ソフトウェア開発環境 dmCASE†

松浦 佐江子†† 大林 正晴††

現在，ソフトウェア開発の新しい技術であるプロトタイピング・オブジェクト指向・形式的仕様記述・AI・自動プログラミング等が研究されているが，ソフトウェア開発における確固たる設計の枠組みはまだ確立されていない．このような新しい技術の上に立ったソフトウェア開発のライフサイクルを考慮した統合的な開発環境の実現が重要な課題である．本論文では，Balzer らの提唱するソフトウェア開発のパラダイムにおけるソフトウェア開発環境 dmCASE を提案する．ソフトウェア開発工程の早期において仕様レベルの検証・保守を行うためには設計方法論と実行可能な形式的仕様記述が必要である．さらに環境として人間の思考過程に即した柔軟なユーザインタフェースをもつ環境が重要な役割を果すと考える．より知的な作業にのみ従事するために，dmCASE は設計方法論・形式的仕様記述・支援ツールの統合化された環境を目指している．これら3つの柱に対する提案と記述実験を通じての評価について報告する．

1. はじめに

現在，ハードウェアの発展に伴いプログラミングを助ける様々なツールや柔軟なインタフェースが多数現れている．しかし，1990年代のソフトウェア開発が抱える本質的な問題は解決されていない．ソフトウェア開発の新しい技術であるプロトタイピング・オブジェクト指向・形式的仕様記述・AI・自動プログラミング等が研究されているが[2)-4)]，ソフトウェア開発における確固たる設計の枠組みがまだ確立されていない．そこで，こうした便利な道具や新しい技術の上に立ったソフトウェア開発のライフサイクルを考慮した統合的な開発環境の実現が重要な課題であり，生産性，保守性，信頼性の向上といったソフトウェア開発の問題解決への道ともなると考える．われわれは，Balzer らの提唱するソフトウェア開発のパラダイムにおけるソフトウェア開発環境 dmCASE を提案する．ソフトウェア開発工程の早期に仕様レベルにおける検証・保守を行うためには設計方法論と実行可能な形式的仕様記述が必要である．さらに環境として人間の思考過程に即した柔軟なユーザインタフェースをもつ環境が重要な役割を果すと考える．dmCASE は設計方法論・形式的仕様記述・支援ツールの統合化された環境を目指している．また，設計の対象を3次元の構造として捉える空間設計と，それを平面で切って2次元構造として設計する平面設計との2つの視点による設計を提案する．

† The Software Development Environment: dmCASE by SAEKO MATSUURA and MASAHARU OHBAYASHI (Kanrikogaku, Ltd.).

†† (株)管理工学研究所

本論文では2章においてソフトウェア開発環境に対するわれわれの考え方を論じる．つぎに，3章において設計方法論「概念による設計法」とその特徴を4章において形式的仕様記述言語について，5章ではツールによる支援について述べる．そして6章では dmCASE の構成および特徴を説明する．最後に7章において，dmCASE における，記述実験による評価および問題点について述べる．

2. ソフトウェア開発環境

ソフトウェア開発のライフサイクルのパラダイムとして Balzer らの提唱するパラダイム[1)]における環境を考える．ソフトウェア開発工程の早期における仕様の獲得および検証・保守はこれまでの開発状況に比べて，より知的な作業に従事することを可能にすると考えるからである．開発環境に必要なものとしてつぎの4つが考えられる．

- 曖昧な要求から問題のモデルを構築する（仕様を獲得する）ための方法論
- 検証可能な形式性と実行可能性をもつ仕様記述言語
- 方法論および検証を支援する技術，すなわち，ツールによる支援
- 実システムへの変換

方法論に従って形式的な仕様記述を行い，それと同時に形式性による静的な検証およびプロトタイプ的実行による動的な検証を行う．すなわち形式性の高い仕様記述言語を用いて曖昧さや誤りのない仕様を生成することによってソフトウェアの信頼性を高め，かつ手戻りを減らして生産性を向上させることができると考える．プログラム作成と異なるところはシステムの設

計図を作成することと共に抽象度の高い仕様記述を行うことである．すなわち，実システムの環境（マシンの特性，ウィンドウの構成等）に依存しないレベルで記述を行う．実際のプログラムは仕様を具体化することでも生成できる．

われわれの提案するソフトウェア開発環境 dm-CASE においては上記の要素をつぎのように具現化した．

- 設計方法論としてオブジェクト指向的な「概念による設計法」
- 形式的仕様記述言語として関数型言語 ML

さらにツールとしての支援に対する基本姿勢はつぎのとおりである．

- 人間の思考過程に即したユーザインタフェース
- 本質的に設計に関わる仕事だけに従事できる環境
- 思考を妨げない制約
- 検証

3章において説明する「概念による設計法」は拡張性および保守性を考慮したモデル化を行える．このことは，ここで述べる仕様レベルにおけるソフトウェア開発工程を一括管理する上で重要なことである．ただし，実システムへの変換については，まだシステム化を行っていない．

3. 設計方法論

本章では，仕様のモデル化技法としてわれわれが採用した「概念による設計法 (Design Method based on Concepts 略称 DMC)」[11] について説明し，他のモデル化技法との比較を行いながら，その特徴を述べることにする．今後，本設計方法論を DMC と記述する．また，以下の説明において DMC での特殊な用語については「」で示した．

3.1 DMC の基本概念

DMC はシステム構造のオブジェクト指向的な設計法であり，抽象データ型が基礎になっている．設計の指針となるのは，要求の中に現れる具体的な＜もの＞やそれらを抽象化した＜概念＞である．これを「チップ」と呼ぶ．要求はこうした「チップ」の集合と考えられる．そして，これらの相互関係を2次元のネットワークとして表現し，システムのモデル化を行う．このネットワークを「概念構造図」と呼び，概念構造図上に配線された「チップ」を「インスタンス」と呼ぶ（**図1参照**）．

