [1.2] データの抽象化
[2] カプセル化
[3] 継承
[4] 組み合わせ
[5] コミニュケーション(メッセージによる)
[6] 組織化の方法
  [6.1] 実体(オブジェクト)と属性
  [6.2] 全体と部分
  [6.3] 部類と実例
[7] 規模
[8] 挙動の分類
  [8.1] 直接原因による分類
  [8.2] 時間推移による分類
  [8.3] 機能類似による分類

この観点から、オブジェクト指向の仕様化技術と上記の4つに大別された仕様化技術と比較してみる。

機能の構造に着目する仕様化技術では、手続きの抽象化には有効であるが、それ以外にはメリットがない。したがって、この仕様化技術から生まれたプログラムは、データの抽象化やカプセル化などを考慮することが難しく、データは手続きに従属して、なすがままになる。そのため、あるデータを扱っている手続きは、プログラム中に散在する。この場合、データ構造の変化は、散在した手続きの全てを見直す結果となる。日本のソフトウェア開発現場は、この機能の構造に着目する仕様化技術が主流である。

情報の流れに着目する仕様化技術では、手続きの抽象化、規模、直接原因による分類には有効である。それ以外は難しい。本質的には、機能の構造に着目する仕様化技術と同じ問題を引き起こす。しかし、挙動の分類の手段が与えられているので、影響が及ぶ範囲を検索できる。欧米のソフトウェア開発現場では、この情報の流れに着目する仕様化技術が主流である。上記の機能の構造に着目する仕様化技術と情報の流れに着目する仕様化技術を、オブジェクト指向の仕様化技術では、カプセル化され情報隠蔽されたオブジェクトの内部処理の記述のためだけ利用する。



みんなでグチャグチャ　　オブジェクトとして整理される

実体と関係に着目する仕様化技術では、組み合わせ、実体(オブジェクト)と属性、全体と部分、部類と実例の組織化の方法としては優れているが、それ以外の情報の流れの側面が弱くなってしまう。データベースを使用するソフトウェアの現場で頻繁に用いられているので、手続きの抽象化や挙動の分類そして継承などは考慮していない。オブジェクト指向の仕様化技術では、実体と関係に着目する仕様化技術の中から、汎用的なis-a関係とhas-a関係を取り出して、抽象化や継承そして組み合わせに利用していると考えることができる。

事象の構造に着目する仕様化技術では、継承と組織化の方法を除いては非常に有効な方法である。特に、時間推移による挙動分類では、オブジェクト指向の仕様化技術よりも威力を発揮する。カプセル化やメッセージによるコミニュケーションなどは、オブジェクト指向の仕様化技術と共通するところが多い。

<参考文献>

Yourdon, E. and Constantine, L.著, 原田, 久保 共訳, ソフトウェアの構造化設計法, 日本コンピュータ協会, 1986.
Constantine, L., "A modular Approach to Program Optimization," Computers and Sutomation, Vol. 16, No. 3, May 1967.
Hamilton, M. and Zeldin, S., "Higher Order Software - A Methodology for Defining Software," IEEE Transactions on Software Engineering, 1976.
DeMarco, T., Structured Analysis and System Specification, Prentice Hall, 1978.
Ross, D. T., "Structures Analysis (SA) A language for Commucating Ideas," IEEE Transactions on Software Engineering, 1977.
Chen, P. A., "The Entity-Relationship Model toward a Unified Approach of Data", ACM Transactions Database Systems, March 1976.
Hull, R. and King, R., "Semantic Database Modeling: Servey, Aoolications and Research Issues", ACM Computing Surveys, September. 1987.
Shlaer, S. and Mellor, S., Object-Oriented Systems Analysis, Prentice Hall, 1988.
J. L. Peterson, "Petri net theory and the modeling of systems", Prentice Hall, 1981.
M. A. Jackson, System Development, Englewood Cliffs, NJ, Prentice Hall, 1982
有沢, 山崎, 他 「マイケルジャクソンのシステム開発法」の連載, bit, 1990.
Booch, Grady, "Object-Oriented Development," IEEE Transactions on Software Engineering, February 1986.
Loomis, M., Shah, A., and Rumbaugh, J., "An Object Modeling Technique for Conceptual Design," European Conference on OOP, June 1987.
Seidewitz, Ed and Stark, Mike, "Towards a General Object-Oriented Software Development Methodology," Ada Letters, Volume 7, Number 4, 1987.
Meyer, Bertrand, Object-Oriented Software Construction. Prentice Hall, 1988.
Ward, Paul T., "How do Integrate Object Orientation with Structured Analysis and Design," IEEE Software, March 1989.
Coad, Peter and Yourdon, Edward, Object-Oriented Analysis. Prentice Hall, 1990.

1.4. 並列分散協調系

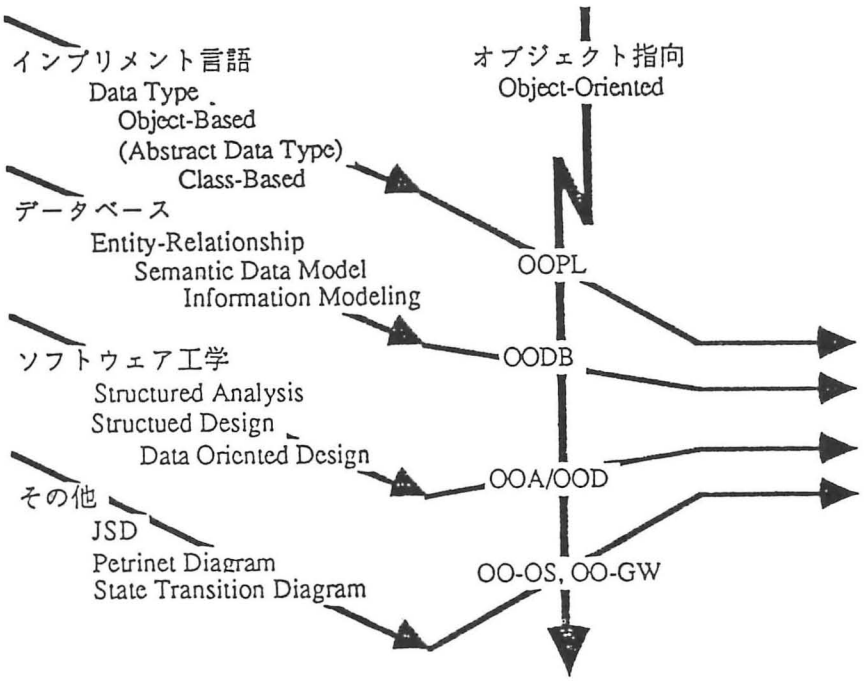
並列系や分散系そして協調系も、オブジェクト指向を取り入れている。並列プロセスや分散処理の実体として自己完備なオブジェクトが採用され、オブジェクト間のメッセージ通信をプロセスやエージェント間のコミュニケーションと考える。また、GroupwareやCSCWなどでも、オブジェクトの自立性を一人の人間とみなし、協調モデルを考えているものが多い。

<参考文献>

Yasuhiko Yokote and Mario Tokoro, "The Design and Implementation of ConcurrentSmalltalk," ACM OOPSLA '86 Proceedings, 1986.
Yonezawa, A., and Tokoro, M., Object-Oriented Concurrent Programming. MIT Press, Cambrige, Mass, 1987.
Geoffrey A. Pascoe, "Encapsulations: A New Software Paradigm in Smalltalk-80," ACM OOPSLA '86 Proceedings, 1986.

Hewitt, C., The challenge of open systems. Byte (Apr.), 232-369, 1985.
Kum-Yew Lai and Thomas W. Malone, "Object Lens: A "Spreadsheet" for Cooperative Work," CSCW'88, P.115.
Jeff Conklin and Michael L. Begeman, "gIBIS: A Hypertext TOol for Exploratory Policy Discussion," CSCW'88, P.140.

以上のような最近の「オブジェクト指向への潮流」をまとめると次の図のようになる。これは、オブジェクト指向と各分野の直交性の良さの証拠である。オブジェクト指向の手法は、これからのソフトウェアの開発の指導原理となるべきものである。

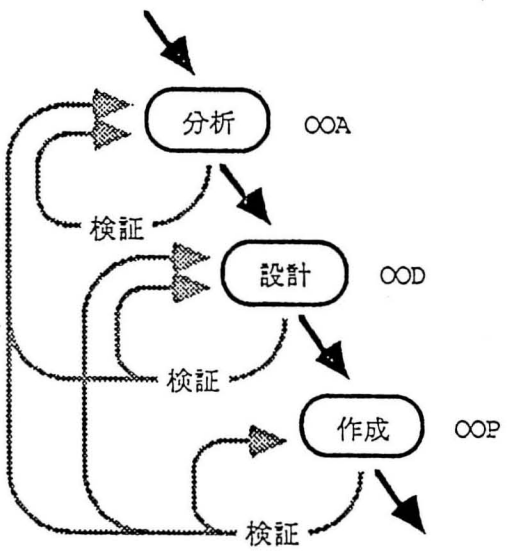


注意しておきたいのは、「オブジェクト指向の手法が従来の構造化の手法に取って替わるものではない。」ということである。むしろ、構造化の手法を内包して、より大きく強力になった手法とみなすほうが良い。

2. オブジェクト指向ソフトウェア開発

従来のソフトウェア開発は、上流工程から下流工程に向かって、作業の後戻りを許さないウォータフォール型で進められている。ところが、オブジェクト指向ソフトウェア開発では、作業の後戻りを極力避けるようにするのではなく、むしろ積極的に作業の後戻りを行なって進行する。したがって、その作業現場であるオブジェクト指向環境は、頻繁な試行錯誤に耐えられなければならない。その意味で「究極のバッチ・システム」とも呼ばれる。

その作業工程を次に示す。OOA->OOD->OOPへの流れはあるものの、オブジェクト指向ソフトウェア開発では、検証と書かれて後戻りしている工程が重要な役割を果たす。検証して間違いを発見し、直したり改良したりというメンテナンス作業がシステム開発そのものだと位置づけている。従来のメンテナンス作業は、日陰の嫌な仕事というイメージが強いが、オブジェクト指向の作業環境では、日陰の嫌な仕事こそが開発の主体である。



このオブジェクト指向ソフトウェア開発の科学哲学的背景は"Error elimination is one of the driving forces behind evolution."「間違いの削減は進化へ導く推進力のひとつである。」というゲルハルト・フッシャーの考え方である。この考え方の起源は、カール・ポパーの「科学的発見の論理」である。フッシャーは、より良いソフトウェアを開発するためには、知識増幅器(intelligence augmentation)が重要であり、それこそがコンピュータの目指すところであると主張している。

かつて物理学者のアインシュタインは、「鉛筆は私よりも賢い。」と言った。この意味は、鉛筆を持たない私より、鉛筆を持った私の方が、はるかに賢いということである。鉛筆がアインシュタインの能力を増幅し、彼を明晰にしたのである。現在、この鉛筆に替わるものがコンピュータであり、オブジェクト指向ソフトウェア開発環境であり、OOAとOODのためのCASEツール(次世代CASEツール)である。

<参考文献>

Gerhard Fischer, "Human-Computer Interaction in Software: Lessons Learned, Challenges ahead", IEEE Software, January 1989.

Gerhard Fischer, "Cooperative Knowledge-Based Design Environments for the Design, Use and Maintenance of Software", Proceedings of 10th Software Symposium, June 1990.
カール·R·ポパー著, 大内 森 訳, 「科学的発見の論理(上下)」, 恒星社厚生閣, 1971.
青木淳, "オブジェクト指向プログラミング環境におけるインテグレーション過程," Proceedings of 10th Software Symposium, June 1990. または Proceedings of WOOC, 箱根 1990.
Coad, Peter and Yourdon, Edward, Object-Oriented Analysis. Prentice Hall, 1990.

2.1. OOAとOODそしてOOPのプロセス

オブジェクト指向ソフトウェア開発のプロセス(作業工程)の内容を示す。上述したように、決して上から下へスムーズに進むのではない。

オブジェクト指向分析(OOA)
1. オブジェクトの確定
2. 構造の確定
3. サブジェクトの定義
4. 属性の定義
5. サービスの定義

オブジェクト指向設計(OOD)
1. ハードウェア·アーキテクチャの決定
2. ソフトウェア·アーキテクチャの決定
3. 特殊化
4. 組立て
5. 抽象化
6. 要素分解
7. サービスの詳細化

オブジェクト指向プログラミング(OOP)
1. オブジェクト指向設計仕様からの
   ソース·コード·テンプレートの生成
2. 実際のサービスのインプリメント
3. 性能測定

検証して間違いを発見し、直したり改良したりというメンテナンス作業こそがソフトウェア開発なので、ウォークスルー通したり、支援ツールを使用しながら、上記のプロセス毎に成果を反証していく姿勢が必要がある。

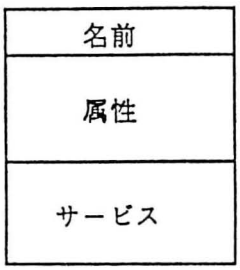
2.2. オブジェクトの確定

問題領域に登場するオブジェクトを見つけだすことである。実際の確定は、分析者や設計者が直感を働かせて見つけるものである。そのHow toは以下の通りである。

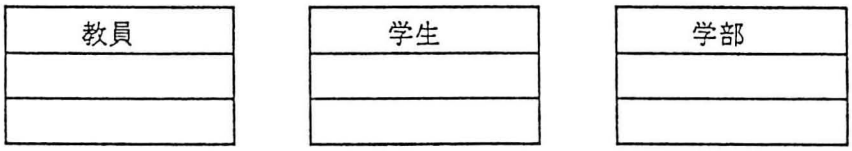
何処を観るか: 問題領域、テキストや図版、ユーザとの対話
何を捜すか: 構造、他のシステム、装置、事象、役割、場所、組織単位

何を考えるか: 必要な記憶やサービス、共通属性、共通サービス、本質的な要求
何を排除するか: 不必要な記憶やサービス、単一のインスタンス、派生した結果
名前は: 単数名詞、形容詞+名詞、標準用語、読みやすい名前

オブジェクト指向の真髄は、オブジェクトを擬人化する見方にある。オブジェクトがある意味で人間(問題解決能力を保有するもの)のようにみなせることである。オブジェクトを表すには、名前と属性そしてサービスの3つのフィールドを持つ記法を使用する。



たとえば、大学をモデル化してみよう。そこには、教員、学生、学部などが存在する。この段階では、属性やサービスを定義する必要はなく、登場するオブジェクトの名前を決定する。



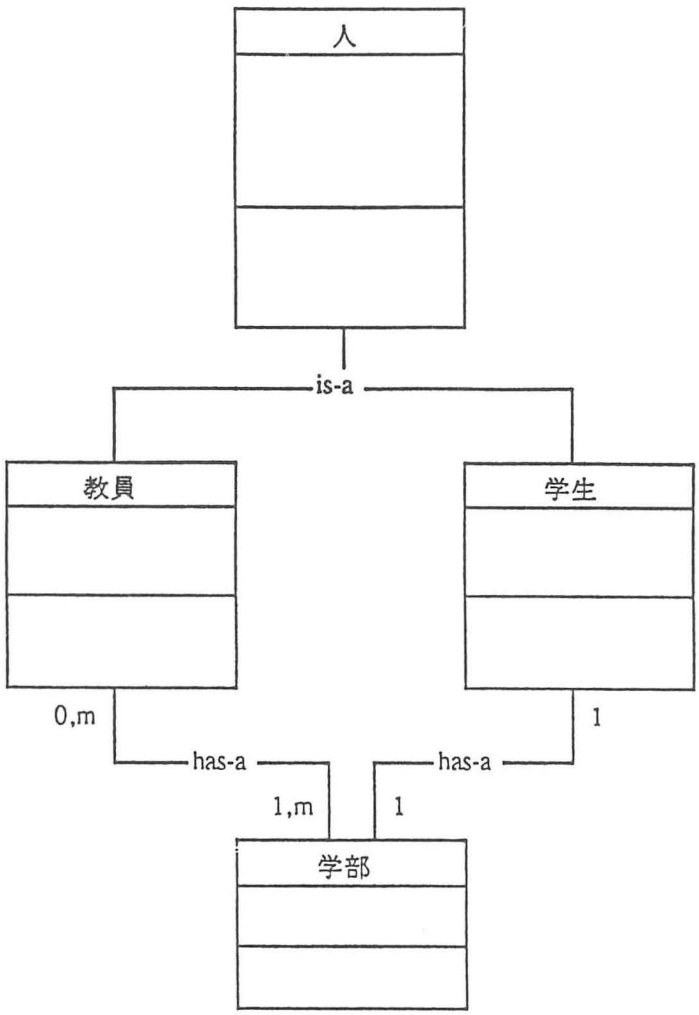
2.3. 構造の確定

構造とは、問題領域の複雑性を表現するものであり、オブジェクト指向では2種類存在する。

Classification Structure: 一般化と特殊化を反映したクラスの分類を表現(is_a関係)
(Generalization-Specialization Structure)
Assembly Structure: 全体と部分を反映した集団を表現(has_a関係)
(Whole-Part Structure)

オブジェクト間の関係に着目して、その中から「is_a関係」と「has_a関係」を切り出していく。このアプローチは、実体と関係に着目する仕様化技術である「実体関係によるモデリング」や「意味データモデリング」そして「インフォメーションモデリング」などに類似している。オブジェクト指向の構造の確定では、これらのモデリングの関係構造を汎用的なis-a関係とhas-a関係へと分類していくことに相当する。

先ほどの大学のモデリングでは次のようになる。



2.4. サブジェクトの定義

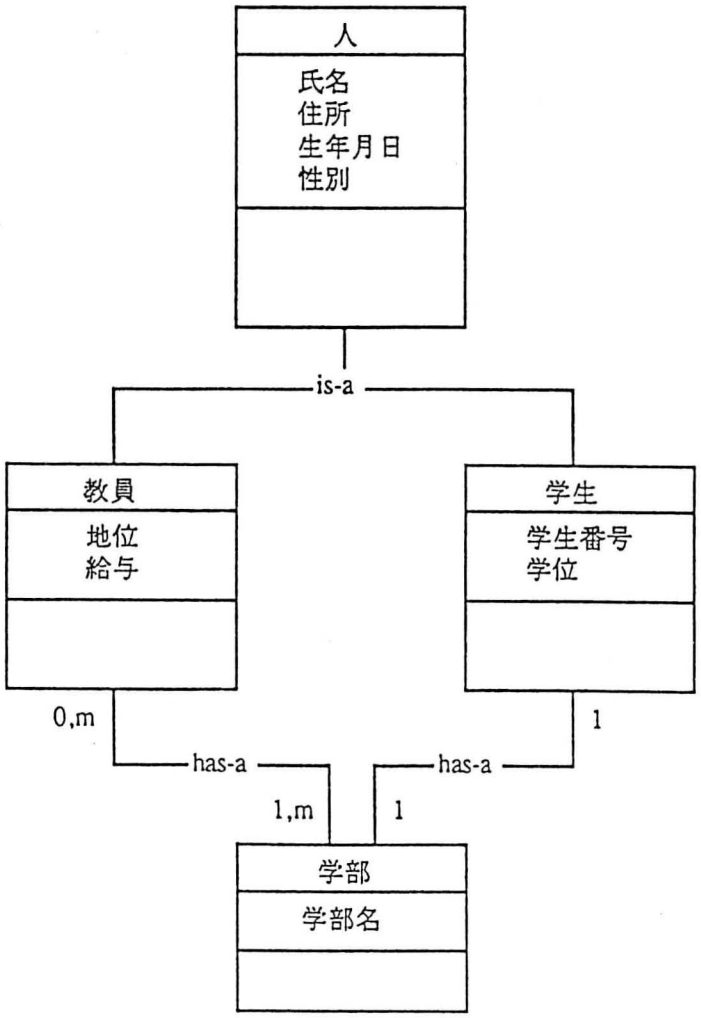
サブジェクトとは、オブジェクトの概観を表すものであり、様々なオブジェクトをチャンクとしてまとめ抽象化して表現することである。ちょうど、Smalltalk-80の「Model-View-Controller」が、それぞれサブジェクトに対応する。Smalltalk-80では、アプリケーションを構築する際に、モデルというデータ管理を専門に行なうオブジェクトや、ビューという表示まわり行なうオブジェクト、そして、コントローラという指示まわりを行なうオブジェクトを、数多く使用する。ところが、それらの数多くのオブジェクトを、モデルとビューそしてコントローラというチャンクに分類して考えることが、サブジェクトを定義していることに他ならない。

小さなソフトウェアの開発では、このサブジェクトの定義をする必要のないものが存在する。たとえば、先ほどの大学の例では、サブジェクトは必要ないであろう。

2.5. 属性の定義(インスタンス結合)

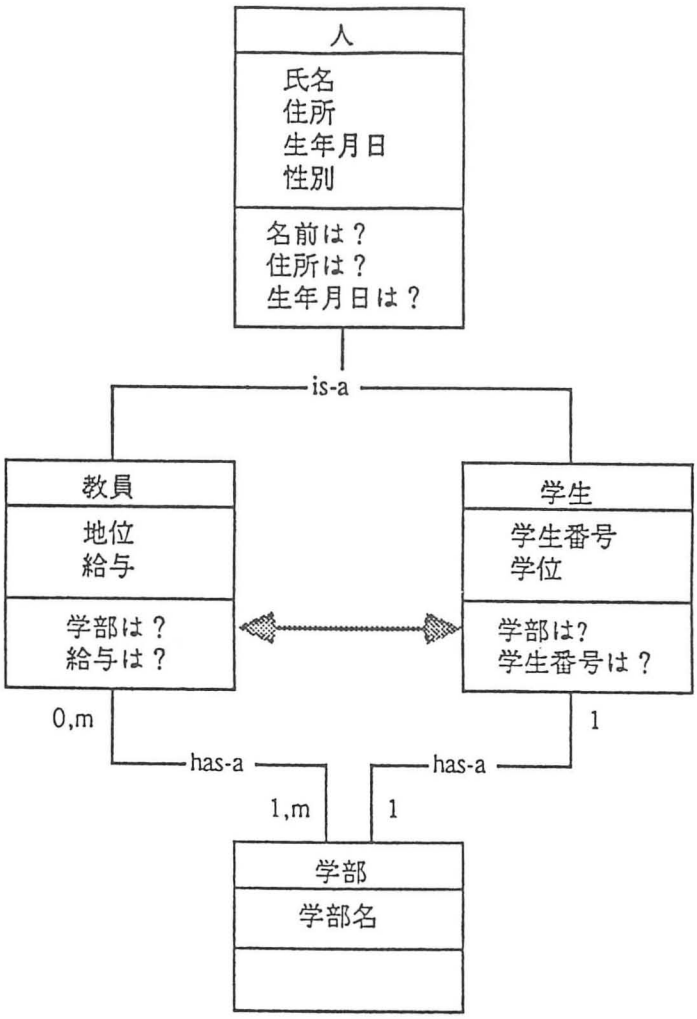
オブジェクトを特長づける属性(内部状態)を定義することである。その際に、インスタンス結合も考える。全体と部分を反映したAssembly Structureを確定する際に、オブジェクト間の多重度や必須度などを付加したが。インスタンス結合は、全体と部分の関係にないオブジェクト同士でも、問題領域に登場する数

などに対応関係がある場合に付加する。



2.6. サービスの定義(メッセージ結合)

オブジェクトが提供するサービス(メッセージを受け取った時に実行される処理)を定義をすることである。メッセージインタフェースの確定まではいかないが、大まかな処理の概要をテキストで明らかにする必要がある。そして、サブジェクト間のメッセージ交換経路に則って、他のオブジェクトとのメッセージ結合を考える。



2.7. ハードウェア･アーキテクチャの決定

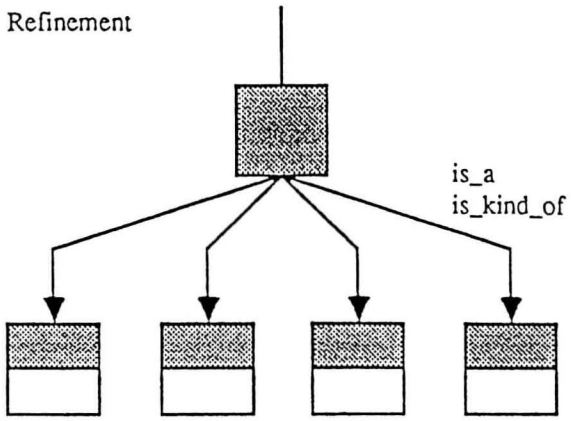
オブジェクト指向の言語および環境がすでに搭載されているハードウェアが望ましい。

2.8. ソフトウェア･アーキテクチャの決定

オブジェクト指向の言語および環境に加え、クラスライブラリが豊富なものを採用するのが良い。

2.9. 特殊化(Refinement)