「チップ」は＜もの＞や＜概念＞に関連するデータやプロセス（＝機能）の集合体である．こうしたデータやプロセスを表す＜もの＞や＜概念＞も広義の「チップ」であり，あらかじめ要求の中から抽出したり新規に生成し，「チップ」の集合体を作成する．この広義の「チップ」の集合体を「単語表」と呼ぶ．「単語表」は情報整理と設計のための“断片”の登録表の役割を果す．

例えば，“西麻布図書館の図書館システムを設計せよ”という要求があったとする．この時要求の中に現れる“西麻布図書館”は具体的な＜もの＞であり，“図書館”はこれを抽象化した＜概念＞に相当する．要求を分析して，“図書館はさまざまな本の集りを操作して，登録や検索などの仕事を行う”と考える．この中から広義の「チップ」として“図書館”，“本”，“登録する”，“検索する”などの単語が抽出できる．“図書館は本の集りを操作する”ことを“図書館”に関することが“本”に関することを参照して仕事を行うと考えて“図書館”と“本”の間に関連があるとする．この関連を「概念構造図」上の有向線分として記述する．すると，「チップ」“図書館”と“本”が図書館システムのモデル内で「インスタンス」として具現化される．

「インスタンス」の設計は抽象データ型（オブジェクト）の設計であり，情報隠蔽を実現する．ただし，「インスタンス」はデータに対するアクセス機構だけでなく＜概念＞に付随するプロセスも含む意味ではいわゆるオブジェクトである．そして，「インスタンス」は，固有のデータだけでなくそれに関連するデータをも含めた複合的な抽象データ型である．例えば，上記の“本”インスタンスはデータとして“本”を扱い，データ“本”はその属性である“書名”・“作者”・“分野”といったデータにより構成され，それぞれの属性を取り出す機能をもつと考える．このように“本”という＜概念＞に関連する4つのデータと3つの機能の集合として“本”インスタンスを設計できる．

上記のように“図書館”と“本”とから構成される

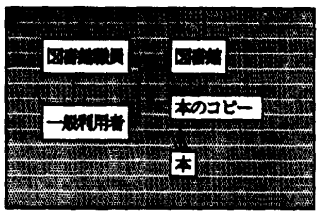


図1 「概念構造図」
Fig. 1 An example of "The conceptual structure chart."

モデルにおいて，必要なデータや要求された機能を設計する．例えば図書館の仕事として“ある分野の本の検索”を考える．このとき，本に対して“本の分野を取り出す”というメッセージを送り，検索のための情報を得る．言いかえれば，“本”インスタンスの機能である“本の分野を取り出す”を“図書館”インスタンスが参照して，“図書館”インスタンスの“分野で検索する”機能を実現するということである．また，それぞれの「インスタンス」に固有のデータは有向線分の下位方向にあるものを参照できる．このように「インスタンス」は「チップ」を「概念構造図」上で他の「インスタンス」と配線したものであり，「チップ」の具現化とは，「チップ」のもつ記号に新たに接続した「インスタンス」の記号を付け加えることである．

このように「インスタンス」のネットワークで構成されたモデルを基にして，データやプロセスを「インスタンス」で象徴される＜概念＞に付随して整理しながら設計を行う．

また，「インスタンス」の仕様とは「インスタンス」のもつ＜概念＞を形式的に記述したもので，記号群と宣言群から構成される．記号群とは「インスタンス」に所属する「記号」の集合であり，具体的にはデータやプロセスを表す単語群である．宣言群は「記号」のインプリメンテーションの集合である．すなわち，記号群は「インスタンス」が他の「インスタンス」に唯一公開しているインタフェースであり，宣言群の情報は外部に対して隠されている．「チップ」の具現化には「参照」によるものと「継承」によるものの２種類がある．「参照」は相手のプロセスのみを自己プロセスに呼び出すことができる関係である．「継承」は相手の「インスタンス」をオブジェクト指向でいうスーパクラスとしてプロセスの依頼を行う関係である．このような「概念構造図」と「インスタンス」の設計により，データの透過性とプロセスの隠蔽が表現できる．

上記のように「概念構造図」を構成する一方，個々の「インスタンス」の仕様を形式的仕様として記述する．すなわち，各「インスタンス」の＜概念＞に付随したデータの構造やメッセージの内容を形式的仕様記述言語により規定する．なお，「インスタンス」仕様が記号群と宣言群に分離されている１つの理由として，同じ記号をもつ「インスタンス」のモジュール部品が生成できることがある．すなわち宣言群は記号群の記号のインプリメンテーションであるから，異なるインプリメンテーションを適用して様々な「インスタンス」を実現することができる．

DMC による設計の手順および，その時点における判断の基準を**表１**に示す．なお，ここで用いた例題は図書館問題[10]である．

曖昧な要求の中から，問題の大まかなモデルを構成し，構成要素である「インスタンス」を設計していく過程において，共通の概念で捉えられるものを１つの塊として認識し，整理しようとするのが DMC である．このように共通の＜概念＞で整理されたモデルは理解性や拡張性が高められ保守においても有効であると考える．

それでは，DMC は他のモデル化技法とどこが違うのだろうか．データフローモデルでは処理間で受け渡されるデータに着目してシステムのモデルを構成し，E-R モデルでは，entity（実体概念）とその関係からモデルを構成する．この entity は生成・消滅の概念をもつ実体であり，DMC での＜概念＞のようなデータとプロセスの複合体ではない．DMC では，データとプロセスを一体として考え，動作を考えて統一的に記述する．一方，上記の技法ではシステムをデータの視点から分析し，それとは別にプロセスの分析を行っている．しかし，人間の思考においては，データのみとかプロセスのみといった見方よりそれらが混在した＜概念＞として物事を捉えているのではないか．DMC でのモデルは，システムを理解するために頭の中でシミュレーションを行いながら作り上げるメンタルモデルに近いと思われる．設計を行う際に，ある特定の視点から考えることは一方でわかりやすいが全体を把握するには不十分である．DMC では人間の思考に近い自然なモデルを構築し，それを基盤として詳細な設計を行い，それらを視点を変えて見ることにより，システム構築の理解を可能にすることを提唱している．すなわち DMC では全体像を「概念構造図」によって把握し設計を進め，ツールの支援によって様々な視点での分析を可能にしている．

また，オブジェクト指向（Smalltalk など）との相違は何であろうか．オブジェクトにはインスタンスとクラスの概念があり，オブジェクト間のメッセージのやりとりのほかにクラスの上位下位すなわち継承関係をもっている．DMC ではオブジェクトとしては「チップ」と「インスタンス」があるが，設計においては関係が規定された「インスタンス」のみが意味をもつ．

すなわちクラスを定義してインスタンスを生成するのではなく，はじめから「インスタンス」の設計を行うのである．設計の始めから抽象度の高い階層的思考は困難であり具体的なものから考えるべきである．「チップ」に関しては後に部品化・再利用に絡めて述べることにする．また，DMC ではオブジェクト間のメッセージのやりとりを「概念構造図」によりグラフィカルに提示している．「概念構造図」における「インスタンス」の上位下位関係には「参照」関係と「継承」関係がある．「参照」がメッセージの受け渡しである．「継承」はいわゆるスーパクラスの関係を定義するものであるが，「継承」関係の「インスタンス」を1つのオブジェクトとして認識する手段であり，あくまでも2次元の構造の中に存在する．この「参照」と「継承」の使い分けによって情報隠蔽の制御が可能である．

3.2 DMC の拡張

3.1 節で DMC の基本概念を説明したが，複雑で大規模な実際のシステムの設計に対応するには1枚の「概念構造図」の記述では不可能である．そこでこの問題に対処するために DMC の拡張について述べる．

これまでの DMC で扱う基本単位を「ボード」と呼ぶ．すなわち1つの「概念構造図」で表されるモデルの単位である．「ボード」の設計は2次元の平面的な設計である．そこでその前段階として3次元の空間設計を考える．設計の目的は実世界の対象をシステム模型へとモデル化することである．そこでまず，システムを3次元の世界で認識し，よりわかりやすい平面の世界で切って設計を行う．そしてその平面が空間内でどのような位置にあるかを認識するために3次元の立場で考え直しながらシステム模型を作り出す．このように，DMC における設計には(1)空間設計，(2)平面設計の2種類があり，互いの視点を補いながら設計を進める．つぎにこれらを説明する．

(1) 空間設計

システムを独立に扱えるサブシステム単位に分割す

表 1 DMC による設計手順
Table 1 Design step on DMC.

手順	作業項目	作業に対する視点および判断基準	結果
①	問題の要旨を短文で捉える．	他の人にシステムのしたいことを説明できる．説明が足りない場合，文中に出てくる単語の説明文を補足する．	自然語の文
②	①の文から「インスタンス」となる「チップ」を抽出し，「単語表」に登録する．	①の判断と同じく人に説明できる構造が作れるかを考える．主に文中の主語および目的語が抽出される．	「単語表」
③	与えられた問題文の中からシステムの行う処理やデータを表す「チップ」を抽出し「単語表」に登録する．	プロセスとしては，問題文中に機能として書かれているものを，データとしてはプロセスの対象となるもの，すなわち，機能を表す文の主語や目的語の中から選択する．	「単語表」
④	②の単語を使って「概念構造図」を作成する．	①の判断と同じく，人に説明できる構造が作れるかを考える．実際には①の文脈から主語と目的語の関係が「概念構造図」における上位概念と下位概念の関係になるように②で抽出した単語を配置する．	「概念構造図」図1参照
⑤	④の単語群を使用目的に従って整理する．	単語をプロセスか，データか，例外処理に使うものかで整理する．	「単語表」
⑥	⑤で整理した単語を各「インスタンス」に割り付ける．	プロセス名はプロセスの主体である「インスタンス」に割り付ける．データ名は同じ名前の「インスタンス」または，「インスタンス」名と同じデータの構成要素である場合はその「インスタンス」に割り付ける．	「インスタンス定義表」
⑦	プロセスやデータ型の詳細な記述を仕様記述言語で行う．	各プロセスの実現を下位の「インスタンス」への機能の分担として行う．データは問題の文脈および機能の実現からデータ構造を考える．	仕様記述言語によるプロセスおよびデータの定義
⑧	バックトラック：⑦において必要となった単語の「単語表」への登録や「インスタンス」への割り付け，「概念構造図」の再構成を行う．後の作業は③～⑦の繰返しである．	登録済み単語の他の「インスタンス」への割り付けや，新規の単語が必要になった場合．また1つの「インスタンス」に付随する単語が多くなりすぎた場合にはこの「インスタンス」で表される概念が細分化されうると考えて新しい「インスタンス」を登場させ，「概念構造図」の再構成を行う．	「単語表」「概念構造図」「インスタンス定義表」「インスタンス」仕様表2参照

る．すなわち，与えられた要求に対して独立性を意識して問題を記述するキーワードを選択し，それらをサブシステム名として取り上げる．このサブシステムと「ボード」が1対1に対応する．これらのキーワードから3次元構造のシステム模型を構成する．すなわち，システム模型は「ボード」の層から構成される．

この時点で要求の中から各サブシステムを説明するための「チップ」を抽出する．この「チップ」が各「ボード」内で具現化されて「インスタンス」となる．1つの「チップ」を複数の「ボード」でそれぞれの場面に適した具現化を行うことがある．「チップ」の概念をこのようにいくつかの側面をもつ「多相チップ」に拡張する．「インスタンス」はシステム模型内で一意であるが，複数の「ボード」に現れる「インスタンス」は「多相チップ」の具現化として共通の＜概念＞を表す．

サブシステムごとの平面設計では，こうした「多相チップ」を使って「概念構造図」において2次元構造を設計する．設計中に追加した「チップ」も「多相チップ」としてシステム全体で共有されそれぞれの「ボード」で具現化される．この作業により各「ボード」の層が「多相チップ」によって縦に配線され，3次元のシステム模型が形成される（**図2**参照）．