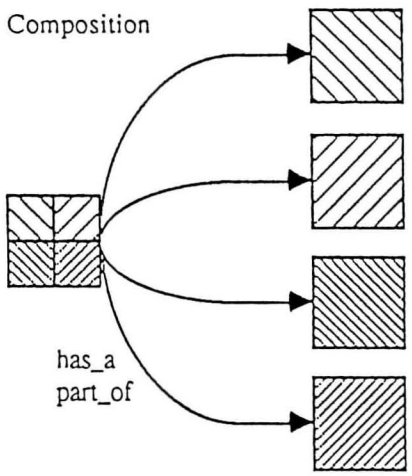
プログラムは、何もないとこから作り始めるのではなく、すでに存在するプログラムを改良するという形で進める。オブジェクト指向プログラミング環境では、このために、多くの型定義(クラス)の間に存在する類似性を整理できるインヘリタンス機構(継承機構)が具備されている。インヘリタンス機構は、ツリー構造(単一継承)やラティス構造(多重継承)があるが、プログラムを分類してライブラリ化するという目的は同一である。この機構を利用して、すでに存在するクラスの特殊化や改作を行なうのがRefinement過程である。Refinementは、最初に試みるオブジェクト指向プログラミング技術である。



Refinement過程は、挙動(機能)や状態(構造)が良く似た上位に位置するクラス(スーパクラス)を捜し出すことから始まる。そのために、既存プログラムを効率良く検索するシステムが不可欠である。Smalltalk-80のブラウザのようなものが理想的である。しかし、設計仕様からプログラム検索するには不十分で、文書検索に似た技術開発が望まれる。スーパクラスが決定したら、その下にサブクラスを作成する。その構造は、is_a, is_kind_of関係で結ばれる。インヘリタンスには、機能インヘリタンス(機能継承:上位の保持する挙動またはプロトコルを継承する)と構造インヘリタンス(構造継承:上位の保持する状態または内部変数を継承する)があるが、どちらのRefinementも多く行なわれる。

2.10. 組立て(Composition)

複雑なオブジェクトは、構成する部分オブジェクトのいくつかを、全体としてまとめる形で作成する。この組立て過程がComposition過程である。これは、前述のインヘリタンス機構を利用したクラス階層構造とは異なり、あるクラスのインスタンス変数(内部変数, 私的領域)に、他のクラスのオブジェクトを束縛することである。このCompositionは動的であり、前述のRefinementは静的である。動的というのは、プログラムの実行中に、頻繁に更新がかかるという意味である。Refinementのようなクラス階層構造は、静的で、プログラムの実行中に、スーパクラスが変化することは少ない。



Compositionの例としては、「家は屋根や床そして柱などから構成される。」というようなものである。屋根や床などのクラスを作成して、それらのクラスのインスタンスを、家のクラスのインスタンスが束縛して参照する。これらは、has_a, part_of関係で結ばれ、全体-部分の関係(Whole-Part関係)を表す。実際の複雑なオブジェクトは、この全体-部分の関係が幾重にもネストされて構成されている。これは、データ構

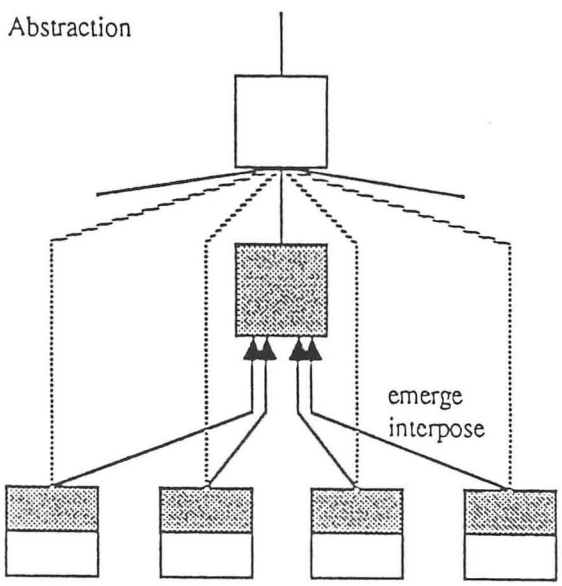
造の分解を支援し、データ型の抽象化レベルの違いにより、レイアを切り分けるという利点を生む。全体を知っていれば十分である場合には、部分の詳細を知る必要がない。逆に、全体を知らなくとも、部分だけで十分な場合もある。このことにより、トップダウン的なアプローチとボトムアップ的なアプローチのどちらにも適応できると言える。

Composition過程において、全体を表すクラスは、最小限の挙動(機能)や状態(構造)を有するクラスを、スーパクラスにすることがある。これはSmalltalk-80で、ObjectまたはModelというクラスのサブクラスとして、新たなクラスを作成することと等しく、既存のプログラムの中に継承できるクラスがないような場合である。もちろん、Refinementの際に、新たなインスタンス変数(内部変数)を定義して、そこからCompositionを行なうことも多い。

また、インスタンス変数に他のオブジェクトを束縛しないが、あるクラスが計算をするために、他のクラスや他のオブジェクトを使用することがある。これもCompositionの変形とも考えられる。has_a, part_of関係と言うよりは、むしろcall, invokeの参照関係である。このCompositionは、挙動(機能)のみであり、状態(構造)のCompositionでないことが多い。

2.11. 抽象化(Abstraction)

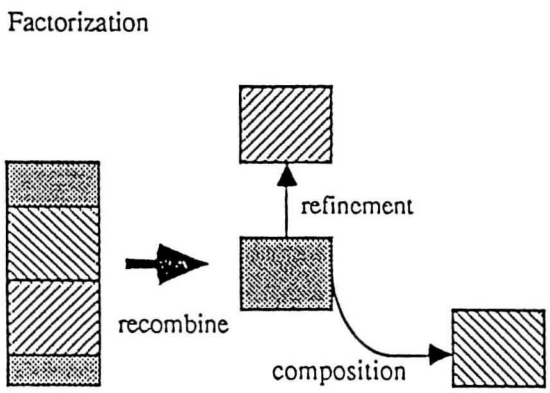
Abstraction過程は、抽象クラス(抽象型)を作成するプロセスである。これは、オブジェクト指向プログラミングの中で、いちばん難しい技術である。あるクラスのRefinementで生まれたいくつかの具体的なクラス(ConcreteClass)の中から、ある共通の挙動(機能)や状態(構造)のパターンを見つけ出す。その共通のパターンを実現する新たなクラスを作成し、それぞれの具体クラスの上に持ち上げ、元のスーパクラスとの間に差し入れる。この差し入れたクラスを抽象クラス(AbstractClass)と呼ぶ。このAbstraction過程は、コード量と冗長な表現の削減、理解度の向上、再利用性とメンテナンス性を高めることが目的である。



一般に、Abstraction過程において、機能インヘリタンスの形成の方が、構造インヘリタンスの形成よりも容易である。概して、抽象クラスは状態(構造)を持たず、多くの共通となる挙動(機能)を持つ。それらは、内部メッセージへたらい回しされて、サブクラスの主要なメッセージに帰着することが多い。様々なクラスに散在していたポリモフィックなメッセージの統合も、機能インヘリタンスの形成で行なわれる。

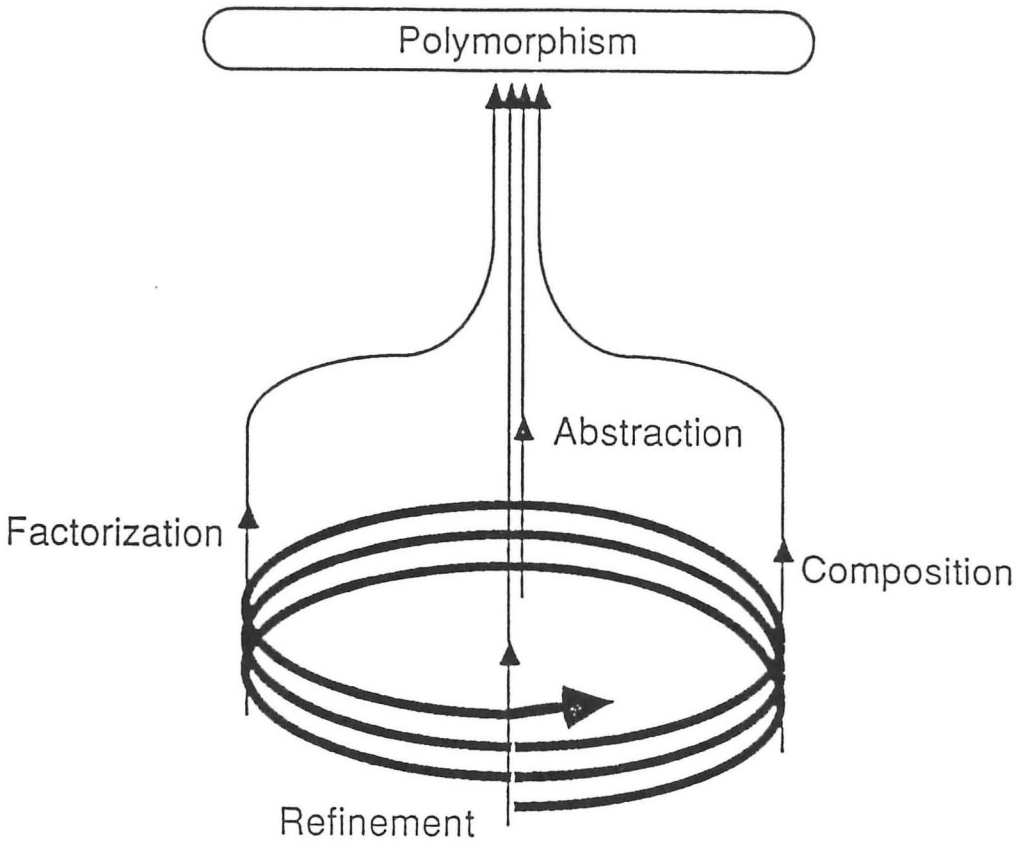
2.12. 要素分解(Factorization)

Factorization過程は、大きくて複雑なオブジェクトの内部状態や機能を分解して、それらの再使用および部品化を行なう目的で実施する。まず、分離しても再利用できる挙動(機能)や状態(構造)を識別する。そして、それらをモジュール化するために、小さなクラスを定義する。一方、大きくて複雑なクラスから、小さなクラスの機能を除去し、元々の機能や構造を達成するためにRefinementとCompositionを利用して再構成する。最後に、新たに作成した小さなクラスの共有や再利用を、他のオブジェクトについても可能な限り行なう。この一連の作業がFactorization過程である。



例えば、Compositionを利用するFactorization過程の例として、「ウインドウは、枠幅, 枠の色, 内部の色などの多くの部分オブジェクトで構成されている。」とする。これをFactorization過程を通して要素分解すると、「枠幅, 枠の色, 内部の色のみを有する枠というクラスを作成する。」そして「ウインドウは、枠というクラスのインスタンスを束縛する。」ということになる。要素分解して切り出された枠というクラスは、ウインドウに限らず、スクロールバーや、枠付きの画像など、様々なものに再利用することができる。

オブジェクト指向プログラミングのインテグレーション初期過程では、RefinementとCompositionの試行錯誤が頻繁に行なわれる。そして、プロトタイプが完成し、オブジェクト指向プログラミングのインテグレーション後期過程へ移行する。そこでは、AbstractionとFactorizationの試行錯誤が頻繁に行なわれる。AbstractionとFactorization過程は、これと拮抗するRefinementとCompositionの過程へ知らず知らずのうちにシフトし、これらの過程が螺旋的に繰り返され、ポリモフィズム(多相)最大の方向に向かってインテグレーションの度合いが高まって行く。



オブジェクト指向プログラミングのための環境は、この頻繁な試行錯誤に耐えれるシステムでなければならない。AbstractionとFactorization過程は、再構成や改編の作業と性質が似ている。何回もの再構成や改編にスムーズに対応できなければ、オブジェクト指向プログラミング環境とは言いがたい。言語のみでは、その中に内在するオブジェクト指向プログラミングの能力を有効に引き出すことはできない。必ず強力なプログラミング環境とペアになっていなければならない。設計課程のプロセスでのこのような作業を踏まえて、ハードウェア·アーキテクチャやソフトウェア·アーキテクチャの決定は、この点に注意すべきである。

とかく日本では、「オブジェクト指向」という言葉だけが先行し、オブジェクト指向型のプログラミング言語で作成されたものが、すべてオブジェクト指向のプログラムであるかのような錯覚さえ覚える。オブジェクト指向プログラミングの真価は、この何回もの再構成や改編の試行錯誤を行なうというインテグレーション過程を経て、はじめて発揮されるのであって、プログラミング言語に由来するものではない。言語と環境が統合されて、オブジェクト指向の方法論が議論でき、作成したプログラムのインテグレーションを高め、部品化や再利用性を向上させることができるのである。

2.13. サービスの詳細化

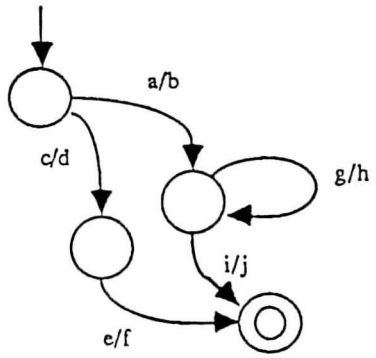
メッセージを受け取った時に実行される処理を定義にしたがって詳細化して記述することである。実際には次に示すようなことを行なう。

メッセージ·インタフェースの確定
プレ·コンディションの確定
ポスト·コンディションの確定

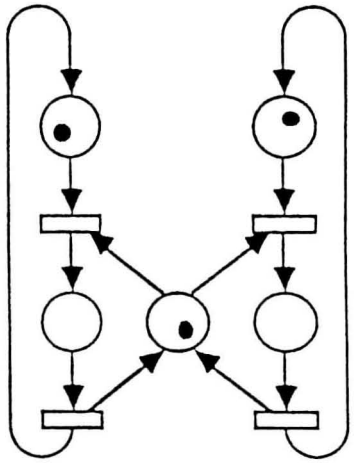
オブジェクトの状態を考察
メッセージのタイミングを考察

このために、状態遷移図やペトリネット図など、必要であれば、データ・フロー図やコントロール・フロー図などを使用すると良い。

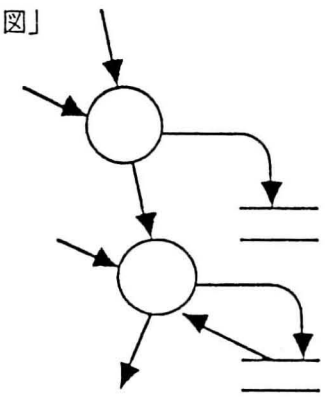
「状態遷移図」



「ペトリネット図」

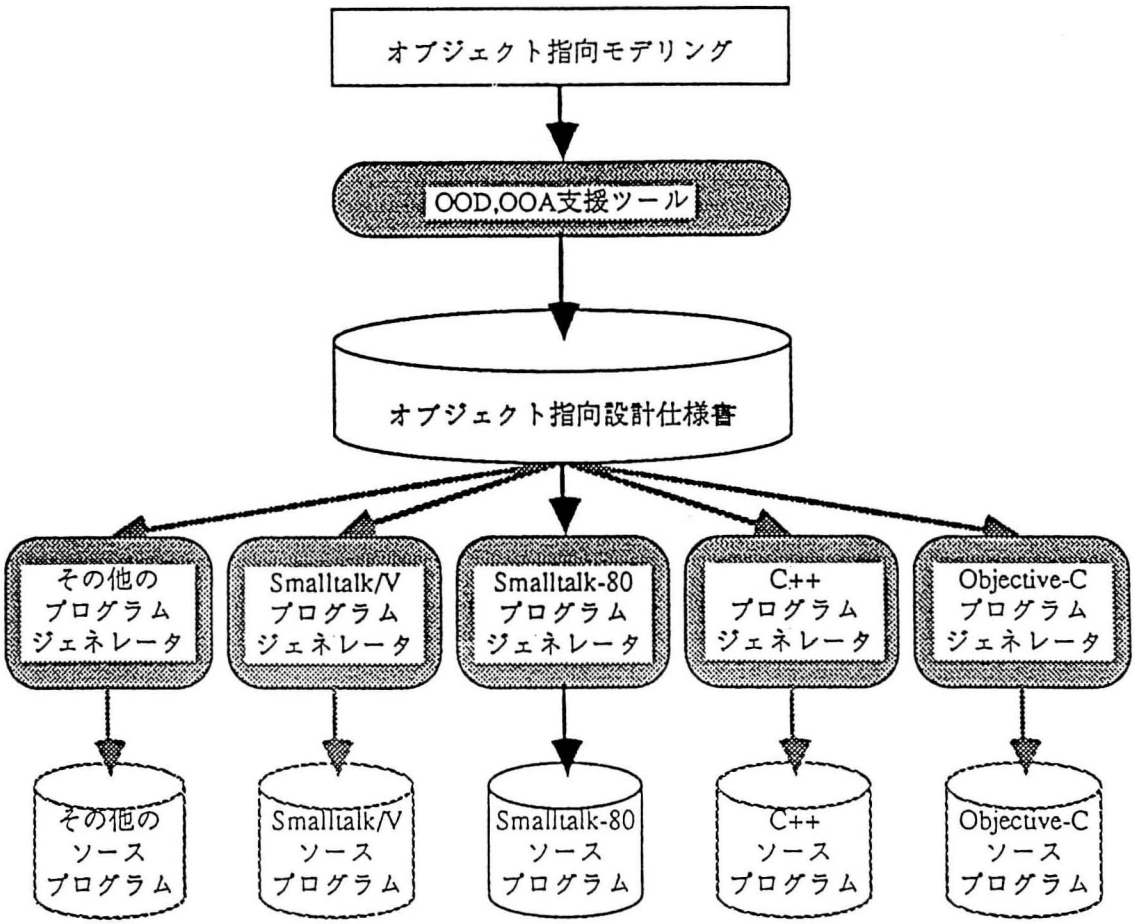


「データフロー図」



2.14. ソース・コード・テンプレート生成のためのツール

オブジェクト指向設計仕様から、ソース・コード・テンプレートを生成することである。メッセージ・インタフェースのプレ・コンディションとポスト・コンディションをコンストレイント・チェッキングとして埋め込んだテンプレートを生成する。

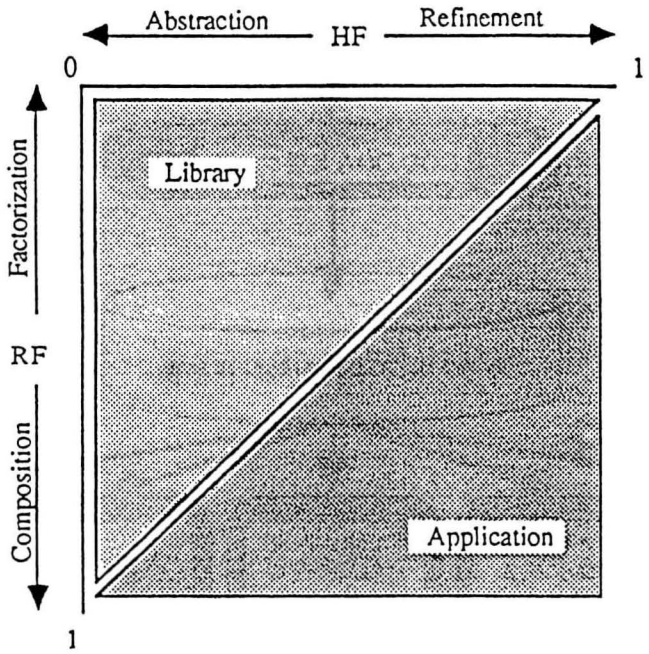


2.15. 性能測定

どのオブジェクトに負荷が集中しているか、どのサービス(メッセージ)がボトルネックになっているかを調査する。また、設計や分析がうまく行ったかを測定する。

Smalltalk-80では、MssageTallyのspyOn:の機能を利用して、負荷集中やボトルネックを調査できる。メトリクスとしては、「特殊化-抽象化のファクタ(HF)」と「組み立て-要素分解のファクタ(RF)」そして「ポリモフィズムのファクタ(PF)」を使用すると良い。

```
----------
 - 45 tallies.
**Tree**
100.0 BZBlizzard class test1
 97.8 Integer timesRepeat:
   91.1 primitives
   6.7 [] in BZBlizzard class test1
     6.7 Form displayOn:at:clippingBox:rule:mask:
       4.4 Form copyBits:from:at:clippingBox:rule:mask:
**Leaves**
91.1 Integer timesRepeat:
----------
```



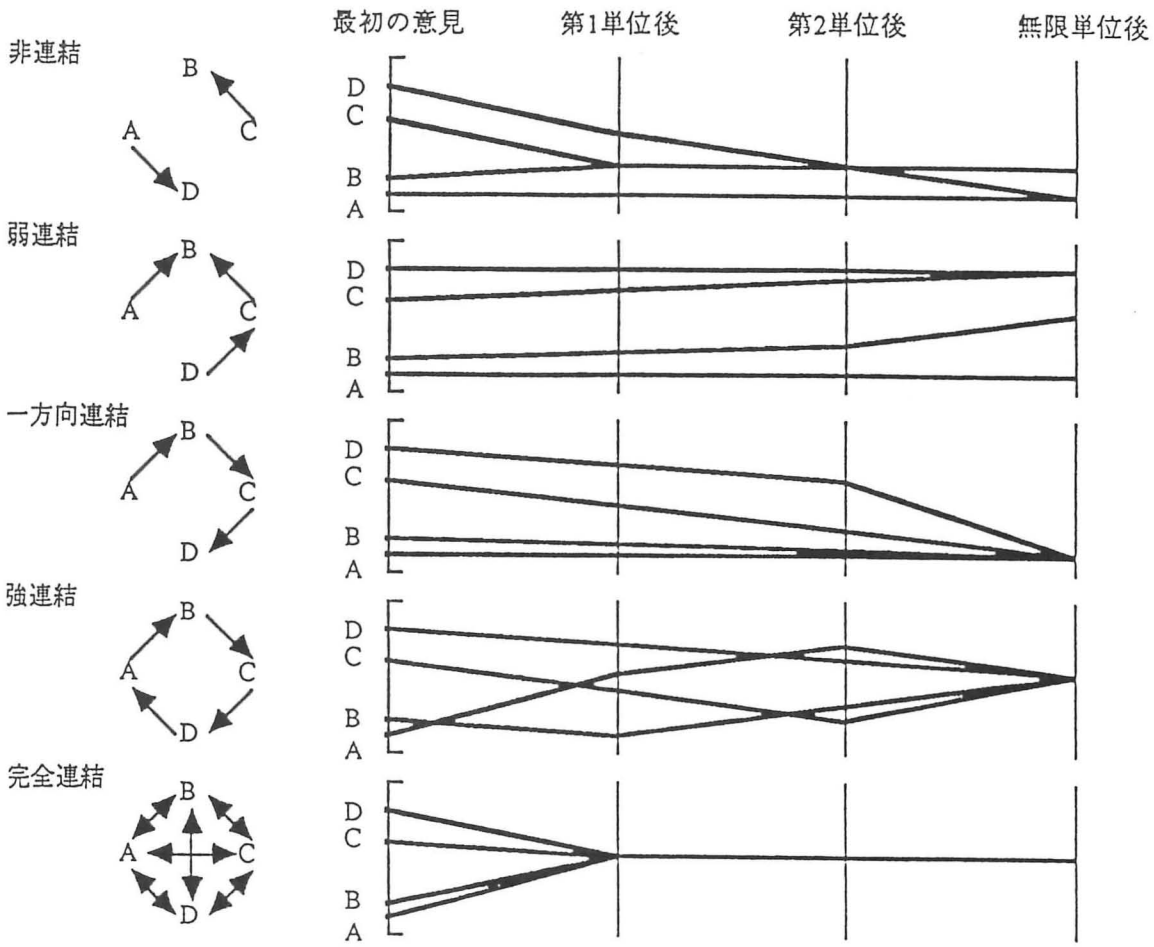
2.16. ウォークスルー

最後に、決められた記法で書かれた資料をもとに、ウォークスルー(レビュー)やプレゼンテーションを実施する方法を述べる。次に示す5つのステップ毎にウォークスルーを開き、懐疑的で批判的な検証(試行錯誤または間違いの削減)を行なう。

Subject Layer　オブジェクトのチャンクを表現する
Object Layer　チャンク別に登場オブジェクトを表現する
Structure Layer　抽象化-特殊化と全体-部分の構造を表現する
Attribute Layer　オブジェクトの内部状態を表現する
Service Layer　オブジェクトのサービス(挙動)を表現する

レビューする資料は、オブジェクト図とも呼ぶべきものであり、5つのレイヤごとに詳細化の部分が異なる視点を与える。ウォークスルーの参加メンバは、開発メンバ全員である。すなわち、分析者、設計者、作成者、検証者、そしてエンド･ユーザを含む「ジェルド･チーム(情報連結集団)」そのものである。

ウォークスルーでは。今日のことが明日くつがえされる。素早くプロトタイプできる環境が必要であり、上下の区別がなく、行動のスピードが大切である。この集団の構成メンバは、完全に情報連結し、ジェルのように形がないチームでなければならない。Frenchの論文の中に「勢力構造と意見の一斉化過程」という興味深い結果がある。それを図に示す。A,B,C,Dは構成メンバを表し、矢印は勢力の方向を表す。A---->Bは、AがBに対して大きな影響力を持っていることを示す。



階層構造を持つ組織は、「一方向連結」の形であり、意見の一斉化過程は、時間がかかりながら、勢力の強い人の意見に傾いていく。ジェルのように形がないチームは、「完全連結」の形であり、短時間で、リーズナブルな意見に一斉化する。「非連結」や「弱連結」の形では、いくら時間をかけても、意見の一斉化は達成できず、力が発揮できるわけがない。かといって、階層構造を持つ「一方向連結」の組織も、発展と進化を望むならナンセンスである。階層構造を持つ「一方向連結」の組織では、すったもんだしたあげく、結局のところ、上長の意見になるので、上長の技術レベルが、その集団の技術力になる。しかし、技術革新の早いコンピュータ業界で、上長がいつも高い技術レベルを保持できるかは疑問である。技術革新を望み、知識と情報をサービス財として提供するのであれば、少なくとも「強連結」、理想的には「完全連結」のチームが不可欠になる。

3. オブジェクト指向ソフトウェア開発の実際

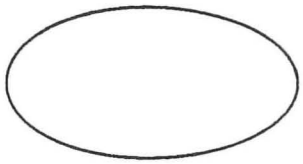
問題：

ウインドウによるテキスト編集のためのエディタを作成せよ。テキストは文字列だけではなく、書体を変えられるようにしなければならない。また、文字挿入位置や文字の選択は、マウスにより行なえるようにする。その上、ウインドウに閉じ込められたテキストは、単語の途中で改行が入らないように、単語ラッピングすること。編集操作は、ポップアップメニューにより行なうことを原則とする。

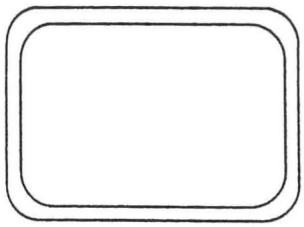
3.1. オブジェクトの確定

'This is sample string'
編集文字列

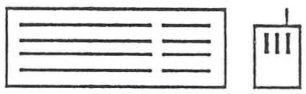
Text ('This is sample string')
編集テキスト



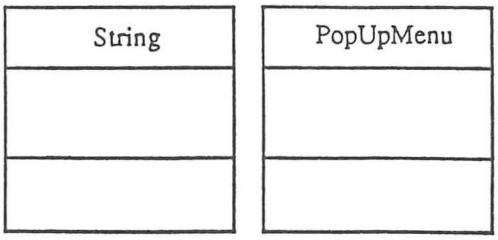
データ管理プログラム

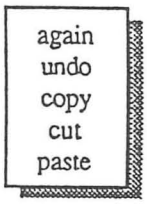


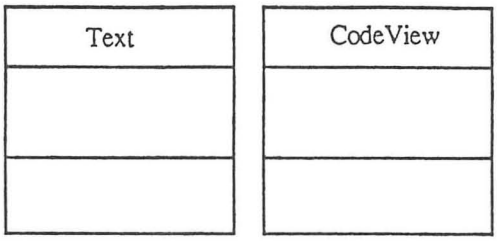
ディスプレイ

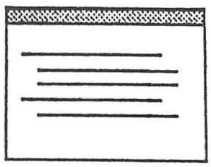


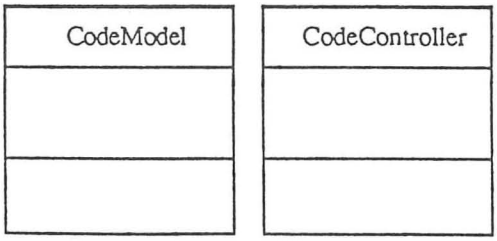
キーボード&マウス

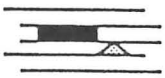


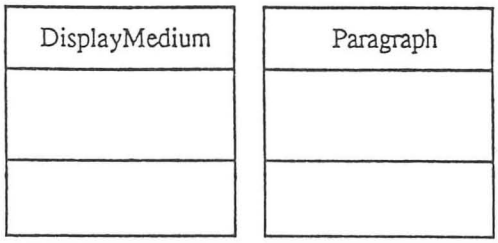


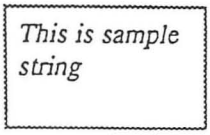


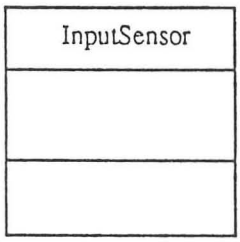




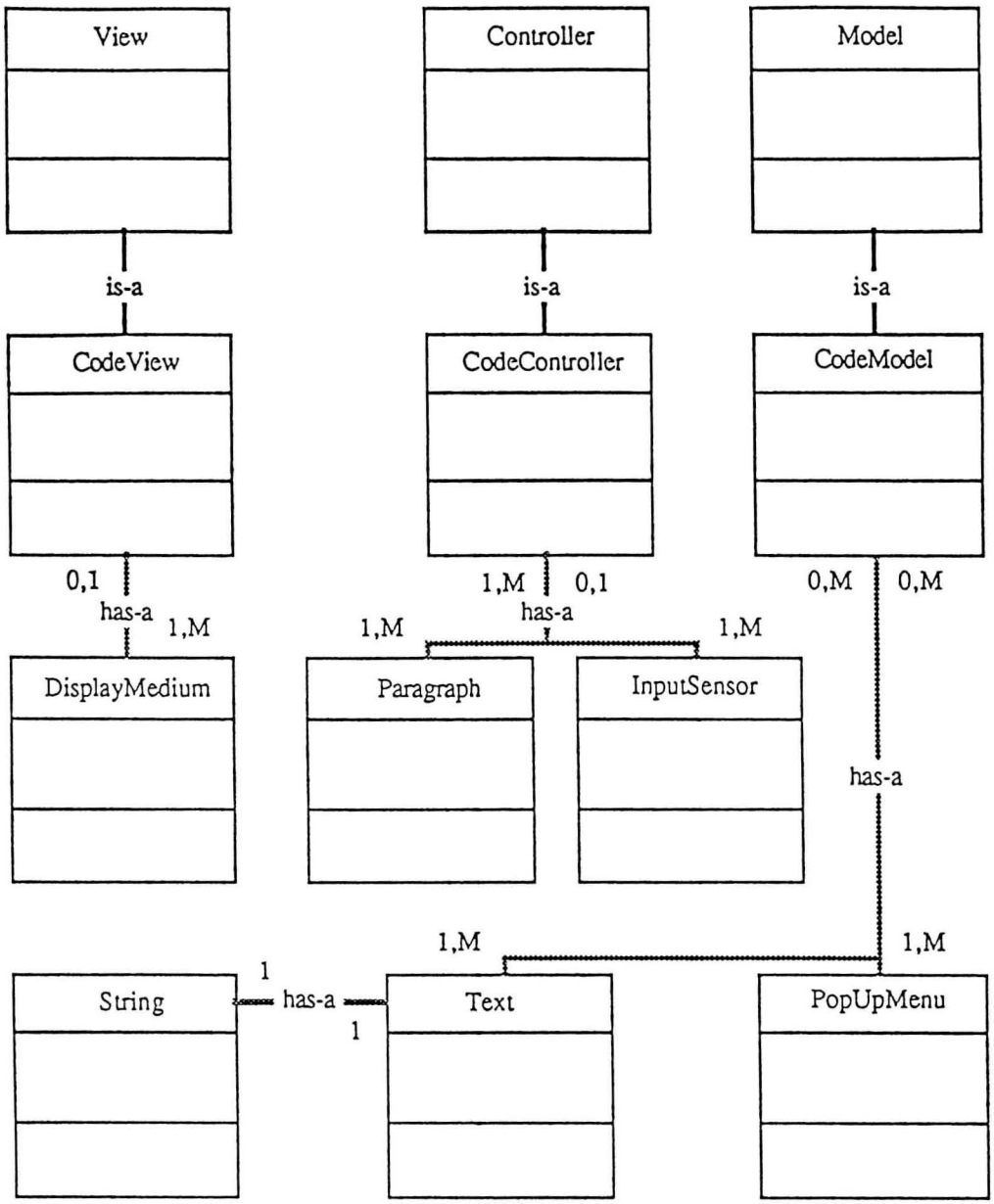




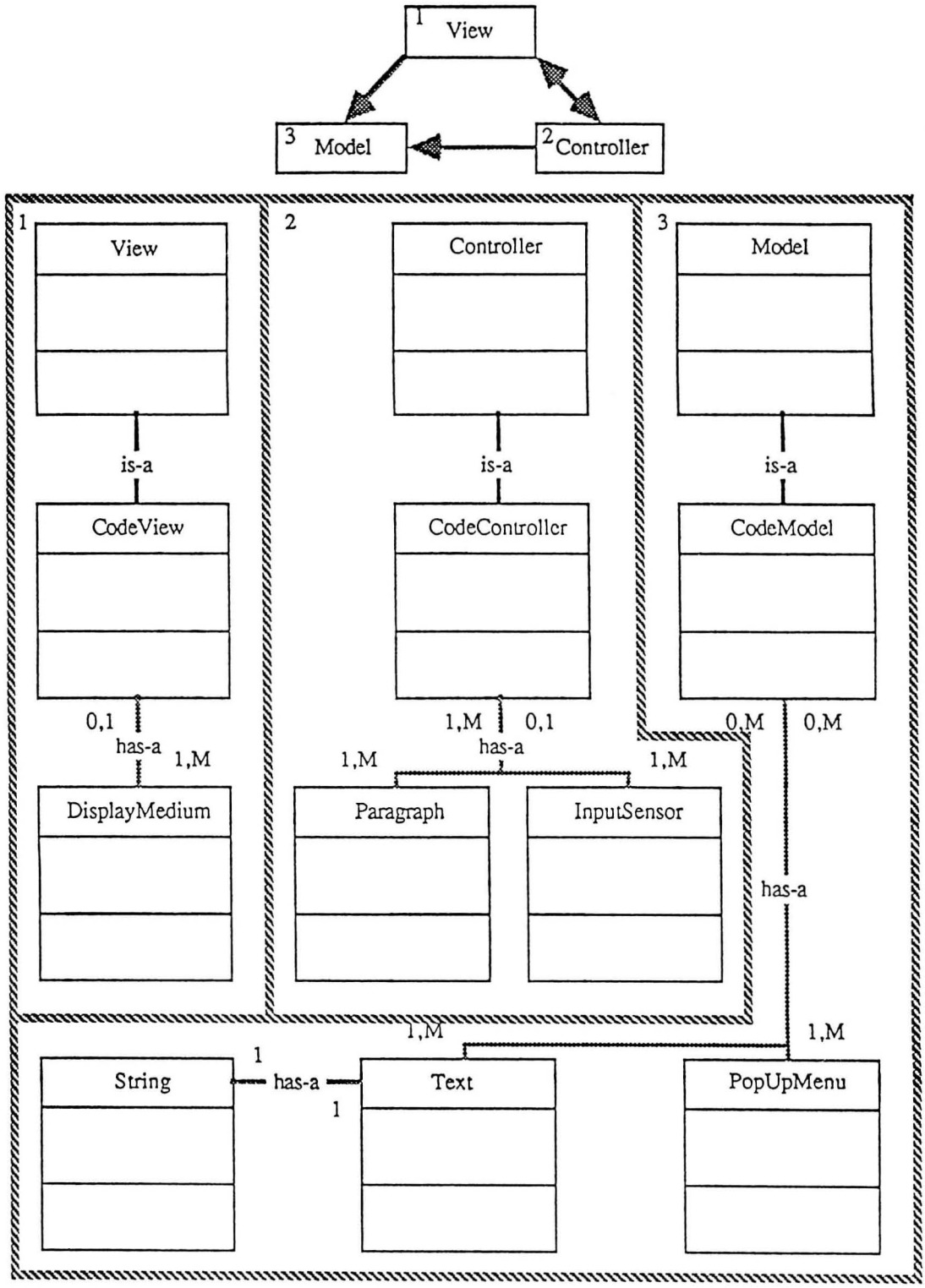




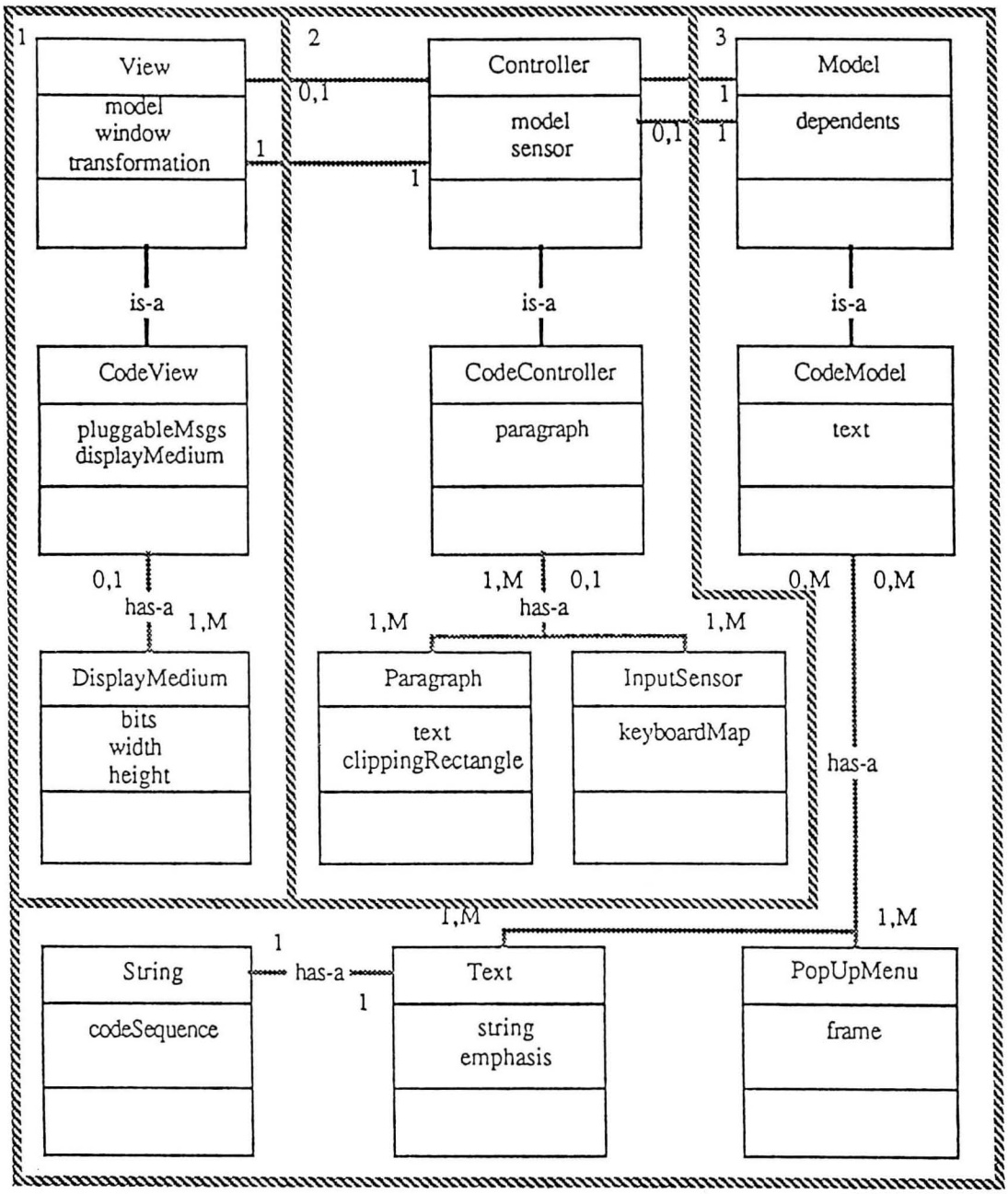
3.2. 構造の確定


View
Controller
Model
is-a
is-a
is-a
CodeView
CodeController
CodeModel
0,1
has-a
1,M
1,M
0,1
has-a
1,M
0,M
0,M
DisplayMedium
Paragraph
InputSensor
has-a
1,M
1,M
1
has-a
1
String
Text
PopUpMenu

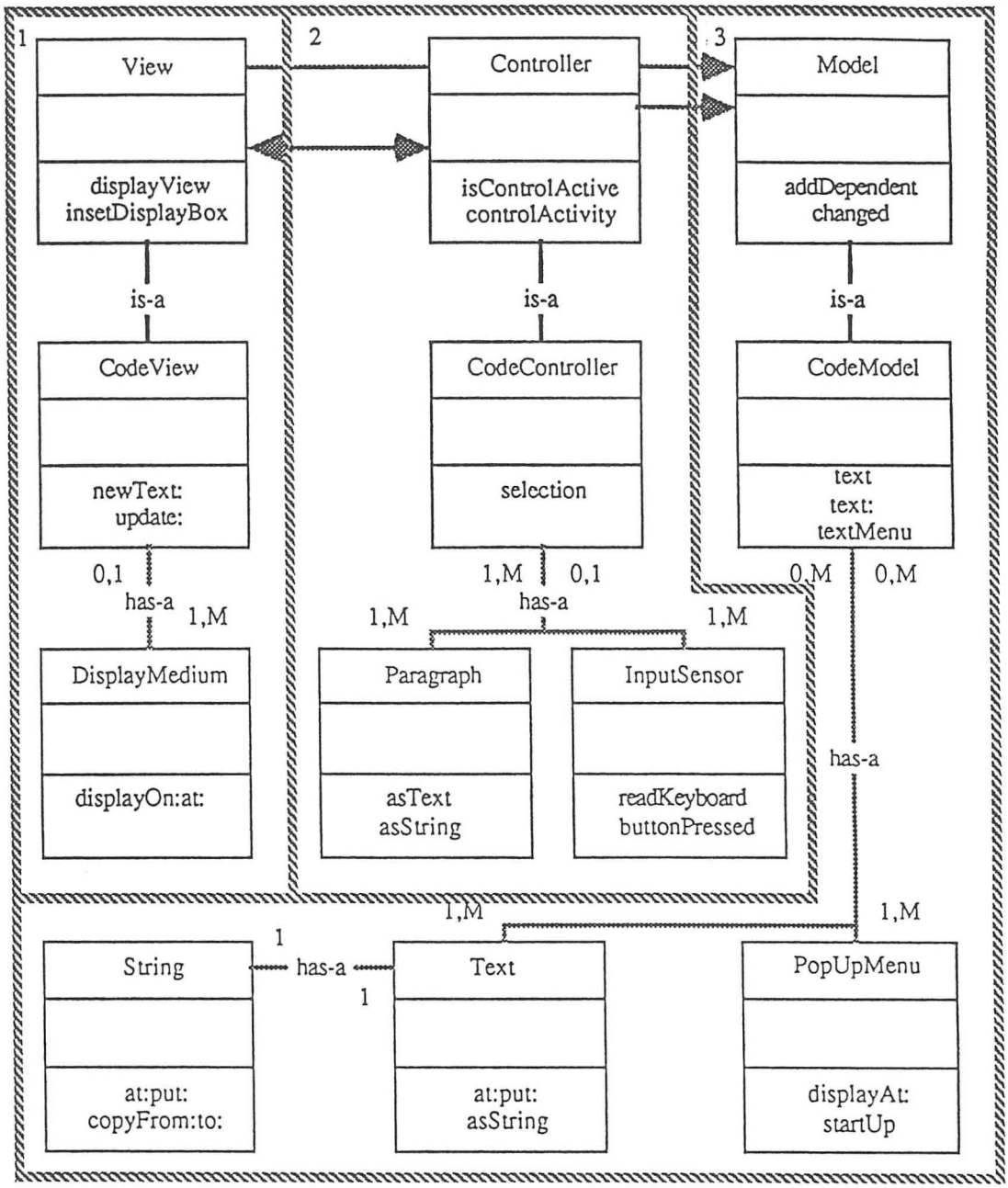
3.3. サブジェクトの定義



3.4. 属性の定義


1
2
3
View
model
window
transformation
Controller
model
sensor
Model
dependents
0,1
1
is-a
CodeView
pluggableMsgs
displayMedium
CodeController
paragraph
CodeModel
text
0,1
has-a
1,M
DisplayMedium
bits
width
height
Paragraph
text
clippingRectangle
InputSensor
keyboardMap
String
codeSequence
Text
string
emphasis
PopUpMenu
frame

3.5. サービスの定義


1
2
3
View
displayView
insetDisplayBox
Controller
isControlActive
controlActivity
Model
addDependent
changed
is-a
is-a
is-a
CodeView
newText:
update:
CodeController
selection
CodeModel
text
text:
textMenu
0,1
has-a
1,M
1,M
0,1
has-a
1,M
0,M
0,M
DisplayMedium
displayOn:at:
Paragraph
asText
asString
InputSensor
readKeyboard
buttonPressed
has-a
1,M
1,M
1
String
at:put:
copyFrom:to:
has-a
1
Text
at:put:
asString
PopUpMenu
displayAt:
startUp

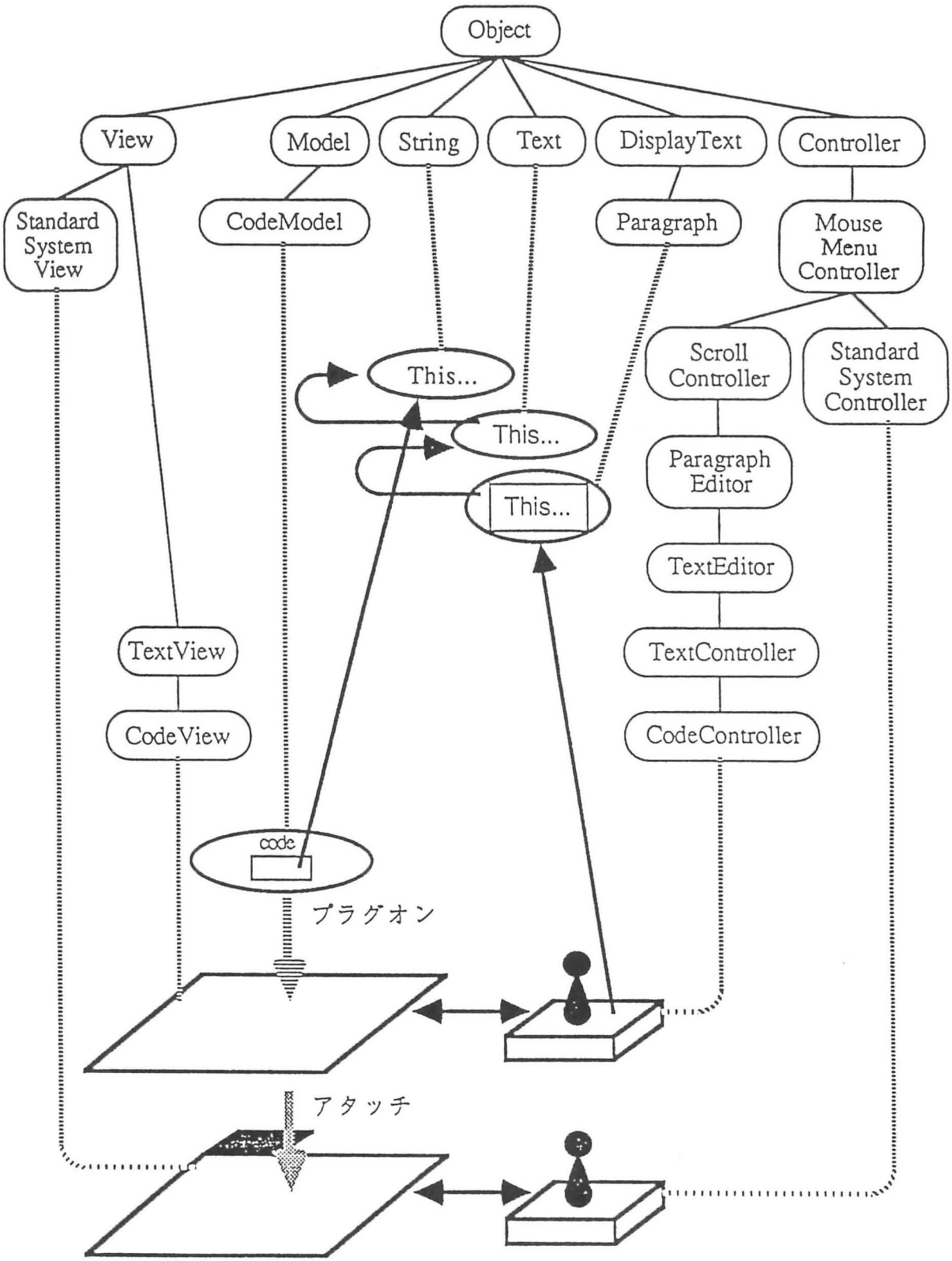
3.6. ハードウェア･アーキテクチャの決定

MacintoshIIci
メモリ: 8Mbytc
ディスク: 80MBytc

3.7. ソフトウェア･アーキテクチャの決定

Virtual: 論理メモリ空間を16MByteに拡張するソフト
Smalltalk-80 version 2.5: オブジェクト指向言語および環境
PluggableMVC: アプリケーション構築フレームワーク