（2） 平面設計

「ボード」の設計が平面設計に相当する．ここでの単語使用は「多相チップ」を除いては「ボード」内で閉じているので，他の「ボード」の設計を意識する必要はない．「ボード」内の単語はシステム模型から見た場合サブシステムの識別子付の単語とみなすことによって一意性が保たれる．

設計の基礎的技法としてシステムの階層分割がある．この場合，分割されたモジュールの縦の繋がりはわかるが，横の繋がりが見えにくくなり，同じようなモジュールがあちらこちらに出現することがある．

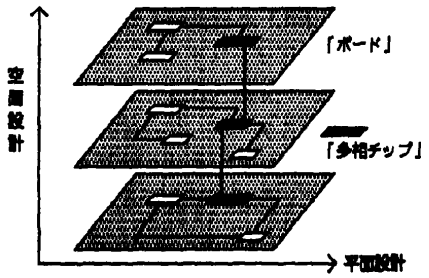


図2 「ボード」の多層化
Fig. 2 The multi-layer "board."

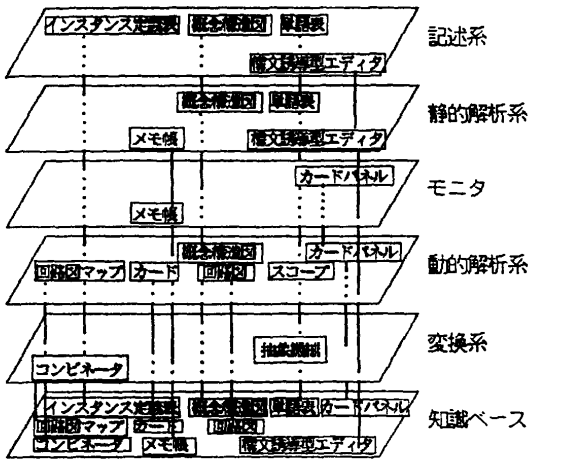


図3 dmCASE のシステム模型
Fig. 3 The system model of dmCASE.

DMC では多層化すなわち，独立した複数の「ボード」の層にシステムを分割し，各「概念構造図」において階層的な構成を行う．そして「多相チップ」により同じようなモジュール（＝＜概念＞）の繋がりを捉える．例えば，**図3**は dmCASE のシステム模型であるが，「単語表」「概念構造図」等がシステム内での共通の＜概念＞として登場している．

3.3 部 品 化

「インスタンス」は「概念構造図」の中で他の「インスタンス」との関連によって元の「チップ」の意味する＜概念＞を設計したものである．そこで，下位の「インスタンス」をパラメータ化することで＜概念＞を表す「部品チップ」が生成できる．例えば，"図書館が本の集りを操作する"ことは，レンタル・ビデオにも応用できる．図書館「インスタンス」において参照している記号を抽象化してアルゴリズムを部品として再利用することができる．ただし「インスタンス」の設計の仕方によっては接続条件の強い「部品チップ」となるだろうが，「部品チップ」を具現化して，再利用することが考えられる．

4. 仕様記述言語

4.1 仕様記述言語としての ML

先に述べたように，ソフトウェア開発工程の要求定義および設計段階における検証を行うためには，形式性の高い仕様記述言語を用いて曖昧さや誤りのない仕様を作成することが必要である．このための仕様記述言語としてつぎのような要素が必要と思われる．

（1） 明確な計算モデルをもつ⇒理解性を高める

（2） 解析・検証・変換可能な形式性をもつ⇒静的な検証および実行コードの生成

（3） 実行可能である⇒プロトタイピング的な動的な検証

（4） 抽象度の高い記述が可能⇒実現と切り離した記述が可能である

このような観点からわれわれは仕様記述言語として関数型言語 ML (Standard ML)[5),6)] を採用することにした．MLは定理証明系を記述するためにEdinburgh大学で開発された言語で，その特徴はつぎのとおりである．

① 高階関数：関数をデータとして扱うことができ，機能の細分化によるコンパクトな記述を可能とする．

② パターンマッチング：関数の引数には変数のパターンが記述できる．

③ 例外処理：例外処理のための式が豊富である．例外識別子を使って複雑な例外処理が行える．

④ 型推論機構：ML は強く型付けされた言語であり，型推論の機構をもつ[8)]．

⑤ 多相型：型変数で表される複数の型をもつ関数を定義できる．入出力が型変数で定義された関数は汎用的な関数として種々の場面で再利用できる．

上記の要素に対応して ML を採用した理由を述べる．

（1） ML は関数としての計算モデルをもつ．関数は2つの側面をもち，1つは写像として集合から集合への対応を明示する．これは「インスタンス」間のメッセージの役割を果す．もう1つは入力から出力を得る計算過程とみなせる．仕様を記述する際によく議論されることに “what” を記述するのか，“how” を記述するのかということがあるが，ここで関数の2面性は前者が “what” を後者が “how” を表していると考えられる．また，仕様を記述するには計算過程に依存しない意味をもつことが仕様の理解性や保守性を高める上で重要であると考える．すなわち，高階関数を使った簡潔な記述は動作の誤りの限定を容易にすることができる．このような性質をもつことから関数は仕様記述言語として適していると言えるだろう．

（2） ML はラムダ理論という理論的基盤をもつのでラムダ計算における様々な結果が利用できる．たとえばカテゴリカル・コンビネータ理論に基づき ML をカテゴリカル・コンビネータ・コードに変換し，それを抽象マシンで実行することができる．

また ML は型付の言語である．一般にプログラミング言語としては型の明記は面倒なものである．しかし，仕様を記述する上で関数の入力や出力をデータ型として規定することはその関数の機能に対するユーザの意図を明確にする上で役立つ．さらに，型推論機構による検証が可能である．

（3） 関数は計算規則であり，式の簡約により実行できる．記号実行も可能であるから，未定義変数を含んだ実行を行える．

（4） 仕様を記述する際には，実システムの環境（マシンの特性，ウィンドウの構成等）に依存しないレベルで記述を行うことが必要である．本質的に何を要求しているかを記述し，自動あるいは対話的に実行プログラムへと変換することが望ましい．また，実際のプログラムは仕様を具体化することでも生成できる．すなわち1つの言語でその記述の抽象度により適応レベルが異なることになる．ML はこのような抽象的な記述が可能であり，変換による実プログラムへの移行も考えられる．

以上のような理由から ML を形式的仕様記述言語とすることにしたが，問題点もいくつか挙げられる．まず，形式的仕様記述言語一般に言われることであるが，言語として難しいとか馴染みにくいということがある．これにはツールの支援による対処が可能であると考える．品質の向上のためには多少の難解さはしかたがないのではないだろうか．

もう1つは ML では自立的に動作する並列プロセスの記述ができないことである．操作的アプローチに基づく仕様記述言語 PAISLey[4)] は関数型の言語であるが，それと共に並列なプロセスの同期を交換関数という互いに交信する関数の記述によって並列プロセスをシミュレートすることができる．並列プロセスをシミュレートするためには言語自体を拡張する必要もある．

4.2 ML の DMC への適合

DMC での設計において形式的仕様記述言語として ML (Standard ML) を使用するために，モジュール機構を付加した[7)]．「インスタンス」の仕様は≪型宣言≫・≪値宣言≫・≪例外宣言≫の記号群と宣言群および下位の「インスタンス」から構成される（表2参照）．各宣言は，それぞれデータ構造・プロセス・エラーハンドリングの定義に相当する．

4.3 ML のカテゴリカル・コンビネータ・コードへの変換

カテゴリカル・コンビネータ理論を用いた関数型言

語処理系の実現が研究されている[9]. Cousineau らはこの理論に基づいた抽象機械 CAM (Categorical Abstract Machine) を提案し, ML (Standard ML ではない) の処理系に適用している. CAM の機能は変更せずに, Standard ML の構文に対応するコンビネータへの拡張を行った.

5. ツールによる支援

dmCASE は, 3, 4 章で述べた方法論と仕様記述言語に基づいて構築されているが, これらの技法や言語の特性をツールとしてどのように支援するかについての考えを述べる.

設計過程は試行錯誤的であるのでこれを巧く支援する必要がある. ツールによる支援の基本的な姿勢は, 人間の思考過程に即したユーザインタフェースを構築し, 思考を妨げない適度な制約を与えることによって, 本質的に設計に関わる仕事だけに従事できる環境を実現することである. そこで特につぎの観点が重要であると考える.

- 検証: 形式性の高い仕様記述言語を使用することによって記述段階における静的な検証が可能である. 方法論や言語の規約を知識として利用する. レベルとしては「概念構造図」によるモデル内の整合性や記述過程の無矛盾性の保証と「インスタンス」を構成する個々の関数の型による検証がある.
- 可視化: 理解性を高めるためには必須の事項である. 可視化の対象としてはシステムのモデル・データ構造・プロセスのアルゴリズムなどがある. これらは仕様の理解性を高めたり, 仕様の動作を理解することを目的とするものである.
- 情報整理: 試行錯誤をしながら設計をするときにいかに情報を整理するかは重要な課題である. わかりやすい情報整理のための枠組みをシステムが供給する必要がある. システムが提供する情報をユーザに選択的に保存させたり, それらの情報の有効性を判断する機構や, ユーザが情報整理のためのメモ (たとえばアルゴリズムメモ) を作成する機能が考えられる.
- テストデータの生成: 動的な検証の一環としてテストがあるが, 設計のレベルに応じたテストを考えなければならない. しかし, テスト技法自体これで十分というものはないので, 現段階では個々の関数に対して型による入力データ値のテンプレートを生成するといったデータ値の制約を導出したり, 言語の型を利用したある程度のテストデータの自動生成が考えられる.
- ガイダンス: 設計段階において発生する様々な情報を効果的に利用すべきである. そこでハイパテキスト機能や対話性を重視したユーザインタフェース, 視覚的効果の高い情報の表現が必要である.
- ナビゲーション: 方法論には種々の規約がある. 規約に従ってユーザがつぎに何ができるか, あるいは何をすべきかということをシステムが誘導することができる. ただし, これが完全に機械に言いなりでは仕事は捗らないだろう. 適度な誘導を考える必要がある.
- モニタリング: 本質的に設計に関わる仕事だけに従事できる環境が望まれるので, システムによる作業の監視を行って必要に応じてユーザに情報を提供する.
- 文書化: システムの要求は始めは自然語等の非形式的な表現である. これを形式的に記述し検証された後

表 2 形式的仕様記述言語による「インスタンス」仕様
Table 2 An example of "Instance" specification.

①	signature	図書館	記号群	① 記号群名	② 継承する「インスタンス」
②	instance			③ 型構成子	④ 値変数の型宣言
③	type	図書館		⑤ 例外識別子	⑥ 記号群の終了
④	val	本の追加	宣言群	⑦ 宣言群名	⑧ 参照する「インスタンス」
⑤	exception			⑨ 型宣言	⑩ 値宣言
⑥	end			⑪ 例外宣言	⑫ 宣言群の終了
				ただし, ③④⑤⑨⑩⑪はそれぞれの列で構成される.	
⑦	module	図書館-Mod			
⑧	instance	本のコピー			
⑨	type	図書館 is 本のコピーのリスト			
⑩	val	本の追加 (本のコピーのリスト, 本) = let val 部数=最大部数(本のコピーのリスト, 本) in 本棚に入れる(図書館, 本のコピーのリスト @部(本, 部数, -, -) end			
⑪	exception				
⑫	end				

には，形式的仕様の意味を正確に反映した，わかりやすい自然語の文書が必要である．これは保守の対象というのではなく，システム理解の補助を目的とする．

●柔軟性：人間の思考はある時はトップダウン的であり，またある時はボトムアップ的であるというように必ずしも系統的に行われるわけではない．特に仕様の記述段階は試行錯誤の過程であるので柔軟な対応が望まれる．