3.8. 特殊化、組立て、抽象化、要素分解



3.9. サービスの詳細化

```
code: aString
    aString is kind of String.
    returnValue is self.

acceptText: aText
    aText is kind of Text.
    returnValue is member of True.

menu
    returnValue is kind of ActionMenu

text
    returnValue is member of True.

edit
    "プラガブルMVCを生成してTopViewにアタッチしてウインドウを開く"

flushMenus
    "キャッシュしているメニューをnil化する"

code: aString
    aString is kind of aString.
    returnValue is self.
```

3.10. ソース・コード・テンプレートの生成

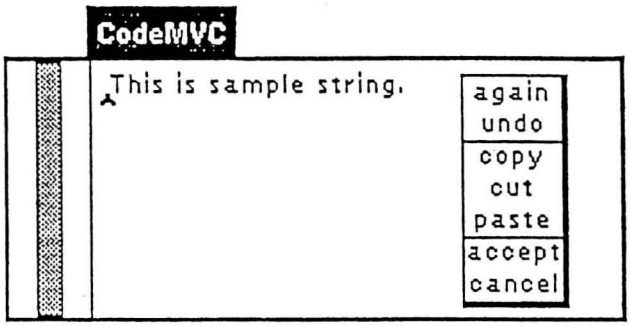
```
Model subclass: #CodeModel
    instanceVariableNames: 'code '
    classVariableNames: ' '
    poolDictionaries: ''
    category: 'Pluggable-MVC'!


code: aString
    Specification requireThat: aString isKindOf: String
    "message expressions"
    Specification requireThat: returnValue isKindOf: CodeModel! !

acceptText: aText
    Specification requireThat: aText isKindOf: Text
    "message expressions"
    Specification requireThat: returnValue isMemberOf: True! !

menu
    "message expressions"
    Specification requireThat: returnValue isKindOf: ActionMenu! !

text
    "message expressions"
    Specification requireThat: returnValue isKindOf: Text! !

edit
    "プラガブルMVCを生成してTopViewにアタッチしてウインドウを開く"
    "message expressions"! !

flushMenus
    "キャッシュしているメニューをnil化する"
    "message expressions"! !

code: aString
    Specification requireThat: aString isKindOf: String
    "message expressions"
    Specification requireThat: returnValue isKindOf: CodeModel! !
```

3.11. 実際のサービスのインプリメント

```
Model subclass: #CodeModel
    instanceVariableNames: 'code '
    classVariableNames: 'CodeModelMenu '
    poolDictionaries: ''
    category: 'Pluggable-MVC'!

CodeModel comment:
'Copyright (c) 1989 Fuji Xerox Information Systems Co., Ltd. All Rights Reserved.
Written by Jun Aoki and Sanae Fujita, 2 May 1989.
This is sample program of pluggable MVC (CodeView ¦ CodeController)
Instance Variables:
    code        <String>
'!

!CodeModel methodsFor: 'accessing'!

code: aString
    code := aString! !

!CodeModel methodsFor: 'adaptor'!

acceptText: aText
    code := aText asString.
    ^true!

menu
    "CodeModel flushMenu"

    CodeModelMenu isNil ifTrue: [CodeModelMenu := ActionMenu
                    labels: 'again¥undo¥copy¥cut¥paste¥accept¥cancel' withCRs
                    lines: #(2 5 )
                    selectors: #(again undo copySelection cut paste accept cancel )].
    ^CodeModelMenu!

text
    code isNil ifTrue: [^Text new].
    ^code copy asText! !

!CodeModel methodsFor: 'editing'!

edit
    ¦ topView codeView ¦
```

```
	topView := StandardSystemView
			model: self
			label: 'CodeMVC'
			minimumSize: 200 @ 100.
	codeView := CodeView
			on: self
			aspect: #text
			change: #acceptText:
			menu: #menu
			initialSelection: nil.
	topView
		addSubView: codeView
		in: (0 @ 0 corner: 1 @ 1)
		borderWidth: 1.
	topView controller open! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --"!

CodeModel class
	instanceVariableNames: ''!

!CodeModel class methodsFor: 'class initialization'!

flushMenus
	"CodeModel flushMenus"

	CodeModelMenu := nil! !

!CodeModel class methodsFor: 'instance creation'!

code: aString
	^super new code: aString! !

!CodeModel class methodsFor: 'examples'!

example
	"CodeModel example"

	(CodeModel code: 'This is sample string.') edit! !
```

3.12. 性能測定



<参考文献>

Coad, Peter and Yourdon, Edward, Object-Oriented Analysis. Prentice Hall, 1990.

青木淳, "オブジェクト指向プログラミング環境におけるインテグレーション過程," Proceedings of 10th Software Symposium, June 1990. または Proceedings of WOOC, 箱根 1990.

Glenn E. Krasner, Stephen T. Pope, "A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Parc Place Systems, 1988.

青木淳, "使わないと損をするModel-View-Controller," 富士ゼロックス情報システム 1988.

上谷晃弘 編著, "統合化プログラミング環境-Smalltalk-80とInterlisp-D-", 丸善, 1987.

3. 課　題

図書館システム 56
酒屋倉庫問題 57
ガス給湯システム 58
クルーズコントロールシステム 60

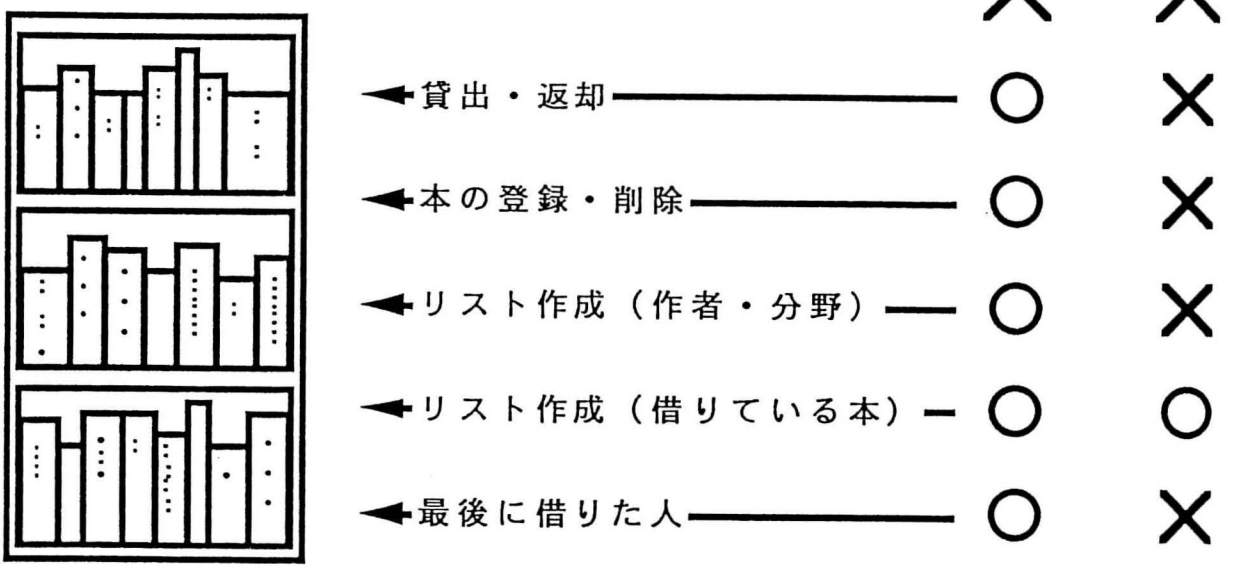
==========
1．図書館システム
==========

以下のトランザクションを処理する図書館データベース・システムを設計せよ.

1 本の貸出し，および返却
2 図書館への本の登録，および削除
3 特定の作者の本，あるいは特定の分野の本に関するリストの作成
4 特定の利用者が借りている本のリストの作成
5 特定の本を最後に借りた人の検索

このシステムのユーザーは，図書館の職員と図書館の利用者である．1，2，4，および5のトランザクションを発生できるのは職員に限られている．ただし4については，自分自身が現在借りている本の検索に限り，図書館の利用者でも行なえるようになっている．データベースは，次の条件を満たさなければならない.

- 図書館の全ての本は，貸出可能か，すでに貸し出されている.
- ある本が貸出可能であり，同時に貸し出されている，ということはない.
- 図書館の利用者は，1回にある決まった数以上の本を借り出すことはできない.



==========
2．酒屋倉庫問題
==========

ある酒類販売会社の倉庫では，毎日数個のコンテナが搬入されてくる．その内容はビン詰めの酒で，１つのコンテナには１０銘柄まで混載できる．扱い銘柄は約２００種ある．倉庫係は，コンテナを受け取りそのまま倉庫に保管し積荷票を受付係に手渡す．また受付係からの出庫指示によって内蔵品を出庫することになっている．内蔵品は別のコンテナに詰め換えられたり，別の場所に保管することはない．
空になったコンテナはすぐ搬出される．

積荷票： コンテナ番号（５桁）
搬入年月，日時
内蔵品名，数量（の繰り返し）

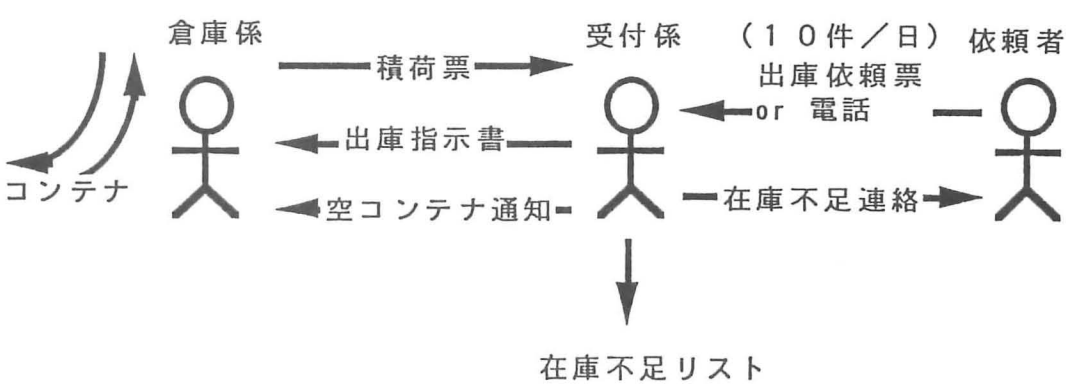
さて受付係は毎日１０件の出庫依頼を受け，その都度倉庫係へ出庫指示書を出すことになっている．出庫依頼は出庫依頼票または電話によるものとし，１件の依頼では１銘柄のみに限られている．在庫がないかあるいは数量が不足の場合には，その旨依頼者に電話連絡し同時に在庫不足リストに記入する．そして該当品の積み荷が必要量あった時点で，不足品の出庫指示をする．また空になるコンテナを倉庫係に知らせることになっている．

出庫依頼： 品名，数量
送り先名

受付係の仕事（在庫なし連絡，出庫指示書作成および在庫不足リスト作成）のための計算機プログラムを作成せよ．

出庫指示書： 注文番号
送り先名
コンテナ番号
品名，数量
空コンテナ搬出マーク
} （の繰り返し）

在庫不足リスト： 送り先名
品名，数量



==========
3．ガス給湯システム
==========

以下に記述したようなガス給湯システムを制御するマイクロプロセッサ部を設計して下さい．

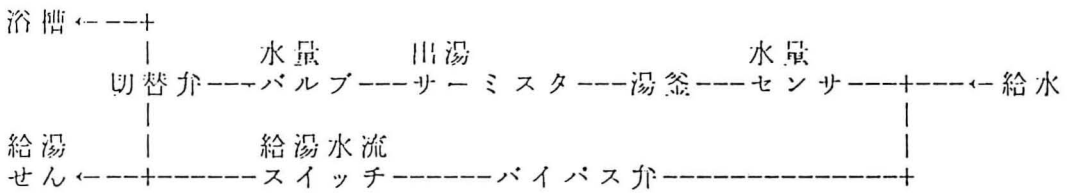
●概要

このシステムは，浴槽へのお湯はりと追いだき，および台所，洗面所等への給湯を行う．給湯せん（給湯を受ける蛇口）を開くと，自動的にガスが点火されて給湯が開始される．お湯はり，追いだきはコントローラのスイッチを押すことによって開始できる．お湯はり，あるいは追いだき中に給湯せんが開かれると，お湯はり，追いだきを一時中断して給湯を行う．給湯が終了するとお湯はり，追いだきが再開される．

コントローラにはメインコントローラと風呂コントローラがあり，どちらでも電源のＯＮ／ＯＦＦができる．メインコントローラでは給湯の温度の設定，風呂コントローラではお湯はり時の湯量，温度の設定ができる．

●システムの構成

湯釜，浴槽，給湯せん（給湯を受ける蛇口）は配管 (—) を介して図のように結合している．



切替弁は湯釜からの温湯を浴槽側あるいは給湯側に切替える．システムが停止している時，および給湯を行っている時は常に給湯側に切替わっていなければならない．水量バルブは湯釜からの温湯の流量を制御する．出湯サーミスタは温湯の温度を計測する．水量センサは湯釜へ送られた水の量を計測する．給湯水流スイッチは水流を検出した時にＯＮ，検出しない時ＯＦＦである．バイパス弁は，切替弁が浴槽側の時に開き，給湯せんに水圧をかけるためのものである．これにより，給湯せんが開けられたかどうかは，切替弁が給湯側の時は水量センサで，切替弁が浴槽側の時は給湯水量スイッチで検出できる．湯釜には，バーナと点火プラグ，比例弁，フレームロッドが備わっている．フレームロッドは，バーナに着火したかどうかを検出できる．バーナへ送られるガスの量は比例弁で制御できる．システムが停止している時は必ず閉じていなければならない．

メインコントローラ（ＭＣ），風呂コントローラ（ＢＣ）のどちらかの運転スイッチを押すと，ＭＣ，ＢＣの運転ランプが点灯する．この状態で給湯，お湯はり，追いだきを行うことができる．運転スイッチをもう一度押すとシステムが停止する．システムが停止した時，すべてのバルブ，弁は自動的に初期状態に戻る．

●給湯の手順

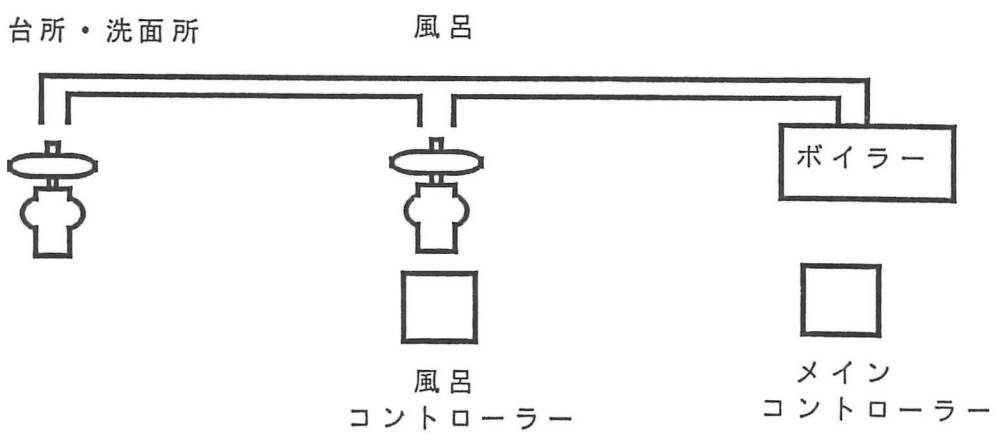
給湯せんが開かれると，水量センサがＯＮ（０以上の値）となる．数秒後，比例弁を開け，点火プラグから火花を発する．バーナに着火したことがフレームロッドによって確認されると，ＭＣ，ＢＣの燃焼ランプを点灯する．着火しなければシステムを停止する．

給湯時の温度は，ＭＣの給湯温度つまみによって設定される．出湯サーミスタ，水量センサからの情報によって，比例弁，水量バルブをコントロールし，温湯を設定された温度に保つ．
給湯せんが閉じられると，比例弁を閉じ，フレームロッドによって消火を確認してから燃焼ランプを消す．
給湯中にＢＣのお湯はり，あるいは追いだきスイッチが押されても何もせず，給湯を続ける．

●お湯はりの手順
ＢＣのお湯はりスイッチが押されたら，切替弁を給湯側から浴槽側へ切替え，バイパス弁を開く．次にバーナに点火し，ＭＣ，ＢＣの燃焼ランプを点灯する．着火しなければシステムを停止する．
浴槽へ送る湯の温度は，ＢＣのお湯はり温度つまみで設定した温度に保つ．
お湯はりスイッチが押されるか，送った湯の総量がＢＣのお湯はり水量つまみで設定した量を越えたら，比例弁を閉じ，湯が給湯用の温度以下になるまで待ってから切替弁を給湯側に切替え，バイパス弁を閉じ，燃焼ランプを消す．
お湯はり中に追いだきスイッチが押された場合，お湯はりから追いだきに切り替える．

●追いだきの手順
ＢＣの追いだきスイッチが押されたら，切替弁を給湯側から浴槽側へ切替え，バイパス弁を開く．次にバーナに点火し，ＭＣ，ＢＣの燃焼ランプを点灯する．着火しなければシステムを停止する．
出湯の温度は，９３° Ｃに保たれる．
追いだきスイッチが押されるか，送った湯の総量が１００リットルを越えたら，比例弁を閉じ，湯が給湯用の温度以下になるまで待ってから切替弁を給湯側に切替え，バイパス弁を閉じ，燃焼ランプを消す．
追いだき中にお湯はりスイッチが押された場合，追いだきからお湯はりに切り替える．

●お湯はり，追いだき中の給湯
お湯はり，あるいは追いだき中に給湯せんが開かれると，給湯水流スイッチがＯＮになる．この時，ＢＣの割り込みランプを点灯させる．同時に，比例弁と水量バルブを制御して，湯を給湯用の温度にしてから切替弁を給湯側に切替え，バイパス弁を閉じ，給湯を開始する．
給湯せんが閉じられたら切替弁を浴槽側に切替え，バイパス弁を開き，お湯はり，あるいは追いだきを再開する．同時に，割り込みランプを消す．



==========
4．クルーズコントロールシステム
==========

要求事項

運転手の２つの仕事の負担を軽減するための自動車用付加システムについて以下に定義する．このシステムは長距離運転時における巡行速度を維持し，自動車の動作状況を評価する．クルーズコントロール部では，このシステムが運転手に代わって巡行速度を維持してくれる．モニタ部では，このシステムが運転手に燃費と，整備時期について知らせてくれる．

クルーズコントロールシステムは，”クローズドループ制御”を実行する -- 与えれらた要求速度の設定値に対して，実際の速度を監視してできるだけ要求速度に近づけるように調整する．このプロセスは運転手が行うものと近いものである．— 運転手はスピードメータを見てアクセルペダル(スロットル)を踏んだり離したりして実際の速度を要求する速度に近づようとする．本クルーズコントロールシステムは，このようなクローズドループ制御に必要とされることを行う．

クルーズコントロールシステムは自動車の速度を監視しなければならない．この情報は運転手から与えられる他の情報とともにシステムに与えられ，燃費や平均速度のような自動車の運転状況の情報となって運転手に示される．更に，自動車の走行距離を記録し，オイルフィルタの交換，エアフィルタの交換，主要点検時期のような，故障を未然に防ぐための適切な整備時期を運転手に知らせてくれる．

本システムは常時その表示や監視の機能を実行することができる．クルーズコントロールの機能はエンジンがかけられた後であればいつでも実行することができる．

クルーズコントロールが実行されたときはいつでも，自動車はその時点で出していた速度(ゴール速度)に維持される．本システムは自動車の速度を車輪の回転率を知ることによって監視し，スロットルの位置を調整することによって要求された速度を維持する．加えて運転手はボタンを押すことによって速度を増すことができる．ボタンを離すことによってこの速度の増加を止めることができる．本システムはその増加後の速度(これが新たなゴール速度)を維持する．

運転手はいつでもクルーズコントロールを停止することができる．クルーズコントロールが動作中にブレーキが踏まれたことを検知すると，その動作を停止する．次に運転手がシステムに動作の再開を命令するとゴール速度に達するように再加速を行う．もし運転手が加速を行った場合は(例えば追い越すため)，アクセルペダルを離した後でゴール速度を元の値に自動的に戻す．

本システムは校正し直すことが可能である．これは自動車がいろいろなサイズのタイヤを使用することができるようにするためである．運転手はあらかじめ測定されている１マイルの距離の開始点で校正を始動し，１マイルの終了点で校正を終了させる．これによって本システムは，１マイルの距離で何回車輪が回転するかに基づいて決められる変換率を設定し直す．しかしながらこの校正中は，クルーズコントロールを止めておかなければならない．

本システムは走行した距離を常に記録している．この情報と運転手から得た情報を基にして燃量の消費率の情報を運転手に示す．本システムは表示装置とキーボードを持っており，これによって運転手との間で情報をやりとりする．燃量を供給したときはいつでも運転手は給油し

た燃量の量を入力する．システムは前回の燃量に対する消費率を計算する．次にシステムは走行距離の記録を初期化し直す．運転手はいつでも平均燃量消費率を知ることができる．

本システムは下の表１に示される整備時期の間隔を監視して，運転手に適切な整備時期を知らせる．表示装置上に適切な指示を出す．注意を促すためのしきい値(整備時期の手前２５０マイル)を越えると断続的にそれを知らせる表示を出す．残り５０マイル以内になると連続的に表示する．運転手はその整備が完了した時点でシステムにそれを知らせることによって表示を消すことができる．

表１　整備時期

オイルとオイルフィルタの交換	３０００マイル毎
エアフィルタの交換	１００００マイル毎
主要部整備	１５０００マイル毎

——
想定
——

次のことを想定することとする．

1. 自動車の速度と加速は共にスロットルの一次関数とする;

$$D = (k1 * S1) + (k2 * S)$$

ここで

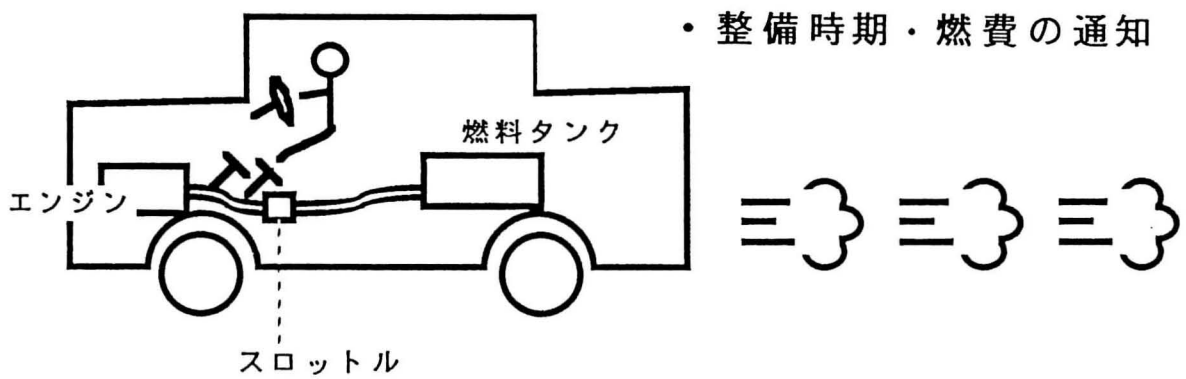
D = スロットルのあき具合
S = 自動車の速度(要求される，または実際の)
$S1$= 要求される加速量