6. dmCASE の概要

6.1 dmCASE における設計プロセス

われわれの推奨するソフトウェア開発のパラダイムから成立する dmCASE における作業プロセスはつぎのとおりである．

（1） システムモデル構築プロセス

始めに与えられる要求は自然語で記述された非形式的仕様である．これから DMC の方法に従ってシステムのモデルを構成し，「インスタンス」の仕様を ML によって記述する．ただし，3.2 節で述べた空間設計については現在はまだサポートしていない．

（2） シミュレーションプロセス

記述された仕様をデータを与えて実行し，動的な検証を行う．結果によっては(1)のプロセスに戻って，モデルの再構成を行う．

（3） 変換プロセス

確認された仕様を効率の良いカテゴリカル・コンビネータに変換して抽象機械上で実行する．なお，2 章で述べた意味での変換については実現していないので，ここでは説明を省略することにする．

6.2 dmCASE の構成

dmCASE のシステム設計を 3.2 節で述べた空間設計と平面設計で説明する．まず空間設計として下記の 6 つのサブシステムを考えた．これは，静的にシステム内における役割分担を考えたものである．つぎに dmCASE の中に現れる＜概念＞を表現する様々な「多相チップ」を抽出する．それらを使って各層の担当者が各自のサブシステム＝「ボード」の平面設計を行った．この結果が図 3 の dmCASE のシステム模型である．図 3 において，各層が「ボード」であり，その中の四角で囲まれた単語が「チップ」を表している．システムを静的に見た場合の各「ボード」の役割はつぎのとおりである．

●記述系[13]：単語表・概念構造図・インスタンス定義表・構文誘導型エディタを使って DMC に従って仕様を記述するためのそれぞれの機能を実現する．

●静的解析系[14]：単語の使用法の検査・概念構造図でのインスタンスの関係とインスタンス仕様の整合性の保証・型推論によるプロセス仕様（値宣言の記述）の検証等，DMC と言語 ML による規約に従った静的な検証や情報の検索を実現する．

●モニタ：記述操作における矛盾の保護や，記述の変化による実行の必要性等，システムモデル構築プロセス・シミュレーションプロセスを監視する．

●動的解析系[14]：カードによる仕様の実行・実行環境の生成・トレース機能・ブレイク機能・実行経路の把握・プロセスアルゴリズムの表示等の仕様の動的解析を実現する．

●変換系：カテゴリカル・コンビネータへの変換と抽象機械での実行を実現する．

●知識ベース：単語・概念構造図・インスタンス仕様・カード等の設計情報の蓄積・管理を行う．

一方，6.1 節で述べた設計プロセスは図 3 のシステム模型におけるシステムの動的な役割である．上記の各「ボード」との関わりはつぎのとおりである．(1) システムモデル構築プロセスにおいては，記述系によって仕様の記述を実現すると同時に静的解析系によって検証を行う．そして記述過程における監視をモニタが行う．(2)シミュレーションプロセスにおいては蓄積された情報を基に，動的解析系による仕様の動的検証や，モニタによる解析情報のモニタリングが行われる．(3)変換プロセスは変換系によるコンビネータへの変換と実行である．各プロセスにおいて情報は蓄積され知識ベースによって管理される．

また，図 3 に登場した dmCASE を構成する「チップ」のうち，ユーザとのインタフェースとなるものには，単語表・概念構造図・インスタンス定義表・構文誘導型エディタ・メモ帳・カード・カードパネル・回路図・スコープ・回路図マップがある．これらの「チップ」が各サブシステム内で具現化され「インスタンス」として種々の工程に登場し，ユーザのインタフェースとなって「ボード」の＜概念＞を形成している．以下に，これらについて説明するが，5 章で述べたツールとしての支援の役割を【 】で示した．**図 4** および**図 5** は在庫管理問題[12]の例であり，以下の説明においてこれを用いる．

（1） 単語表

要求の中から抽出した「チップ」を表す単語を登録し，仕様記述のための情報を整理することを目的とし

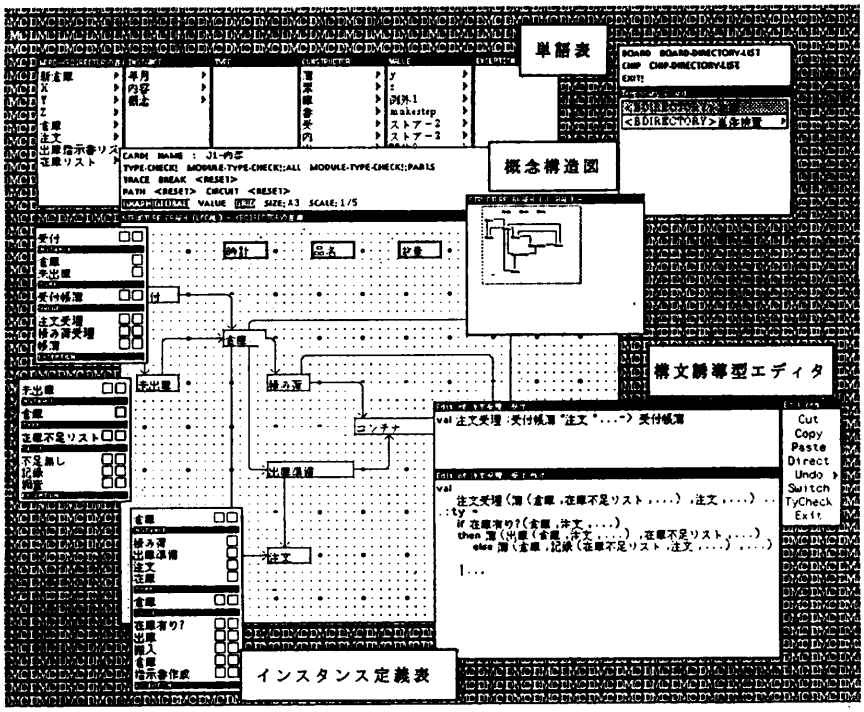


図 4 単語表・概念構造図・インスタンス定義表・構文誘導型エディタ
Fig. 4 The word table・The conceptual structure chart・The instance definition table・The syntax oriented editor.