2. 速度の差は線型のスロットル操作を行うことで補正することができることとする;

$$D = (k2 * S) - (k3 * E)$$

ここで

E = 速度の差



4. 各社の CASE ツールの概要

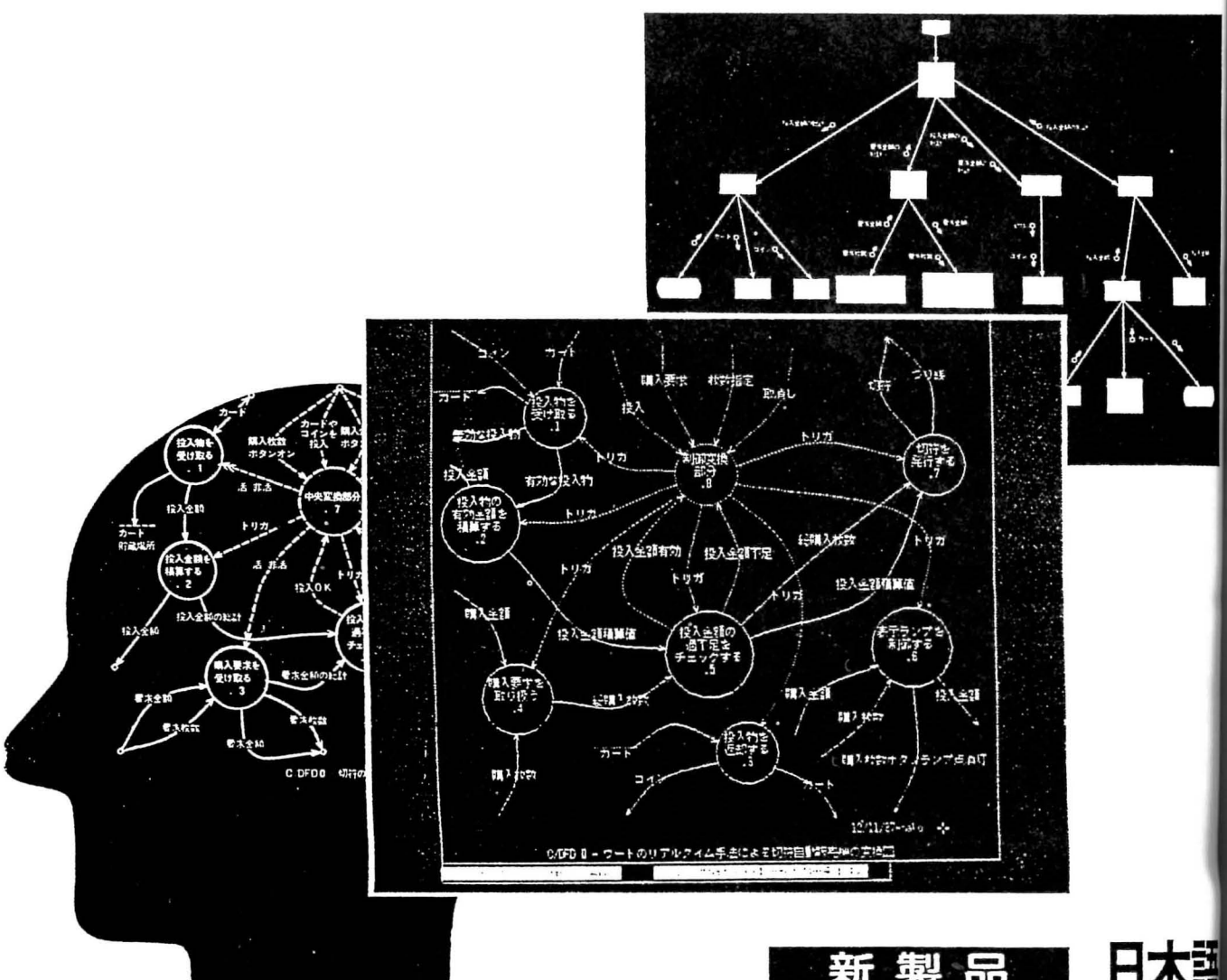
CASE Bench	ソニー・テクトロニクス	64
Grapher Gear	富士ゼロックス情報システム	71
Prototyper's Work Bench	PFU	75
Software through Pictures	SRA	80
Teamwork	東陽テクニカ	84

SONY Tektronix
ソニー・テクトロニクス

新しいソフトウェア開発ハトレーとワード手法をサポート

リアルタイムSAとSDツール日本語化

●リアルタイム手法によりソフトウェアのタイミング情報を視覚化●要求仕様と設計の本格作成で開発時間短縮、費用削減●グラフィック表示で評価がし易い●SAのデータフロー図からSDの構造図に自動変換分析●公開日本語テキスト・ファイル・フォーマットでお手持ちのホストやツール使用可●自動検証・修正・生成●日本語化で実用化一挙推進



新製品　日本語
リアルタイムSA/SDツール
CASE Bench

CASE Bench リアルタイムSAツール

●完全な日本語化を実現
●ハトレー、ワード手法のリアルタイムSA
●オンラインによる要求仕様の作成でソフトウェア開発期間を短縮
●グラフィック表現により要求仕様を記述
●仕様の抜けや矛盾のない信頼性の高い要求仕様を作成

CASE Benchのリアルタイム SAツールは、リアルタイムSA(ストラクチャード・アナリシス)手法を使ってソフトウェアの要求仕様を作成するユーザを支援します。リアルタイムSA手法とは、従来のSA(構造化分析)手法*と異なり、リアルタイム・システム特有の制御やタイミングの記述をうまく行えるように工夫を施した手法のことで、デレク・ハトレー氏らのハトレーの手法と、ポール・ワード氏らのワードの手法が良く知られています**。当社のリアルタイムSAツールは、この二つの手法のいずれかを使ってソフトウェアの要求仕様を作成するユーザに計算機による支援を行います。また、完全な日本語化を実現しています。その結果、オンラインによる仕様の作成や修正が行え、開発期間を大幅に短縮できます。さらにリアルタイムSA手法に基づいて、仕様の抜けや矛盾などがないか自動的に検証できますので、設計仕様の作成を開始する前に、質の高いソフトウェアの要求仕様を効率よく作成できます。さらに、当社SDツールを使うことにより、リアルタイムSAツールで作成された要求仕様を、そのまま設計仕様作成のための入力データとして利用できます。

また、ダイアグラムを含むすべての仕様書が、ASCII のテキスト・ファイルで作成され、そのファイル・フォーマットが公開されているため、仕様書はお手持のホスト・コンピュータやツールで利用することも可能です。

CASE Bench リアルタイムSAツールの主な機能

仕様書の編集機能

リアルタイムSAツールのグラフィックス・エディタを使うことで、ダイアグラムを効率良く簡単に編集できます。

*ダイアグラム上のすべての要素を、オンラインで素早く生成、移動、削除。
*親と子の関係にあるダイアグラム間や、ダイアグラムと辞書の間を、容易に移動しながら編集。
*グラフィックス拡張機能を使うことで、ダイアグラムの大きさやビューを自由に変えながら編集(図1)。
*別のダイアグラムを、同時に複数枚表示しながら編集。(マルチ・ビューポート機能)
*ダイアグラム上の要素の表示形式を、自由に変更(図2)。
*エディタのコマンド・ヘルプを、何時でもオンラインで表示。

また、リアルタイムSAツールで用いられるミニ仕様書や辞書は、システム上で稼動するテキスト・エディタで、十分に編集できます。

仕様書の検証機能

リアルタイムSAツールで作成した仕様書を、以下の項目について検証します。

*ダイアグラムの要素の記述形式。
*親子関係にある仕様書の整合性。
*辞書の記述形式。
*すべてのダイアグラムと辞書との整合性。

仕様書の自動修正、自動生成機能

リアルタイムSAツールで用いられる仕様書を自動修正や自動生成します。

仕様書の表示機能

リアルタイムSAツールで作成したダイアグラム等の仕様書は、当社のすべてのグラフィックス・ターミナルに表示できるほか、当社やDECのハード・コピーすべてに出力できます。また、PostScript対応のプリンタにも出力できます。

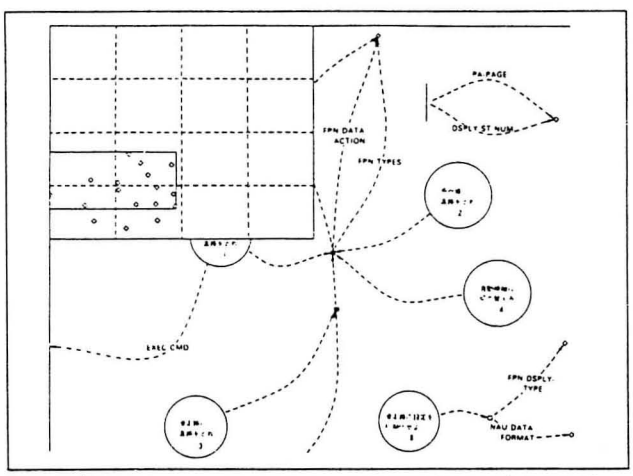


図1 グラフィックス拡張機能を使ってダイアグラムの大きさやビューを変えた編集 (注1)

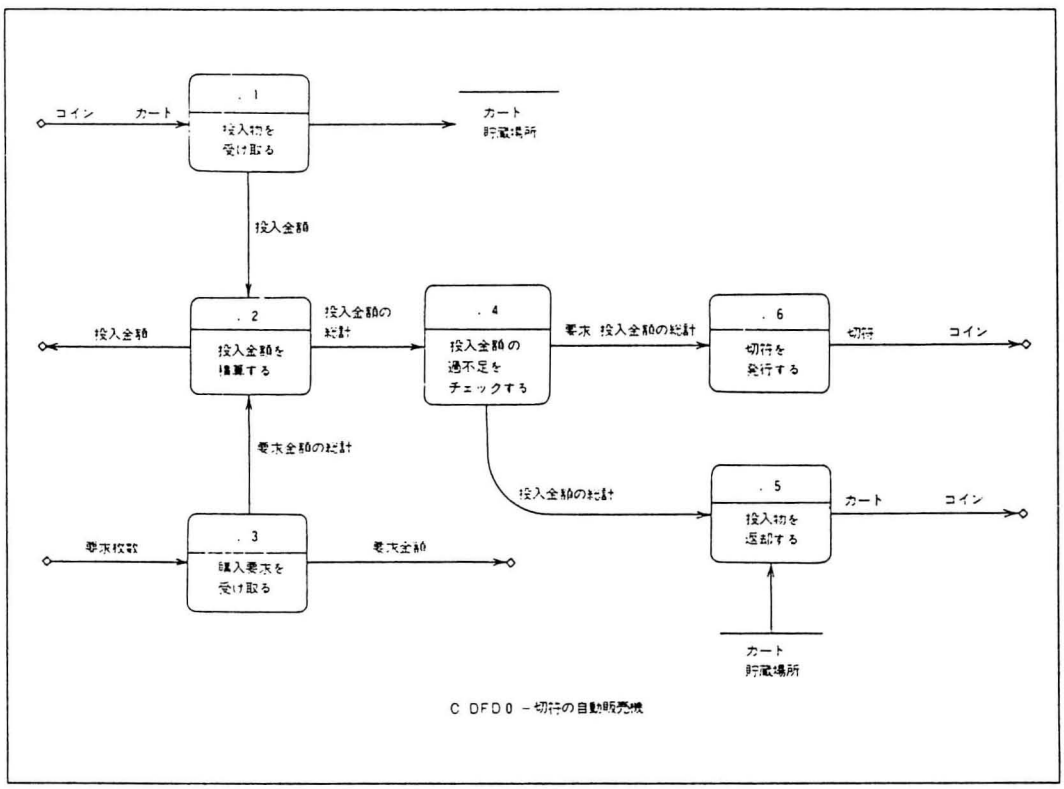


図2　ダイアグラム上の要素の表示形式を自由に変更

＊ Tom DeMarco, Structured Analysis and System Specification, Prentice-Hall/Yourdon Press. 1978.
構造化分析とシステム仕様、高梨・黒田訳、日経BP社、1986.

＊ Chris Gane and Trish Sarson, Structured Systems Analysis: tools and techniques, New York: Improved System Technologies Inc., 1977.

＊＊Derek J. Hatley, The Use of Structured Method in the Development of Large Software-based Avionics Systems, AIAA/IEEE 6th Digital Avionics Systems Conference, Dec. 1984.
ハトレー氏のリアルタイムSA手法――大規模なソフトウェア主体の航空システムを構造化手法で開発、bit誌 Vol. 19, No14, 1987年12月号.

＊＊Derek J. Hatley and Imtiaz A. Pirbhai, Strategies for Real-Time System Specification, Dorset House Publishing, Nov. 1987.
リアルタイム・システムの構造化分析、ソニー・テクトロニクス(株)立田訳、日経BP社、1989。

＊＊Paul T. Ward, The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986.
ワード氏のリアルタイムSA手法、1988.

＊＊Paul T. Ward and Stephen J. Mellor, Structured Development for Real-Time Systems, Prentice-Hall/Yourdon Press, 1985.

注1）図1はDerek J. Hatley, The Use of Structured Method in the Development of Large Software-based Avionics Systems, AIAA／IEEE 6th Digital Avionics Systems Conference, Dec. 1984. の図9を引用しています。
(Copyright ©1986. American Institute of Aeronautics and Astronautics Inc. Translated, with permission, from AIAA Paper No. 84-2595.)

この製品を輸出する場合は外国為替及び外国貿易管理法により政府の許可が必要です。また輸出先によっては米国政府の許可が必要です。

CASE Bench SDツール

●完全な日本語化を実現
●オンラインによる構造図の作成および修正で開発期間を短縮
●要求仕様を設計の入力情報に変換
●システムの階層構造を設計

CASE BenchのSDツールは、SD(構造化設計)手法*を使ってソフトウェアを設計するユーザに計算機支援を行ないます。また、完全な日本語化を実現しています。その結果、オンラインによる仕様の作成や修正が行なえ、開発期間を大幅に短縮できます。また、SDツールは、SD手法に基づいて、仕様の抜けや矛盾などが無いか自動的に検証することができますので、コーディングを開始する前に、質の高いソフトウェアの設計仕様を効率良く作成できます。そして、リアルタイムSAツールで作成された要求仕様が、そのまま設計仕様作成のための入力データとして利用できます。

また、構造図を含むすべての仕様書が、ASCIIのテキスト・ファイルで作成され、そのファイル・フォーマットが公開されているため、仕様書はお手持のホスト・コンピュータやツールで利用することも可能です。

CASE Bench SDツールの機能

ダイアグラムをマージする機能

ダイアグラムから構造図への変換を効果的に行なうため、リアルタイムSAツールで作成した複数のダイアグラムを一枚のダイアグラムに変換することができます(図1)。

ダイアグラムから構造図への変換機能

リアルタイムSAツールで作成したダイアグラムを大まかな構造図に変換します。この時、ダイアグラムの任意のプロセスをボスに指定することで、それを頂点とする階層構造が形成され、各プロセスはその階層構造のモジュールに変換されます(図2)。

また、構造図全体の複雑さを定量的に評価するバーバラ ジン氏等の評価尺度**に基づき、構造図に変換した時のモジュールの独立性が最も高くなるようなプロセスを見つけだし、それをボスとして自動的に設定することもできます。

構造図の編集機能

SDツールのグラフィックス・エディタやフォーマットの機能を使うことで、構造図を効率良く簡単に編集できます。

*構造図上のすべての要求を、オンラインで素早く生成、移動、削除。
*構造図のひとまとまりの要素が移動できる上、それを別ファイルに保存。
*別ファイル構造図と合成。
*ページ接続子を経由して構造図間を容易に移動。
*エディタのコマンド・ヘルプを、何時でもオンライン表示。
*フォーマットの機能により、構造図上の各要素を再配置し、その階層構造を見易くします(図3、4)。

構造図の検証機能

SDツールで作成した仕様書を検証します。

構造図の評価機能

*構造図の複雑度等を図や表で定量的に示せます(図5、6)。

構造図の表示機能

SDツールで作成したダイアグラム等の仕様書は、当社のすべてのグラフィックス・ターミナルに表示できるほか、当社やDECのハード・コピーすべてに出力できます。またPostScript対応のプリンタにも出力できます。

その他の機能

*構造図をいくつかに小さく分けて、読み易くすることができます。
*Cのソースコードを構造図に変換できます。

* Meilir Page-Jones, The Practical Guide to Structured Systems Design, Prentice-Hall/Yourdon Press, 1980.

**Barbara H. Yin et al., The Establishment and Use of Measures to Evaluate The Quality of Software Design, Proc. ACM Software Quality Workshop, San Diego, CA, Nov. 1978.

CASE Benchの動作環境

CASE Benchは、すべてのVAX、MicroVAXシリーズと当社グラフィック・ターミナルまたは、VAXstationシリーズ上で動作します。また、オペレーティング・システムはULTRIX-32、UNIX、VMSで利用できます。

*VAX、MicroVAXおよびVAXstation、ULTRIX-32はディジタル イクイップメント社の商標です。
PostScriptはAdobe社の商標です。
UNIXはAT&Tで開発されたOSです。

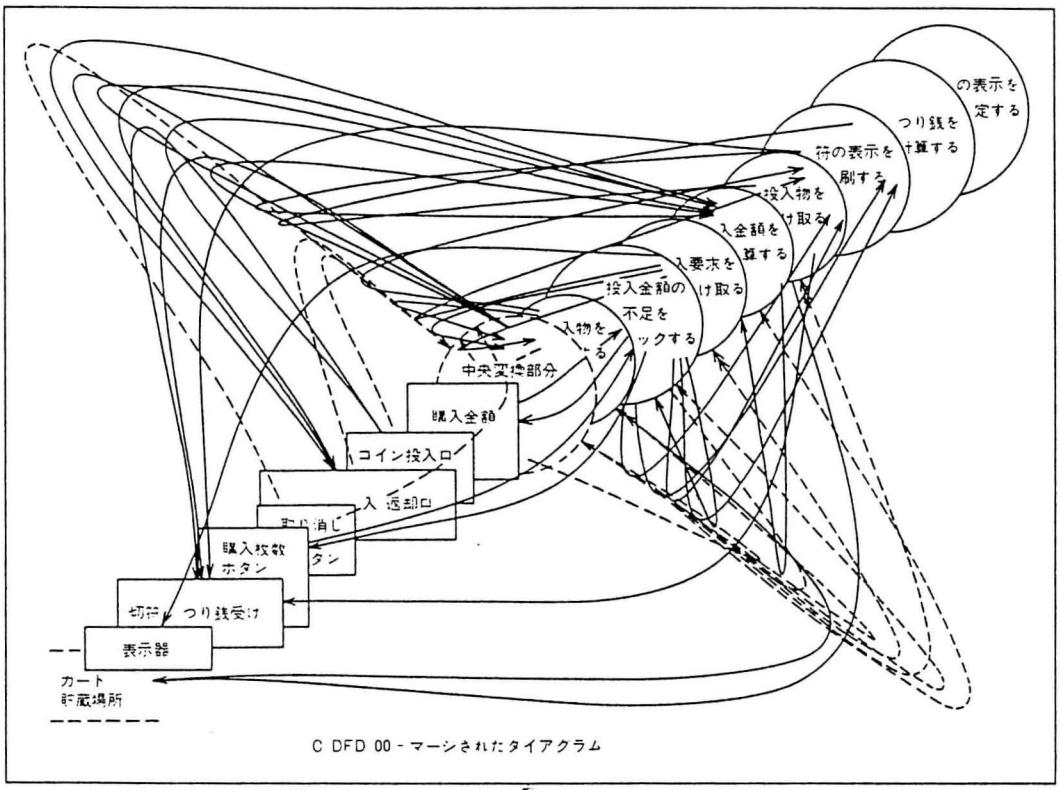
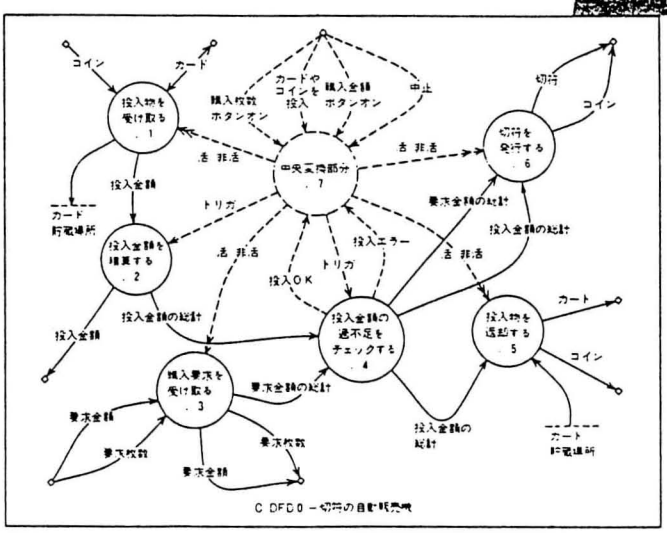
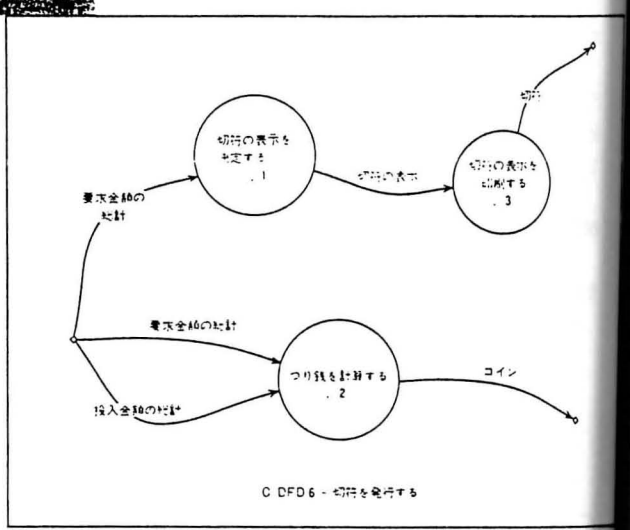