た表である(図4参照). 単語表はインスタンス・データ型・プロセス等の6つの属性の欄をもつ. 項目ごとに分類することで, 切り出した「チップ」の使用目的を定める. 例えば“倉庫”や“受付”は問題の構造を表すインスタンスとし, “注文受理”は注文を受理する仕事を表すプロセス名とする. 単語を整理することはシステムのモデルを頭の中でシミュレーションする上で役立つ. また, その後の仕様記述においての使用箇所に誤りがある等の矛盾を少なくするための制約を与えている. 作成された仕様内で単語がどのように使われているかの情報も提供する. 単語表の主な機能は単語の入力・移動・複写・削除・検索・名前変更である. 【情報整理】【検証】【柔軟性】

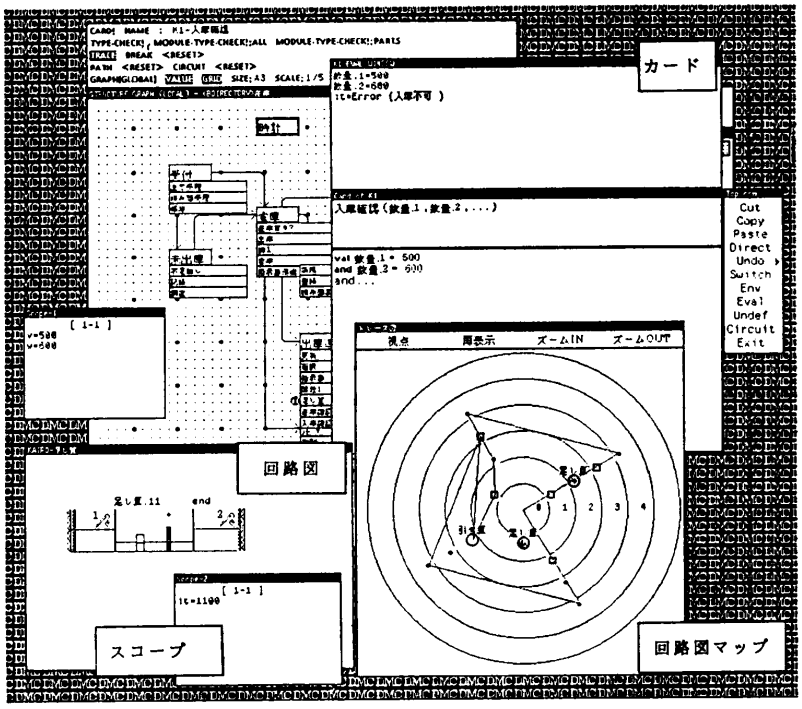


図 5 カード・回路図・スコープ・回路図マップ
Fig. 5 The card・The circuit chart・The scope・The map of circuit chart.

（2） 概念構造図

問題を説明するのに中心的な役割を果す「チップ」によって問題の構造を説明するための「インスタンス」の関連図である（図4参照）．試行錯誤しながらシステムの自然なモデルを作ることを目的とする．図4では“出庫準備”インスタンスは“コンテナ”と“注文”の2つのインスタンスを参照している．概念構造図は仕様を記述する上で問題の全体を把握するための手段である一方，プロセスの流れをこの上で確認したり，データやプロセスの検索を行う等の視点を変えてのモデルの観察を可能とする．概念構造図の主な機能は，作図機能・プロセスパス表示・回路図表示・トレース・ブレイク設定等である．【可視化】【検証】【柔軟性】【ガイダンス】

（3） インスタンス定義表

「インスタンス」仕様を規定する「単語」の登録表である（図4参照）．すなわち，その「インスタンス」で“何”をしたいかを列挙し，そのインスタンスの振る舞いの概要を決定する．図4の“受付”インスタンスは“受付帳簿”いう1つのデータ型と“注文受理”“積み荷受理”“帳簿”という3つのプロセスから構成されている．単語の登録は単語表から選択した単語を行先のインスタンス定義表あるいは概念構造図上のインスタンスの位置を指定することで行われる．単語の移動・複写・削除・エディタの呼び出し・「インスタンス」仕様の一覧表示・仕様の記述度合いの表示の機能をもつ．「インスタンス」仕様の一覧表示は表2の形式の仕様のテキスト表示である．dmCASE では日本語が使えるので単語を巧く使用すれば「インスタンス」の意図を読み取りやすくなる．また，プロセスは下位にあるいくつかのプロセスを参照して形成されている．そこで，このプロセスの流れをテキスト形式で表示するのがプロセス仕様表示機能である．これは概念構造図のプロセスパス表示機能をテキスト化したものである．【情報整理】【文書化】【モニタリング】【柔軟性】

（4） 構文誘導型エディタ

「インスタンス」仕様を ML で記述するためのエディタである（図4参照）．ML の構文要素のテンプレートに対して構文要素メニューや単語表・インスタンス定義表からの単語の選択により記述が行われる．これにより段階的詳細化が可能であり，常に構文の正しさが保証され，構文ミスから開放される．プロセスの呼び出し形式は，そのプロセスの記号群より導出される．例えば，プロセス“注文受理”における“記録（在庫不足リスト，注文）”は“未出庫”インスタンスの“記録”を選択することで導出される．「インスタンス」仕様には記号群と宣言群があるが，プロセスの定義においての記号群はそのプロセスの入力と出力の型の記述である．これは型推論により導出することもできるし，ユーザがこれを記述することは検証およびユーザの意図の確認となる．構文誘導型エディタの主な機能は，編集機能・型推論機能・ハイパテキスト機能等である．また，（1）～（4）は記述過程で重要であり，この間を往き来して設計を進める．そこでこれらの間は操作手順に左右されないようにした．【ナビゲーション】【検証】【ガイダンス】【柔軟性】