図1 複数のダイアグラムを一枚のダイアグラムに変換





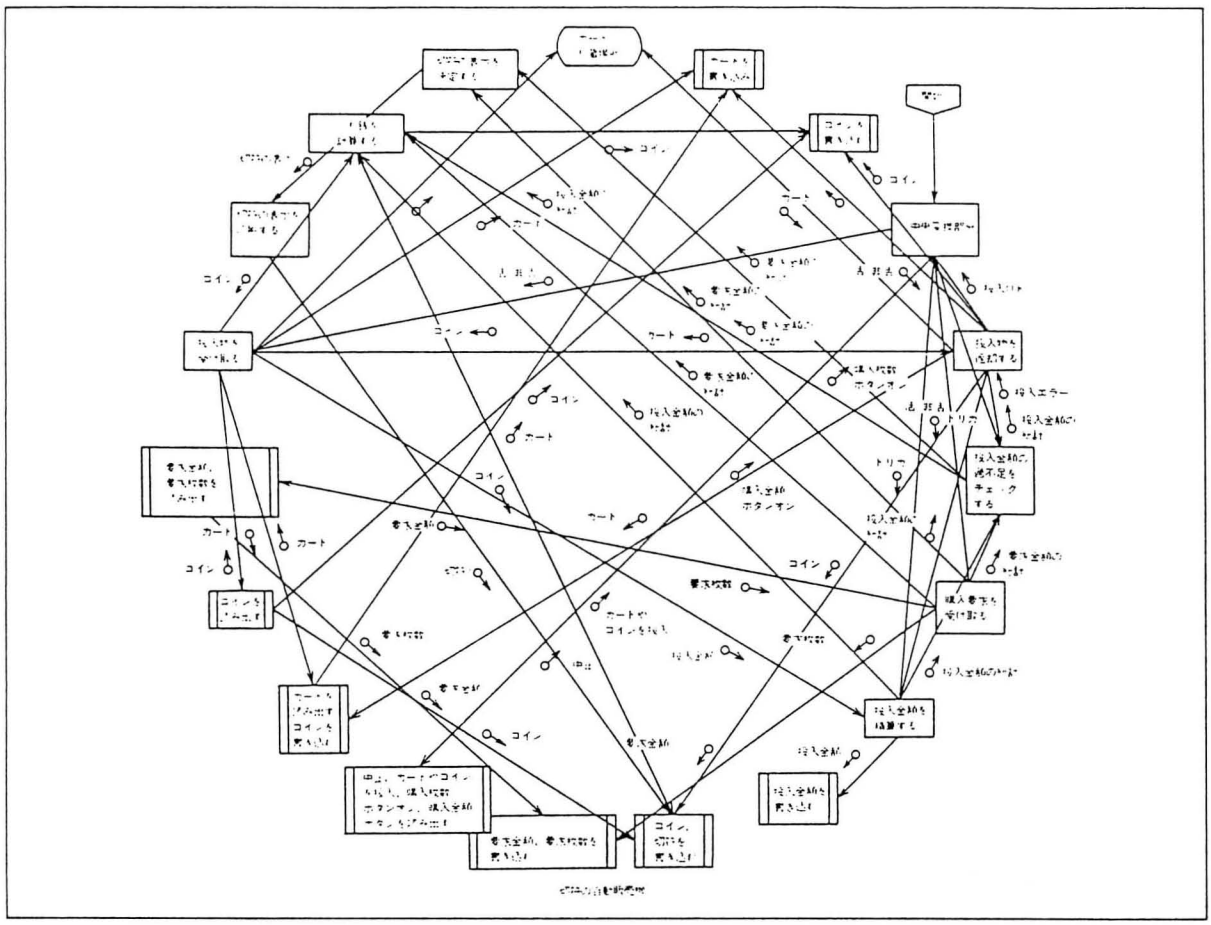


図2　ダイアグラム（図1）を大まかな構造図に自動的に変換（変換分析をサポート）

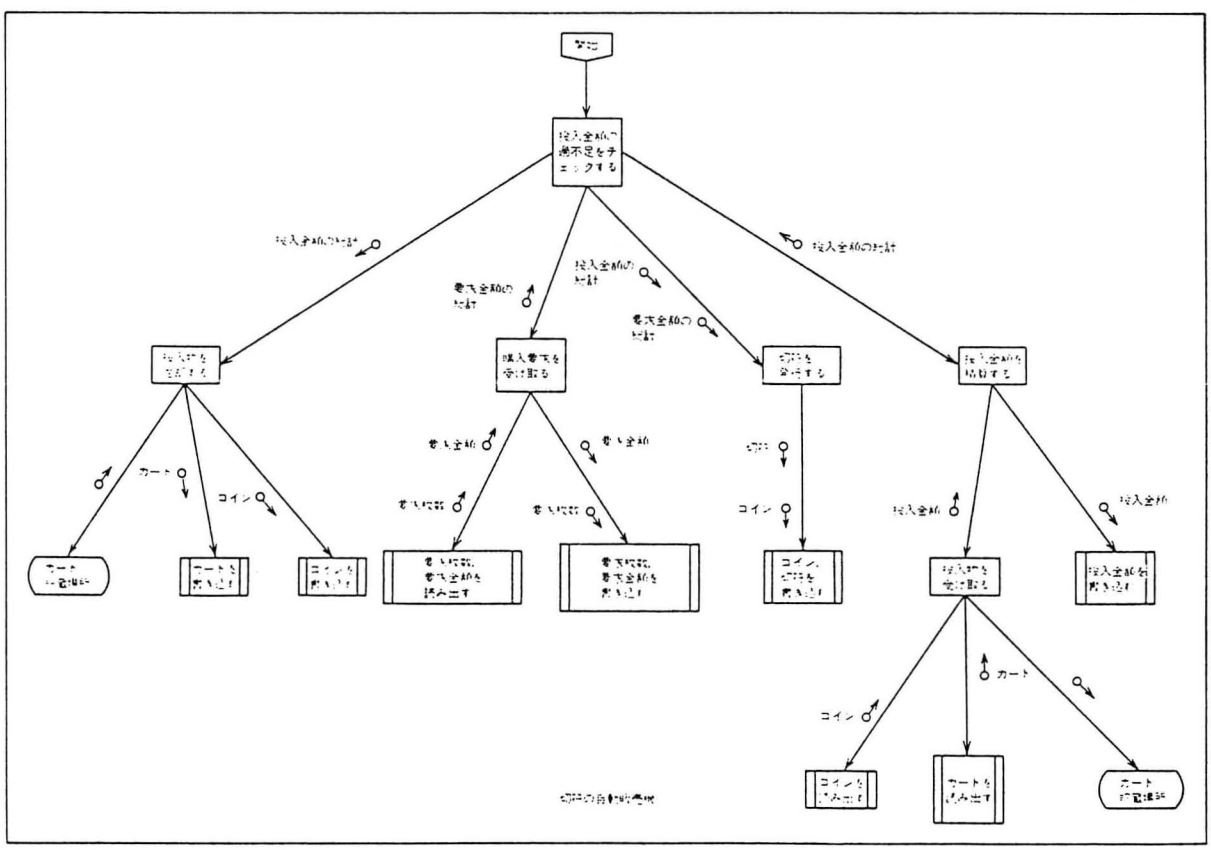


図3　大まかな構造図（図2）のモジュールが見易い階層構造になるよう自動的に再配置（フォーマット機能）

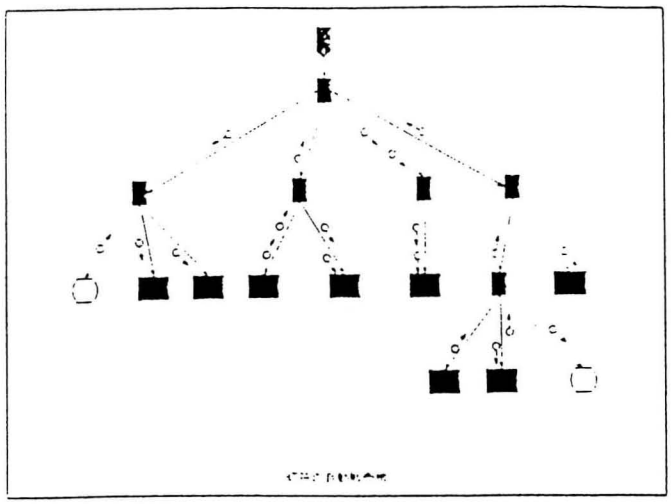


図4　階層性に視点を置いた見易い表示

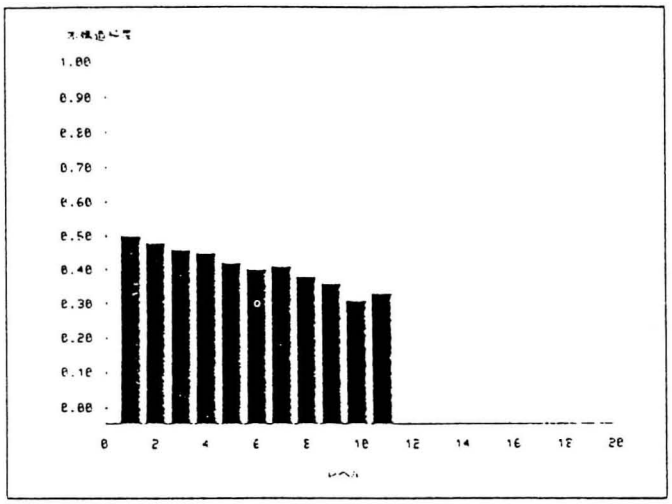


図5　グラフによる複雑度の表示

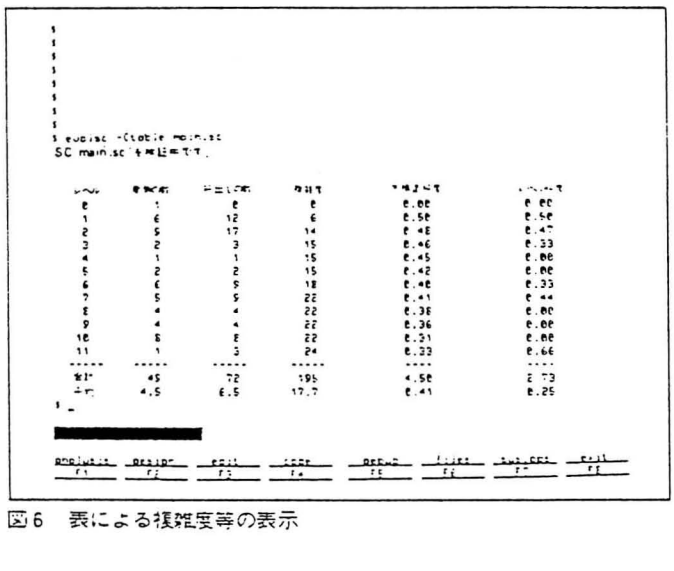


図6　表による複雑度等の表示

SONY
Tektronix
ソニー・テクトロニクス

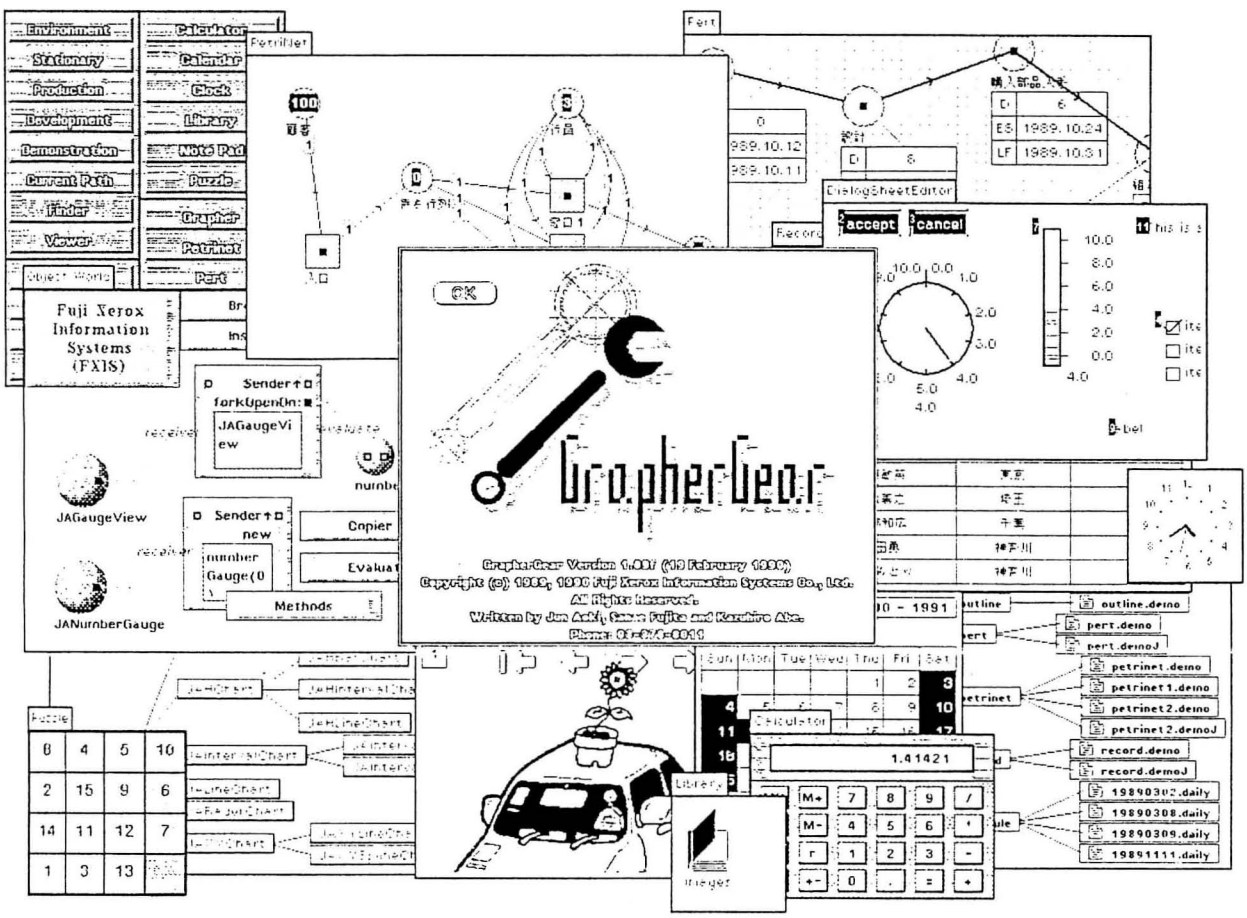
ソニー・テクトロニクス株式会社
ロジックシステム部／東京都品川区北品川5-9-31
TEL03・448-4704　〒141
仙台営業所／仙台市青葉区一番町1-2-25 仙台NSビル
TEL022・267-2181　〒980
土浦営業所／土浦市桜町1-16-12 住友生命土浦ビル
TEL0298・24-2602　〒300
大宮営業所／大宮市桜木町4-37-2 大宮沢田ビル
TEL048・646-0711　〒331
多摩営業所／国立市富士見台2-45-10
TEL0425・73-2111　〒186
厚木営業所／厚木市旭町1-8-14 グリーン会館
TEL0462・28-6768　〒243
名古屋営業所／名古屋市中村区名駅三丁目21-7 三交ビル
TEL052・581-3547　〒450
大阪営業所／大阪市中央区北浜東4-33 大阪大林ビル
TEL06・947-0321　〒540
広島営業所／広島市中区橋本町10-10 広島インテス
TEL082・222-7221　〒730
福岡営業所／福岡市博多区博多駅前3-2-1 日本生命博多駅前ビル
TEL092・472-2626　〒812

記載内容は予告なく変更する場合がありますのであらかじめご了承下さい。

GrapherGear

Smalltalk-80 アプリケーション開発支援ツール

GrapherGear はプログラミングにおけるさまざまな事象をノードとアークで表現できるグラファーを核に、リスト処理機能をはじめとする各種ツール群、Smalltalk -80の基本イメージの機能を拡張できるユーティリティー群で構成されています。また、グラファーをさまざまな分野に応用したプロトタイプ的なアプリケーション群を提供しています。



〈主なプロトタイプ・アプリケーション群〉

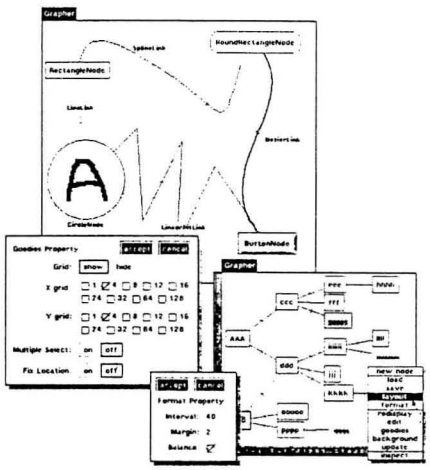
- 視覚的プログラミングツール
- 階層ファイル・システム・ファインダー
- アウトラインプロセッサー
- クラス階層ブラウザー
- ペトリネットシミュレーター
- プラガブルゲージ
- ドキュメントライブラリー
- 工程管理ツール（PERTツール）
- チャートユーティリティ
- マトリックスユーティリティ
- ダイアログユーティリティ
- パーソナル・インフォメーション・マネジャー

〈主なツール群〉

- グラファー（カーネルとライブラリー）
- アラートボックス
- メニューユーティリティ
- リスト処理機能とLispインタープリタ
- 拡張されたFormとOpaqueForm

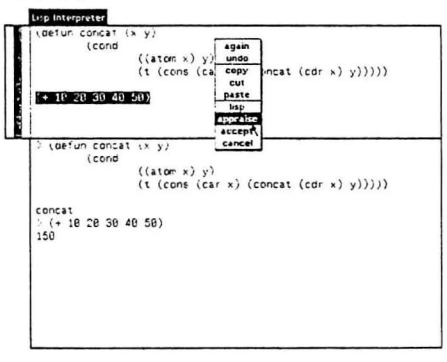
■グラファーライブラリー

サイクリック・リンクのチェックやトポロジカル・ソート、ツリー／ラティス／フォレストのレイアウト整列などの機能。使用可能なノードの種類には矩形ノード、円ノード、ボタン・ノード、角のとれた矩形ノードなどがあり、これらはすべて編集が可能です。また、全てのノードがプログラマブル・アイコンとして作用するため、ノード自身が個々に動作（アクション）を起こすこともできます。アークには直線、屈曲線、スプライン曲線、ベジエ曲線などの種類があり、これらも編集が可能になっています。



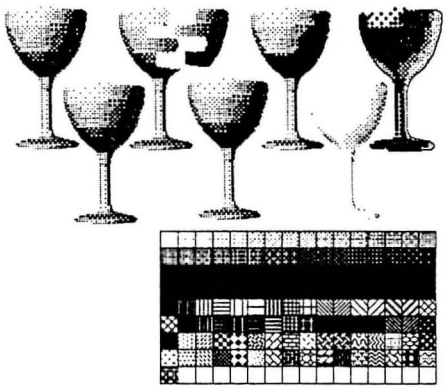
■リスト処理機能とLispインタープリター

Smalltalk-80で記述されたリスト処理機能とLispインタープリター。Lisp1.5程度の機能を有し、LispからSmalltalk-80を呼び出す機能やSmalltalk-80のオブジェクトをLispのアトムとして扱える機能などを提供します。これにより、Smalltalk-80のプログラミングにおける簡単なリスト処理機能やS式へのデータ変換などが可能となります。



■拡張されたFormとOpaqueForm

既存のFormとOpaqueFormの拡張機能。



エンハンスされたForm
▷領域塗りつぶしアルゴリズム
▷視覚効果
▷129種のマスク・パターン

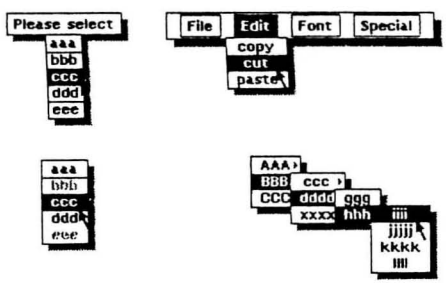
エンハンスされたOpaqueForm
▷任意領域の切り出し
▷完全なコンビネーション・ルールとハーフトーン

■メニューユーティリティ

Smalltalk-80既存のメニューをエンハンスしたメニュー群。

メニュー項目のナビゲーション機能やスクロール機能、さらに、メニューを階層的に表示する機能など多彩な機能を提供。

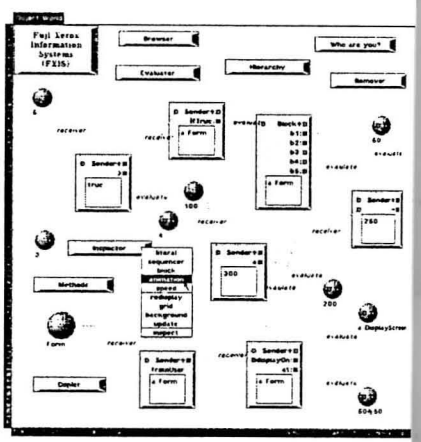
▷ポップアップ・メニュー
▷プルダウン・メニュー
▷カスケード・メニュー



■視覚的プログラミングツール（ARK-like）

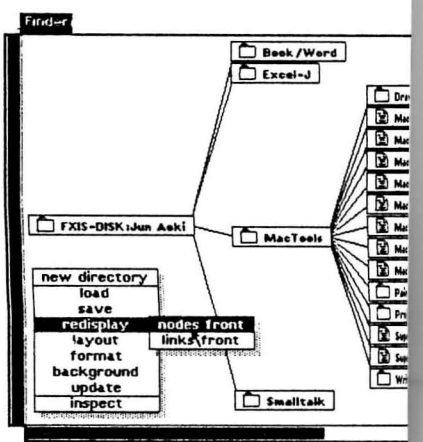
Smalltalk-80のオブジェクトをノードで表し、メッセージの流れをアークで表現して、その間のインタラクション（相互作用）をプログラムとする視覚的プログラミングツール。

オブジェクト間のメッセージ・パッシングの状態がアニメーションによって視覚的に表現されるなど、オブジェクト計算モデルを視覚的に理解できます。作成したプログラムのデバッグへの利用も可能です。



■階層ファイル・システム・ファインダー

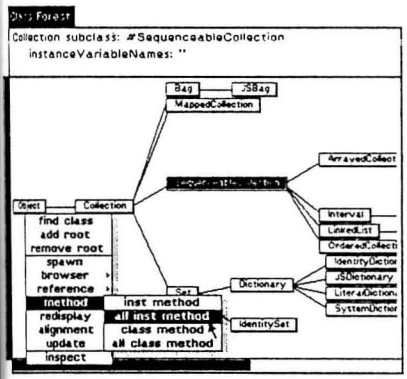
階層構造を持つファイル・システム（階層ファイル・システム）を視覚的に表示し操作できる機能を提供します。



■クラス階層ブラウザー (インヘリタンスブラウザー)

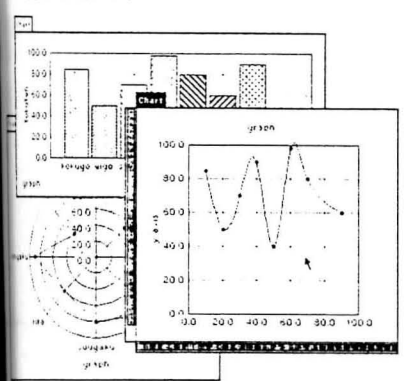
Smalltalk-80における各クラスやメソッドの継承関係をグラフィカルに表示するブラウザー。

各クラスがアイコンとして表示され、各種のブラウザーやメソッドホルダーを起動することができます。また、既存のシステムブラウザーの主な機能も備えています。



■チャートユーティリティ

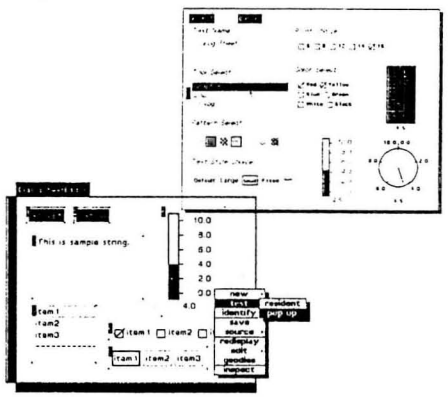
分布、棒、折線、レーダー、インターバル棒、インターバル折線、xy分布、xy折線、xyスプライン曲線の9 種類のチャートが用意されており、用途によって最適なものを選択することができます。また、どのチャートでもデータと表示するチャートの大きさを指定するだけで自動的にスケーリングが行われます。



■ダイアログユーティリティ

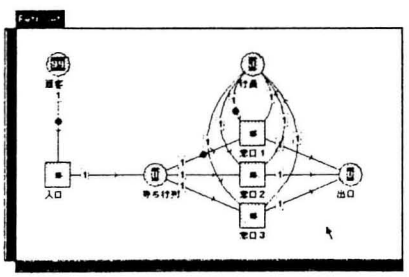
システムとユーザーが対話をするツール。ユーザーに、キーを入力させる場合や、何かを選択させる場合、そして数値を入力させる場合などに役立ちます。また、誤操作や、危険な操作を行おうとした時に、警告をしたり確認を求めたりするパネルとしても使用できます。

ダイアログユーティリティの中には、ダイアログボックスとそれを拡張したダイアログシートがあります。ダイアログシートには、テキスト入力フィールドや選択ボタンの他、ゲージやチューザーなどのパーツもあり、これらを自由に配置することができます。また、これらのパーツをシート上にレイアウトするだけで、実際のダイアログシートを生成させることができるエディタも備えています。



■ペトリネットシミュレーター

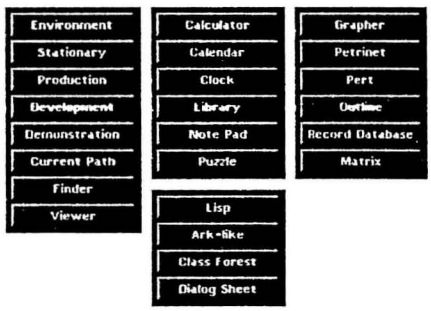
いくつかの事象が並行に発生する状況において、情報の流れや制御を自然でわかりやすい形で記述し、解析することができます。また、このペトリネットシミュレーターは、事象駆動エンジンを持ちトークンの移動をアニメーションによって表現します。



■パーソナル・インフォメーションマネージャー (PIM)

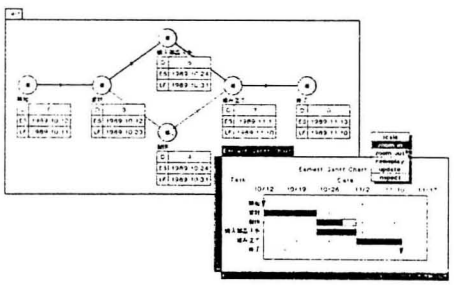
ハイパー・テキスト機能、アウトライン機能、スケジュール機能、レコード検索機能、ファイル検索／結合機能を有機的に結合した個人の情報管理システム。

ここから、GrapherGearのすべてのツールを起動させることが可能です。



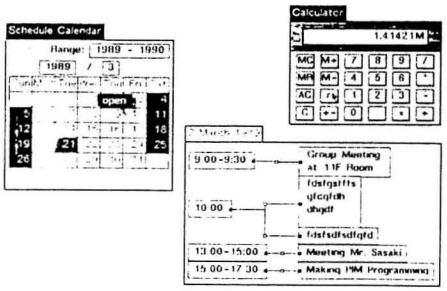
■工程管理ツール(PERTツール)

プロジェクトの日程管理ツールとして、時間の流れと作業の順序関係を表現したフロー型PERT/Timeツールです。



■デスクトップ・アクセサリー

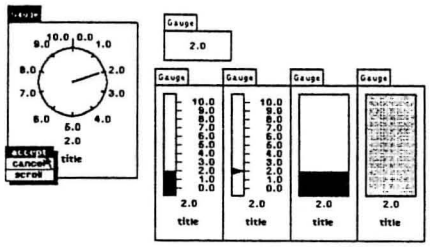
電卓、カレンダー、スケジュール、パズルなど、Smalltalk-80の環境にデスクトップとしての機能を付加します。



■プラガブルゲージ

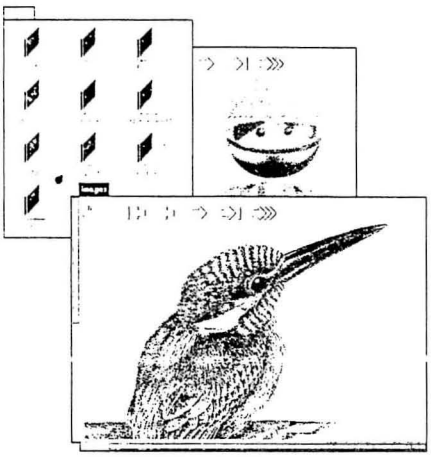
定量的な情報表示やデータを入力させるためのツール。

ゲージには円、棒、デジタル、指針、明暗が用意されており、現実の計器類をシミュレートしていますので、親しみやすいインターフェイスを構築することができます。



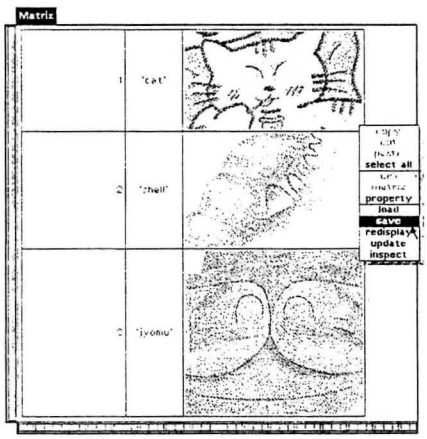
■ドキュメントライブラリー

画像ファイルの集まりを一括管理するライブラリーを作成します。



■マトリックスユーティリティ

行列の内容を対話的に編集するためのインターフェースを備えています。また、マトリックスのセルには数値や文字列の他、絵やゲージ、チャートなどの任意のオブジェクトを格納することができます。



■その他のユーティリティ群

事象駆動ユーティリティ

- 事象駆動のシミュレーションのための機能を提供します。

プロンプトユーティリティ

- ユーザーの答えの選択や文字列の入力を促する基本的なツールの集合です。

並列処理ユーティリティ

- 並列処理を行うための基本機能であるプロミスと時分割を行うタイムスライサーがあり、並列動作プログラムを簡単に開発することができます。

分布関数ユーティリティ

- シミュレーションやオペレーション
  ーチのための分布関数を扱います。

その他

- 文字列の比較のためのメソッド、フ
  ミングのためのメソッド、矩形生成
  めのメソッド、画像生成のためのメ
  ドといったメソッド類や、汎用的な
  キャッシュ、アイコンなどに利用で
  さまざまな画像が登録されたフィギ
  が含まれています。

■その他のプロトタイプ・アプリケーション

ワイヤーフレームユーティリティ

- Smalltalk-80上で簡単な3次元のワ
  ーフレームを描くためのツール。
  x,y,zの各軸について任意の角度での
  が可能です。

レコードデータベース

- レコードデータベースは行がレコー
  列が件名を表すマトリックスからな
  ータベースです。レコードデータベ
  上のデータは、AND/OR検索により
  することができます。また、検索さ
  データからなる新しいマトリックス
  成することも可能です。

※本ソフトウェアを使用する場合は、別途「Smalltalk-80システム・パッケージ」が必要です。
※仕様その他は予告なく変更する場合があります。
※Smalltalk-80は米国パークプレースシステムズ社(ParcPlaceSystems Inc.)の商標です。

●お問い合せは……

富士ゼロックス情報システム株式会社
〒160 東京都新宿区西新宿3-16-6 ☎03-378-8011

ソフトウェア要求定義支援ツール

PWB

(株) PFU

PWBの目的

プロトタイプを用いたソフトウェアへの要求定義,分析
その支援環境をコンピュータ化

→ ソフトウェアの品質と生産性を画期的に向上

具体的には,

○ ソフトウェア開発期間の短縮
○ 実行可能な要求仕様の作成
○ 定義,更新,保守,検証,文書化が容易
○ ユーザの要求をスムーズに引き出すことが可能
○ ユーザとシステム開発者の両方が納得できる要求仕様の作成
○ 目的システムの部分的な先行開発

PWBの特長

◎ ラピッドプロトタイピング

効率的な要求定義や分析,そのプロセスと道具との相互作用
ビジュアルプログラミングの採用

具体的には、

○ 要求定義に図形言語を利用
○ 要求仕様をアニメーション化

◎ モデルベースな要求仕様

実世界のインフォーマルなアプリケーションの概念を,抽象世界の
フォーマルな表現に変換する方法を提供

具体的には、

○ ERA(Entity-Relation-Attribute)モデル
○ 状態遷移図、ペトリネット、キューイングネットワークモデルなど

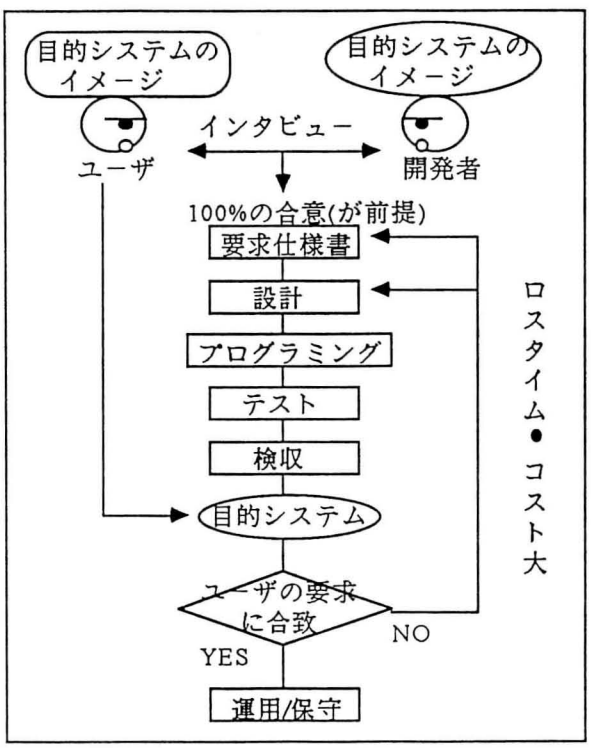
◎ ワークベンチ

ハードウェアとソフトウェアが融合した環境アーキテクチャを構築

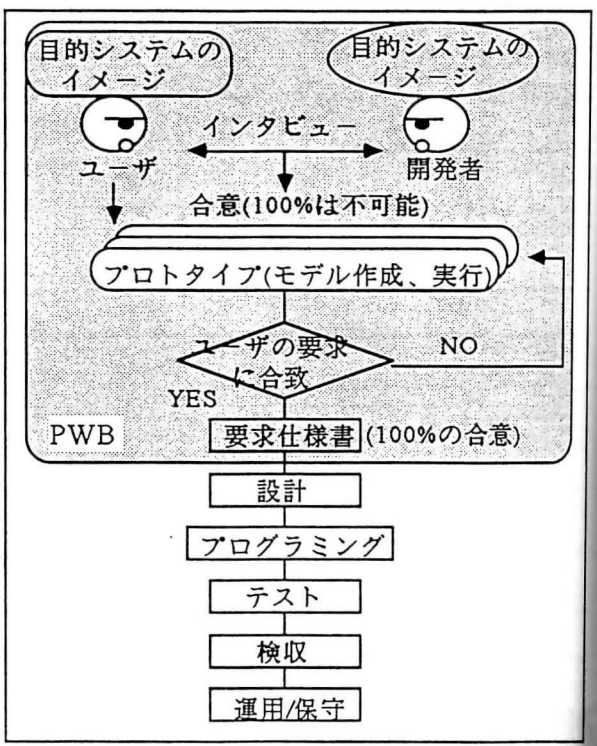
具体的には、

○ ビットマップ付きワークステーション
○ オブジェクト指向システム

ソフトウェアの開発方式



ウォータフォール方式
(従来開発方式)



プロトタイピング方式
(PWB)

PWBの主要機能

◎ 情報管理機能

要求定義、分析に関わるすべての情報を一括管理
要求定義のモデル化や、作成したモデルの疑似実行(アニメーション)の指示
Hyper Card風のビジュアルなMMIを提供

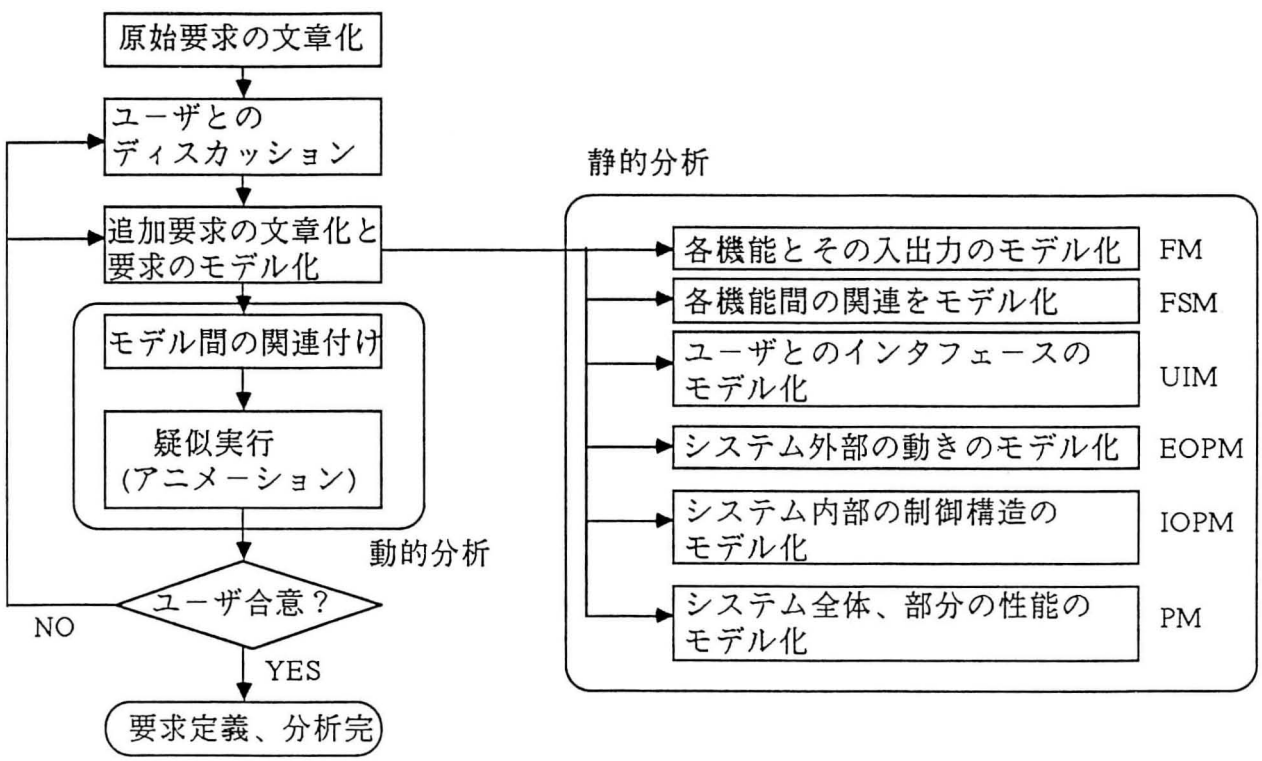
◎ モデル化支援機能

6つの視点からシステムを分析し、モデル化するためのツール群
ある特定の規則に従って容易に要求定義を記述できる図形構文エディタ
ユーザが独自の規則を定義できるメタエディタ

◎ 疑似実行(アニメーション)機能

作成した要求定義のモデルを実行させ、システムの動きを分析、検証
性能要求などの実現性の検証

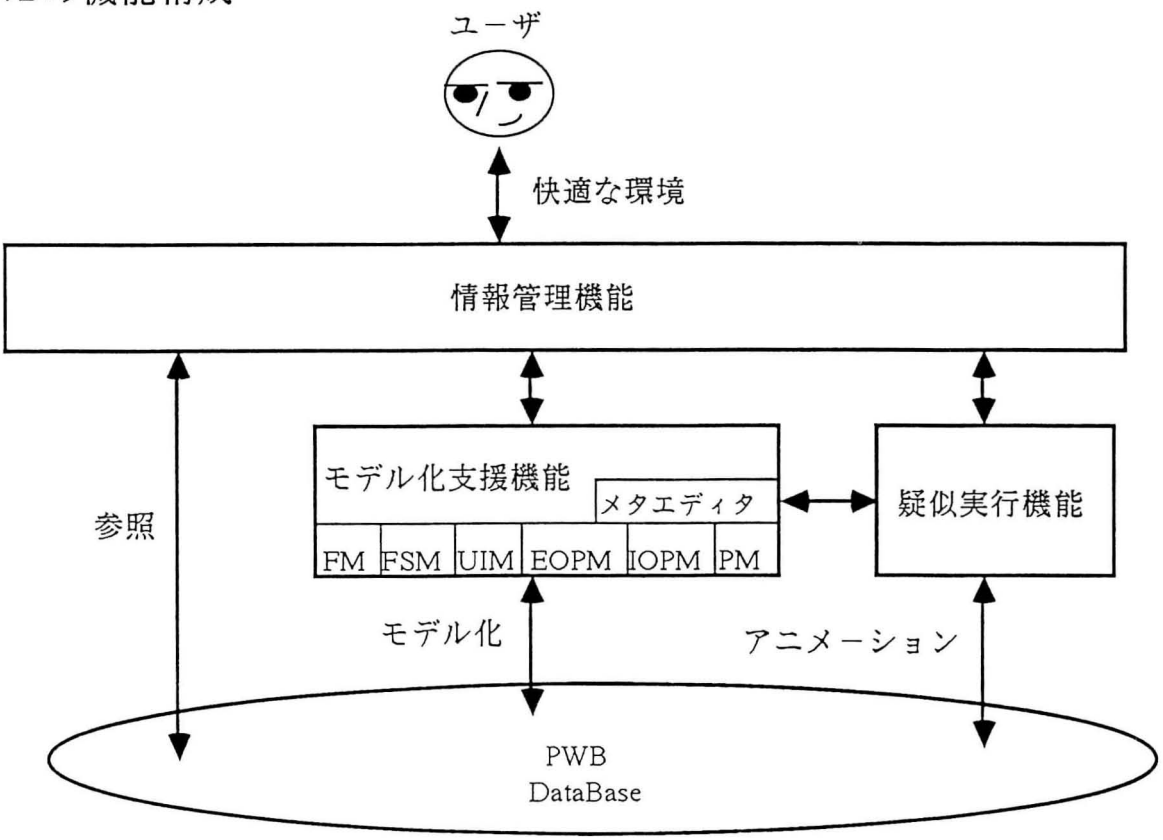
PWBを用いた要求定義、分析手順



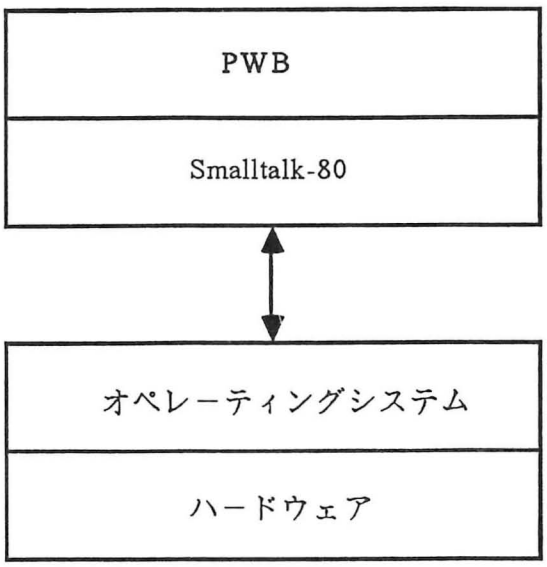
PWBの6つのモデル

モデル	概要	備考
FM(ファンクション モデル)	システムに必要とされる機能,属性,機能の実行を支援する機構を定義する。	
FSM(ファンクショナル ストラクチャモデル)	システムの機能構造をモデル化する。各機能間の関係や機能とオブジェクトの関係を定義する。	
UIM(ユーザインタ フェースモデル)	システムとユーザとのインタフェースや、入出力データの実行をモデル化する。	
EOPM(エクスターナル オペレーショナルプロ ファイルモデル)	システムをユーザの立場からモデル化する。システムの状態がイベントに対してどのように遷移するかを定義する。	状態遷移図
IOPM(インターナル オペレーショナルプロ ファイルモデル)	システム内部における制御構造をモデル化する。プロセスの並列処理,同期処理などの実行状況やタイミング要求を定義する。	ペトリネット
PM(パフォーマンス モデル)	システムの全体または部分的な機能の性能に対する要求をモデル化する。	待ち行列 ネットワーク

PWBの機能構成



PWBの動作環境

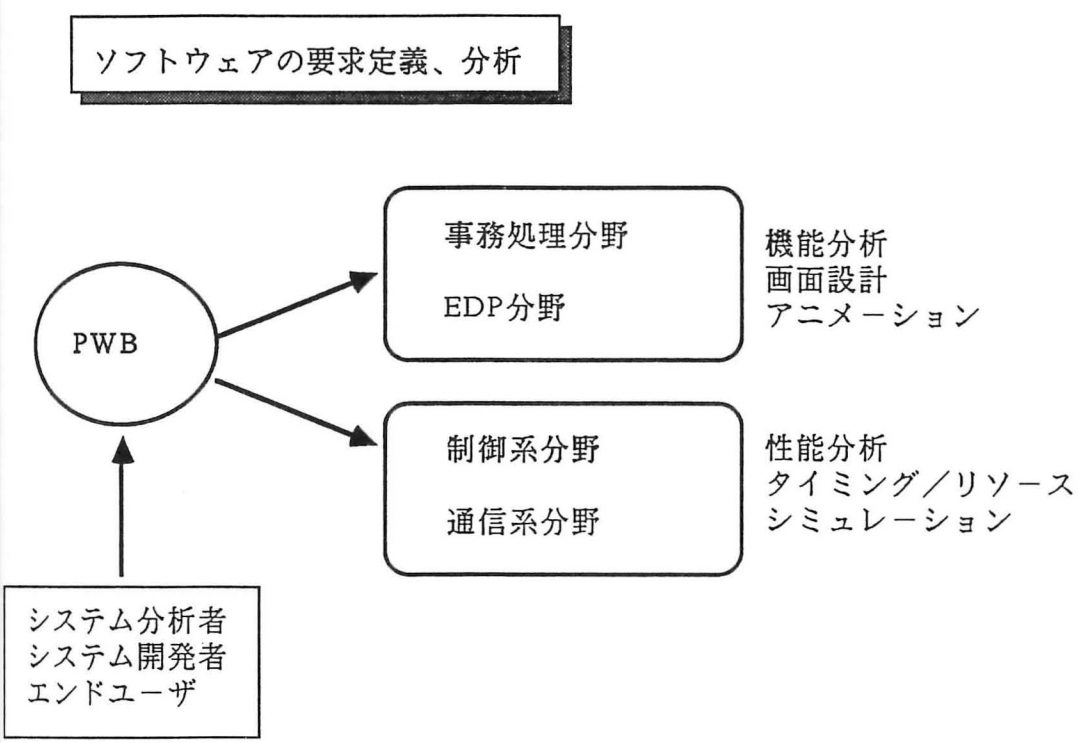


SX/A Smalltalk-80

SX/A
X-window
TISP,JDP

A-50,50s,60,80,Σ230
(マウス
ビットマップディスプレイ(BDA,CDA)
実装メモリ16MB以上)

PWBの適用分野



機能分析
画面設計
アニメーション

性能分析
タイミング／リソース
シミュレーション

Software through Pictures™

《動作環境》

●NEWS日本語版

機種：NWS1700、NWS1800シリーズ
メモリ：8MB以上
システム容量：StP(サンプルデータ込み)25MB
OS：NEWS-OS3.3以上
ウィンドウシステム：X11R3

●Sun-3日本語版

機種：Sun3/80、Sun3/470シリーズ
メモリ：8MB以上
システム容量：StP(サンプルデータ込み)25MB
OS：SunOS4.0.1以上＋JLE1.0.1以上
ウィンドウシステム：SunView

●Sun-SPARC日本語版

機種：SPARCstation
メモリ：8MB以上
システム容量：StP(サンプルデータ込み)25MB
OS：SunOS4.0.3以上＋JLE1.0.3以上
ウィンドウシステム：SunView

《開発元》

米国 Interactive Development Environment,Inc.
(社長 A.I.ワッサーマン、本社 サンフランシスコ)

《問い合わせ先》

株式会社 SRA
CASE営業部
〒102 東京都千代田区平河町1-1-1
TEL 03-234-2623(直) FAX 03-234-8048

*Software through Pictures と Visible Connection は、Interactive Development Environment,Inc の商標です。

Software through Pictures（StP）は、ソフトウェアの生産性／信頼性を飛躍的に向上させるために開発された統合的 CASE (Computer Aided Software Engineering) ツールです。

巨大化、複雑化しているソフトウェア開発分野のなかで最も重要な要求分析/設計工程においては、コンピュータ支援環境の構築が急務となっています。近年、そのような問題を解決する手段として、世界的に注目を浴びているものがCASEです。

StPは、高性能ワークステーション上にインプリメントされており、複雑かつ大規模なシステムの分析/設計作業への適用を可能にするための優れたユーザ・インタフェイス、マルチ・ユーザ環境、 ネットワーク環境を備えています。

《主な機能・特徴》

●オープン・アーキテクチャ

StPはVisible Connectionsと呼ばれるオープン・アーキテクチャの原理を用いて開発されています。

これにより、環境を拡張/カスタマイズすることを可能にし、プロジェクトや組織ごとに異なるさまざまな要求に応じることが可能になります。

●複数の方法論のサポート

構造化分析、リアルタイム構造化分析、構造化設計、プロトタイピング支援などの複数の方法論をサポートしています。開発対象アプリケーションの特性により最適な方法論を組み合わせて利用が可能です。

●統合的データ辞書による管理

データ辞書の自動生成と豊富なチェック機能があります。これらの機能はStPの各エディタと初期メニューから簡単に起動することができます。

●統一されたインタフェイス

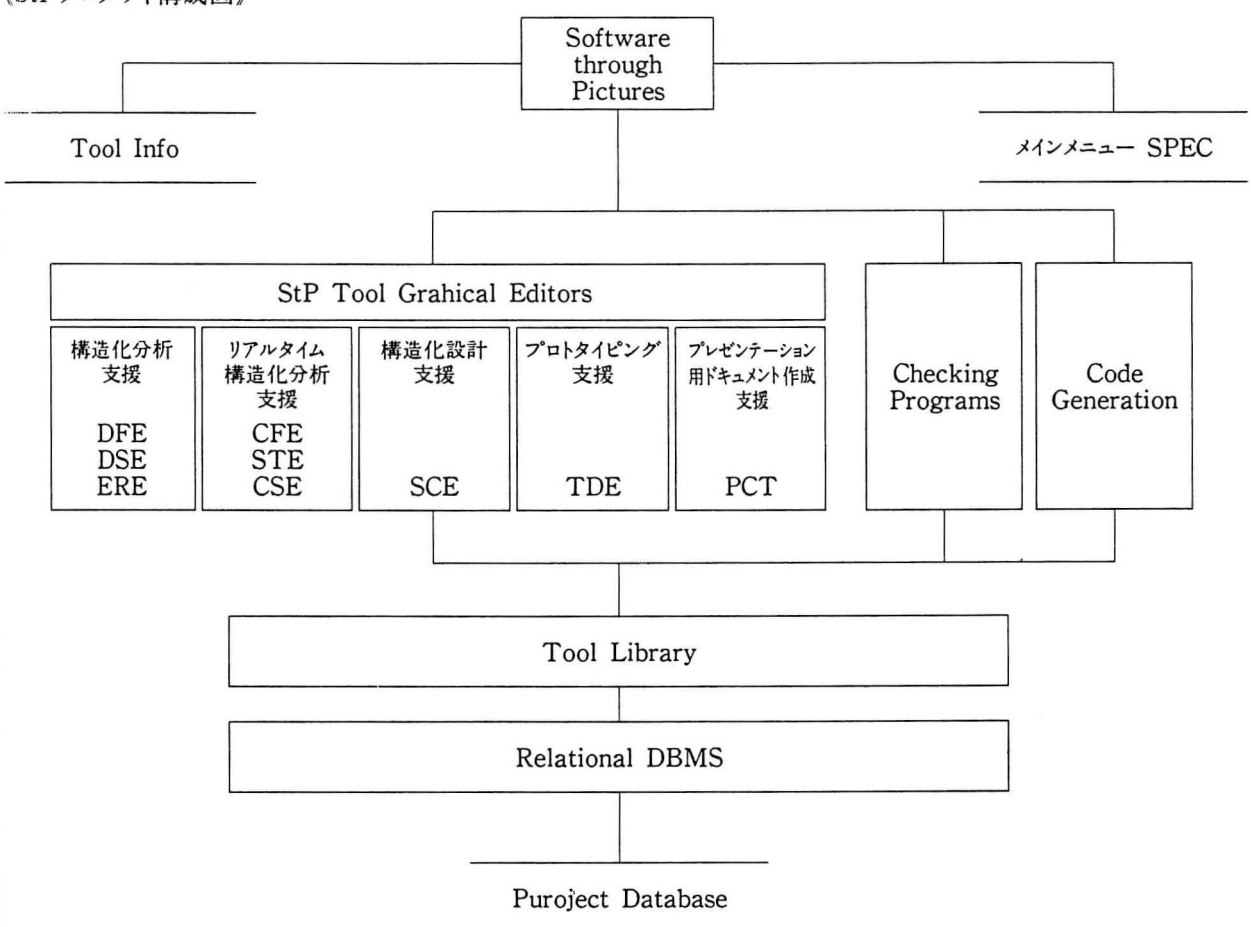
ウィンドウ・システムを利用した簡単で統一的なユーザ・インタフェイスを用いることにより、各々のエディタを使って簡単にダイヤグラムの編集、描画を行なうことができます。

●大規模システムへの対応

プロジェクト・データベース、バージョン管理、ロック機能などを利用したマルチ・ユーザ環境を実現し、また、ワークステーションのネットワーク機能を利用することで大規模システム開発への適用が可能です。

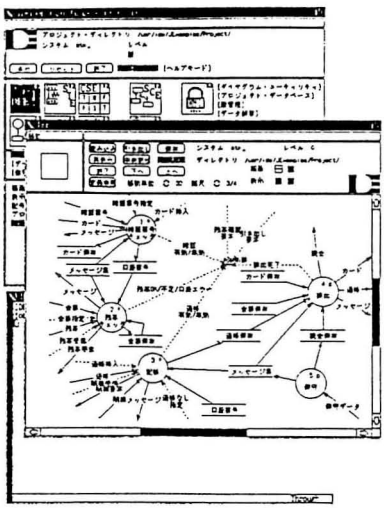
データ辞書構築のためにTroll/USEデータベース・マネージメントシステムを使用しており（約500ページのダイヤグラム情報とデータ辞書の容量は約4MB）、開発対象とするシステムの規模は無制限となります。

《StPプロダクト構成図》



構造化分析支援

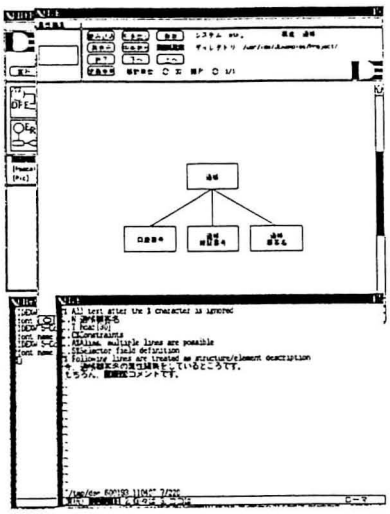
Dataflow Diagram Editor (DFE)



データフロー・ダイヤグラムを描くためのエディタです。このエディタでは、Gane/Sarson、DeMarco/Yourdonの２つの表記法をサポートしています。

記述したデータフローからデータ辞書を生成するプログラムやダイヤグラムをチェックするプログラムも用意されています。

Data Structure Editor (DSE)

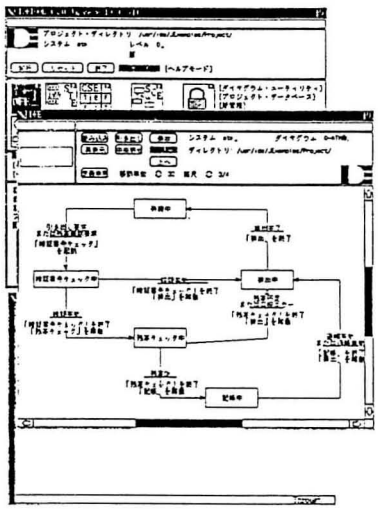


データ構造ダイヤグラム・エディタは、ジャクソン記法を用いてデータ構造を描くためのエディタです。

ダイヤグラムからデータ辞書を生成することもできます。また、ダイヤグラムで定義したデータ構造について、Ada、C、Pascal言語のためのデータ宣言を生成するユティリティも用意されています。

リアルタイム構造化分析支援

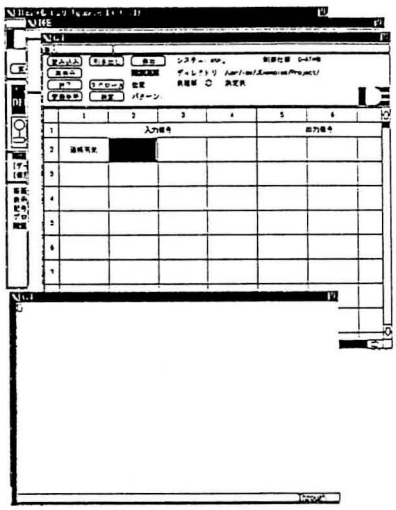
State Transition Editor (STE)



状態遷移ダイヤグラムを描くためのエディタです。

状態遷移ダイヤグラムは、制御仕様の中のリアルタイムな部分をモデル化するために使用します。

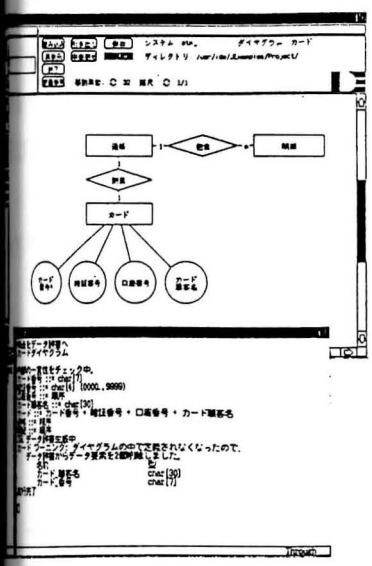
Control Specification Editor (CSE)



制御仕様エディタは、制御を定義するためのエディタで制御仕様定義表として、決定プロセス活性表、プロセス活マトリクス、状態遷移表、状態トマトリクス、イベント論理表作論理表をサポートしています

定義した制御仕様からデー書を生成するプログラムや制様のチェックをするプログラム用意しています。

ntity-Relationship Editor (ERE)

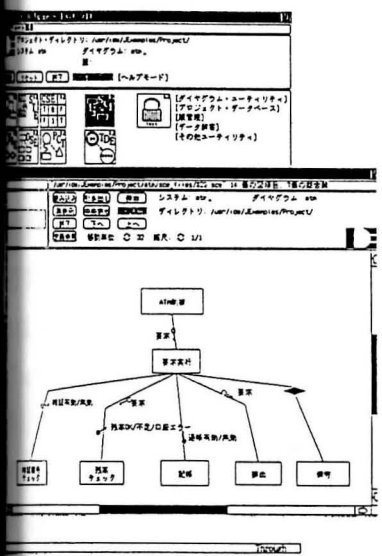


チェンの記法によるER(エンティティ・リレーションシップ)ダイヤグラムを描くためのエディタです。

ダイヤグラムからのデータ辞書生成機能や、Troll/USE、Ada、C、Pascalのデータ宣言部の生成する機能も備えています。

構造化設計支援

tructure Chart Editor (SCE)

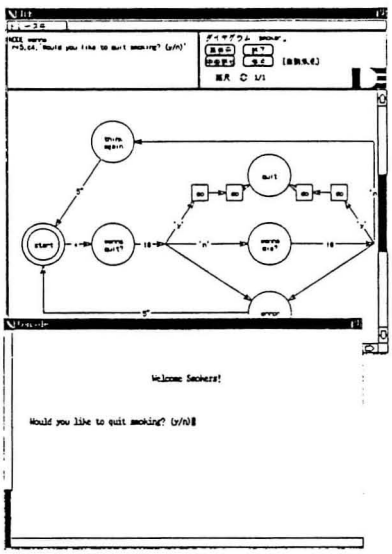


構造化チャートを描くためのエディタです。ダイヤグラムからデータ辞書を生成することも可能です。

インターフェース表生成の機能やダイヤグラムからのプログラム・スケルトン(骨格)生成の機能も備えています。

プロトタイピング支援

Transition Diagram Editor (TDE)

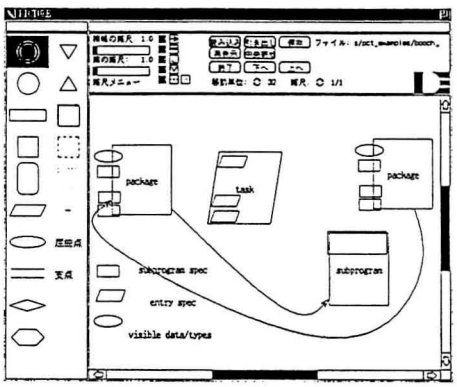


TDEは、USE (User Software Engineering)遷移ダイヤグラムを描くためのエディタです。

USE遷移ダイヤグラムから対話型システムのプロトタイピングを記述するためのプログラミング言語、RAPID/USEを生成することができます。

プレゼンテーション用ドキュメント作成支援

PiCTure (PCT)



PiCTure(PCT)

PiCTureは、どの方法論にも縛られないプレゼンテーション用の図を描くためのエディタです。

CADRE/TOYO

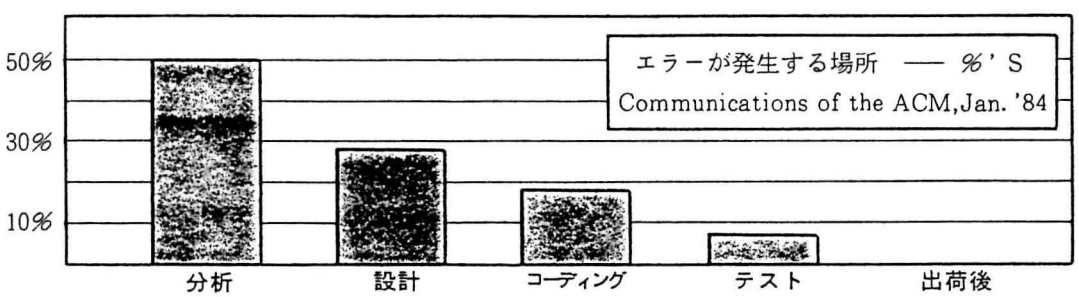
"Teamwork"

—ソフトウェア・システム生産性向上ツール—

近年の商品機能等の複雑化、IC,ASICの発展によるハードウェアのコスト低下に伴い、開発コストに占めるソフトウェアの割合は、年々増加の一途をたどっており、現在60〜80%を占めるまでに至っています。またこれに伴ってシステムの機能性能も、ソフトウェアに依存する部分が非常に大きくなってきました。

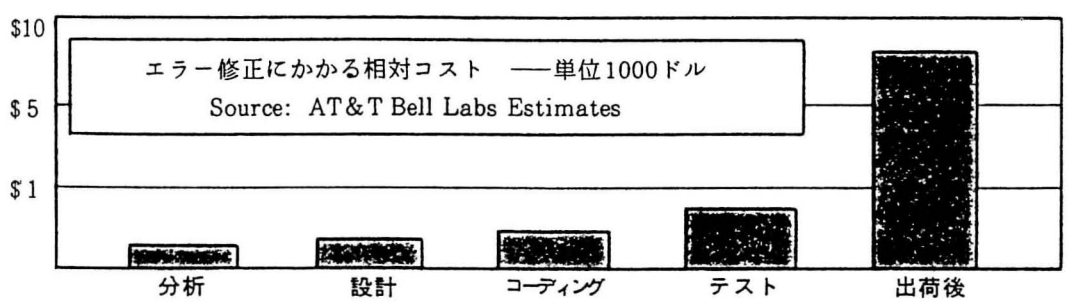
そこでソフトウェアを予算内で、予定納期で、高品質に作り上げることが、重大な課題となって来ております。ところがシステムが複雑になればなるほど最終製品に見過すことのできない不良が入り込む可能性が大きくな結果的にそれが費用の増大、工程の遅れなどの大きな因になっています。

下図は、ソフトウェアにおける不良が、どの段階でど位の割合で発生するものかを調査した結果です。



このように不良は、上流工程の分析・設計段階で組み込まれるものがほとんどです。またこの上流工程で組み込まれた不良に気付かずに作業を進めて行きますと、テスト、デバックなどの下流工程でこれを修正するには大変なコストがかかることも判っております。（下図の通りもしも分析段階で組み込まれた不良が最終工程まで気づかれずに残っているとそれを修正するための費用は倍以上になるという調査結果があります。）

つまり、上流工程の分析、設計での詰めを充分に行とが、予算、工期、品質を守るための、非常に重要ファクタとなります。



ところが、従来の文章を中心とした仕様記述では、システム開発の関係者（アナリスト、設計者、ユーザ等）の間の十分な相互理解を図ることが難しく、仕様の中の冗長性も大きくなってしまうため、分析段階での不良除のためには十分ではありませんでした。

そこで、1970年代中頃に従来の文章記述を中心としたシステム分析の手法にかわり、図を中心とした記述手法である構造化分析が提唱されました。この手法には従来の文章記述手法に比べて、次のようなメリットがあります。

○図を中心にした表現によるコミュニケーションの改善
　誤解により発生する不良の排除

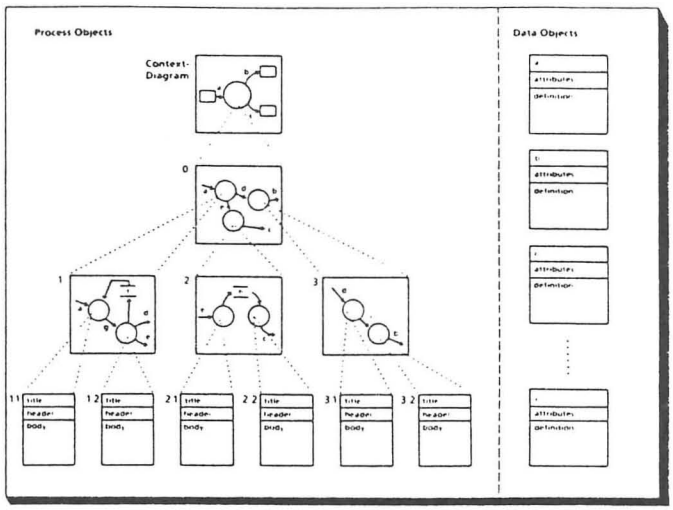
○階層的分割による問題の単純化
　明快な機能分割
　見積の確度向上
　冗長性の最小化
　メンテナンス性の向上

ところが、この原理的に素晴らしい手法も手作業で、これを行うのは特にその一貫性及び定義忘れ部分のチェック、修正作業等に大変な手間がかかるため、近年に至るまで一部での採用に限られていました。

そこでCADRE社では、ワークステーションのコンピュータパワーと図形描画能力を十分に生かし、この手作業を自動化するツールを提供し、この問題を解決しました。製品は“Teamwork”と呼ばれ、手法の実施に伴う煩雑な手作業部分を自動化し、支援するツールです。また、Teamworkには構造化分析用ツールのみでなく、それによって作成されたデータを生かしてプログラム設計を行う構造化設計手法をサポートするツールも用意されています。

Teamworkの各々のエディタは、構造化分析・構造化設計の標準ルールをベースに働き、必要な図、テキストの作成をお手伝いします。従って手法について初心者の方でも、ルールを守って標準手法で分析・設計を進めていただけます。図、テキストのチェック機能が用意されており、必要に応じて、必要な部分だけの矛盾、定義忘れ等を自動的にチェックします。従って、システム分析の担当者はその主要業務である分析の作業に神経を集中でき、難しいシステムを理解のしやすい小さなモジュールにまで階層的に分解することにより、明確で、相互理解の得易い分析結果・システム仕様を導き出すことができます。ソフトウェア設計者は、この分析結果を基に、そこに記述されたデータ定義などを有効に利用して、モジュール性の高いソフトウェアの階層的構造を設計することができます。

構造化分析（システムの階層的表記）



Teamworkの様々なエディタで作成された図、テキストはすべて同一のプロジェクトデータベースにストアされます。従って大きなプロジェクトでも全てのエンジニアが作成したデータを一元管理できます。また、プロジェクトメンバ間のコミュニケーションも改善できます。データベースの全ての要素は用意された標準ライブラリルーチンを使ってアクセスできますので、これを使って文書作成プログラム、工程管理プログラムなどとのインタフェースを簡単にとれます。それによってTeamworkをシステム開発用統合環境の核とすることが可能です。

Teamworkを御使用いただくことにより、ソフトウェアシステムの開発をより効率的にお進め頂け、生み出される製品もより高品質のものになるものと確信いたしております。

Teamworkの構成

■Teamworkの分析／設計ツール

Teamworkは、組み込みシステム、ソフトウェアシステムの開発の上流工程作業を構造化分析・構造化設計で支援するため、５つのエディタを備えています これらはすべてルールに従った図、テキストを適切に記述できるようなポジションセンシティブなエディタになっています。また出来上がった仕様書の一貫性、完全性をチェックする機能も備えています。

(1) 構造化分析ツール

３つの視点からシステムが“何”を実現するかを記述。違った角度からシステムを捕らえることにより、その姿が明確に定義できます。図を中心とした表記により関係者間の誤解を排除します。機能の明確な小単位のモジュールに分割することにより、システム仕様を細部まで明確にします。

- Teamwork/SA
  システム機能の視点から記述
  データの流れと処理の組合せで記述
- Teamwork/RT
  ＳＡで記述される機能仕様に時間とコントロールの視点を付加記述
- Teamwork/IM
  情報の視点からのシステム記述
  エンティティ関係図による情報モデリング

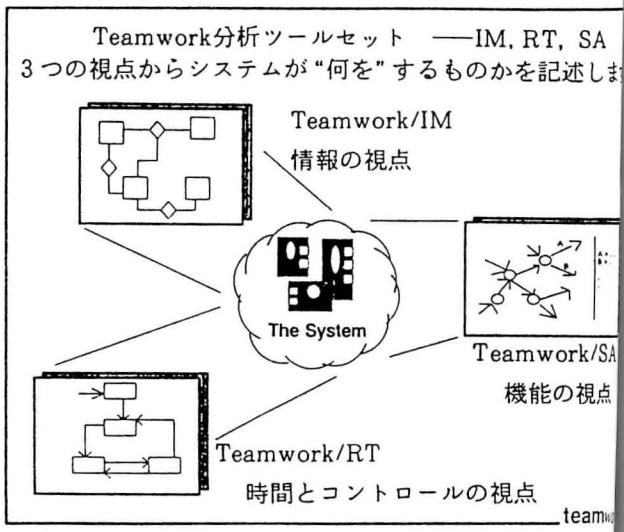
(2) 構造化設計ツール

プログラム構造の設計作業を支援

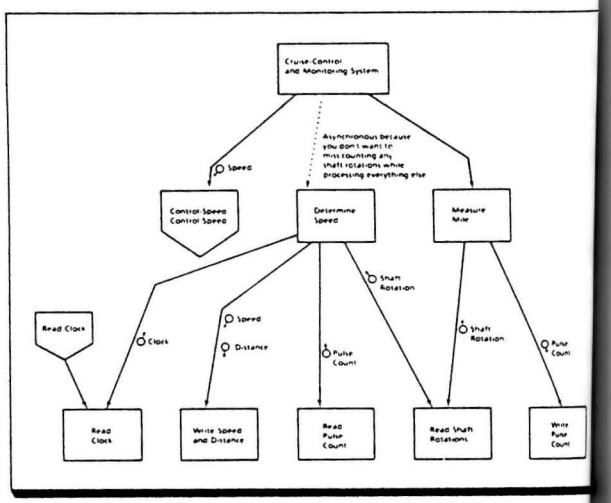
分析結果を生かし、そのソフトウェアによる実現方法を記述します。構造の明確なモジュール構成のプログラムはデバック作業の短縮、メインテナンス性の向上をもたらします。設計データからC,Adaのソースコードのフレームを自動生成するユーティリティも用意されています。

- Teamwork/SD
  ストラクチャ・チャートを用いたプログラム設計用ツール
- Teamwork/ADA
  GEとの提携によるAdaプログラム設計用ツール
  Buhrの提唱したAdaストラクチャグラフによるプログラムの構造記述

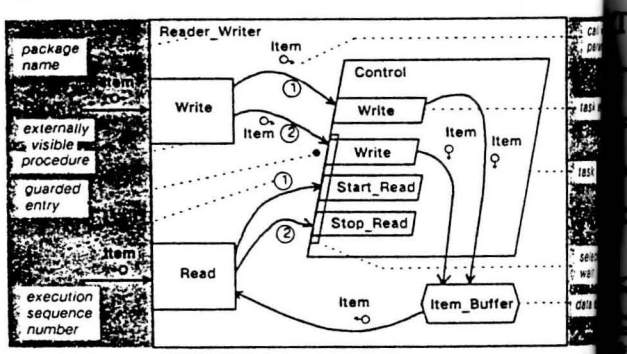
Teamworkの構造化分析ツール



構造化設計（ストラクチャ・チャート）



構造化設計（Adaストラクチャ・グラフ(ASG)）



■統合化プログラミングサポート環境

IPSE:Integrated Programming Support Environment

Teamwork の５つのエディタで作成されたデータはすべて、一元管理されたプロジェクトデータベースに蓄えられます。このデータをホストコンピュータが備えている様々なソフトウェアに有機的に結合することによって、それを単なる分析・設計の結果ではなく、統合的な開発環境の核とすることが出来ます。プロジェクトデータベースはオープンアーキテクチャになっており、"Teamwork/ACCESS" プログラムライブラリがこの統合化を可能にします。

また、システム開発の全ての作業を Teamwork のウインドウ環境で行うため、各ウインドウにユーザメニューを追加できるようになっています。これにより、Teamwork の中からホストのユーティリティやユーザプログラムを起動できます。

Cadre社でも、Teamwork/ACCESS を利用したユーティリティを提供します。現在次の物が用意されています。

●Teamwork/DPI

—Document Production Interface

Teamworkで作成したデータを元にした文書自動生成機能

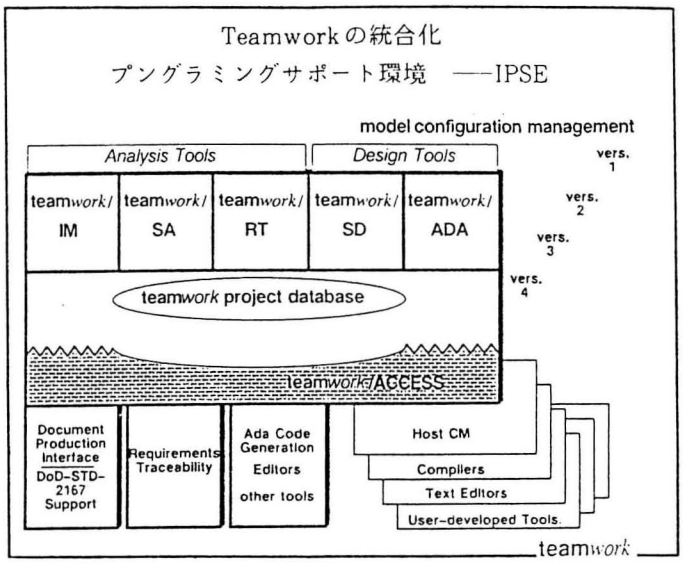
●Ada ソースコードジェネレータ

Teamwork/ADA で作成された設計書からAdaのコードフレームを自動生成

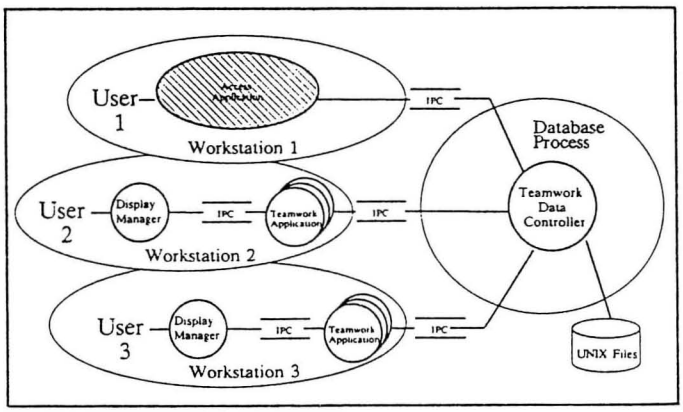
■ワークステーションのネットワーク機能を十分に生かしたプロジェクト環境

Teamwork のプロジェクトデータベースはネットワーク環境ではサーバ上に作成されます。従ってプロジェクトメンバ全てがこれを共有することができ、コミュニケーションの改善、データの分散の回避が出来、よりスムーズなプロジェクトの進行が図れます。

Teamwork/IPSE



Teamworkのプロジェクト環境



Teamwork について、より詳しい資料等の必要な方は下記までお問い合せ下さい。

㈱東陽テクニカ　エレクトロニクス事業部
応 用 技 術 部
(TEL) 03−279−0771

東陽テクニカ

株式会社 東陽テクニカ エレクトロニクス事業部 応用技術部

本　社　〒103　東京都中央区日本橋本石町1-1-2(大東ビル)……☎(03) 279-0771
大阪支店　〒542　大阪市南区南船場2-8-7(三井製糖ビル)……☎(06) 262-3471
水戸営業所　〒312　茨城県勝田市中央町7-3……☎(0292)73-3421
つくば営業所　〒305　茨城県つくば市竹園3-641-1(竹園SC206号)……☎(0298)51-1366
横浜営業所　〒220　横浜市西区北幸町2-9-40(銀洋ビル604号)……☎(045)314-0781
名古屋営業所　〒465　名古屋市名東区高社1-263(一社中央ビル)……☎(052)772-2971
厚木テクニカルセンター　〒243-01　神奈川県厚木市森の里若宮11-1……☎(0462)47-6331

AN-CTI-103-1-89071-7-2SH