（5） メモ帳

試行錯誤的な記述の過程において生じる矛盾を確認し整合性を保つための情報を提示する．整合性に問題が生じた場合に，確認の情報を保存し，ユーザがその情報によって正しい修正が行えるようにすることを目的とする．具体的には単語の削除による他への影響を監視する．他の関数等で参照している関数やデータ型を削除した時，影響のでる箇所をユーザに提供し，削除の確認を行う．その上で削除が行われたならば，メモ帳としてどの箇所の修正が必要であるかの情報を保存する．例えば，図4のプロセス“注文受理”の定義で参照されている“倉庫”インスタンスのプロセス“在庫有り？”を削除すると，“注文受理”の定義を修正しなければならなくなる．このような場合に上記のようにしてメモ帳が生成される．修正が完了すればメモ帳は破棄可能状態になる．また，状況によっては削除単語の復活も可能である．記述の過程を常に監視してメモ情報を更新し，必要に応じてユーザに提示する．【検証】【モニタリング】【情報整理】

（6） カード

記述した仕様を実行する環境である（図5参照）．ML は≪式≫を評価することで実行できる．そこで実行したいプロセス名を指定し，その入力を設定する．図5では，≪式≫入庫確認（数量.1，数量.2）を数量.1＝500，数量.2＝600 で評価している．この入力の設定環境は複数作成でき，様々な環境での実行が可能である．設定環境はテンプレート方式になっている．指定されたプロセスの入力の型あるいは入力のパターンをそのプロセスの定義から導き出して，入力のパターンを生成する．このパターン内に現れたデータ型についてもその定義からパターンを導出する．この

ように実行環境の設定は仕様の定義からの導出によって行われ，定義から可能なテストデータが設定できる．さらに未定義変数を含んだ実行が可能であるので定義の導出の途中で実行ができる．現在はテストデータの型の検出のみであるが，プロセスの定義の解析から値の範囲の制約を与えることも可能である．主な機能は，編集機能（導出・削除・置換）・未定義変数表示・評価・複数環境の生成・回路図呼び出し等である．【検証】【テストデータの生成】

（7） カードパネル

dmCASE ではカードによって各プロセスの実行を行って動的な解析を行う．カード全体を管理する機構がカードパネルである．カードはプロセスが所属するインスタンスごとに管理され，カードの呼び出し・削除・カテゴリカル・コンビネータへの変換とその実行・カード情報の提示の機能をもつ．カード情報には，記述の変更に伴う再実行・再変換の心要性，カードごとの実行状態（入力と出力），実行による動作確認の結果に対するユーザのメモ等がある．【情報整理】【ガイダンス】【モニタリング】

（8） 回路図

記述したプロセス仕様の構造の図式表現である．すなわち，概念構造図のプロセスパス表示機能およびインスタンス定義表のプロセス仕様表示機能の図式化である．ML はラムダ計算の理論に基づく言語であるから，変数の束縛の状態と式の構造を ML の構文に対応する図式パターンによって表現する（図5参照）．プロセスのアルゴリズムの再認識および確認の役割を果すことを目的とし，カードによる実行時には変数束縛の情報と分岐の情報を提示することができる．回路図は［縦線］［横線］［長方形］から構成された［ブロック］の階層構造である．［ブロック］が1つのプロセスを，［縦線］がラムダ束縛の壁を，［長方形］が≪式≫を，［横線］がデータの流れを表現している．≪式≫が構造をもてば下位に展開できる．図5における回路図はプロセス“足し算”の構造を図式化している．【可視化】

（9） スコープ

プロセス仕様の実行時における変数の値を覗く役割を果すもので，回路図から呼び出される（図5参照）．図5ではカードで評価しているプロセス“入庫確認”の実行時において呼び出されたプロセス“足し算”の入力値（上方のスコープ）と出力値（下方のスコープ）を表示している．この値を見て期待する値が束縛されているかを確認する．実行の動作確認を行うために必要な情報はどの地点で誤りが発生しているかを探り出すことである．回路図上でのプロセスの部分再実行や，そのための値設定のインタフェースとして活用したい．【情報整理】

(10) 回路図マップ

プロセス仕様の実行時の状態をプロセス構造のレベルに合せた同心円表示するもので，回路図の情報を集約して構造全体を表現している（図5参照）．回路図マップはプロセス仕様の構造に現れるプロセスを［○］，構造をもつ≪式≫を［□］，定数や変数を［・］としてプロセスの階層構造のレベルを同心円で表したグラフ上にマッピングしたものである．トレースやブレイクを設定したプロセスは［●］［□］で表現される．現在はこのマップ上に実行時の経路を表示し，実行状態を把握することと，トレースやブレイク時にプロセスのマークから回路図を呼び出して変数の状態を観察する媒体としている．図5はカードで実行した“入庫確認”の回路図マップである．“入庫確認”のプロセスにはプロセスとして“足し算”と“引き算”が呼び出され，折れ線で示された経路で実行されたことを示している．また，“足し算”にはトレースが設定され，この［●］を選択すると図5の回路図が開かれ，デバッグの情報が取り出せる．この実行経路を解析してプロセスの命令網羅・分岐網羅等の検出を行い，テストカバレージの定量的な結果を導いてマップ上に表現したいと考えている．【検証】【可視化】

最後に，操作性について一言述べる．記述は試行錯誤によって決定するので統一した操作が思考を妨げないためにも必要である．キーボードとマウスの操作性については，常に両方の操作が混在しているのはやりにくい．また，マウスのみの操作は試行錯誤に向いていると思えるので，単語の入力のみをキーボードから行い，その他の操作は極力マウスによる選択で行うことにした．

7. 記述実験による評価と問題点

われわれの考えるソフトウェア開発環境を現実的に使用するためにどのような問題点があり，さらにどのような要素が必要であるかを明らかにするために在庫管理問題[12]・リフト問題[10]・図書館問題[10]の問題についての記述実験を行った．本実験を通して dmCASE の評価と問題点を考える．本実験はシステムモデル構築プロセスに重点を置いている．記述性・仕様の読み

やすさ・ユーザインタフェースの観点から評価を行う.

(1) 記述性

問題が小規模であるので,1つの「ボード」でモデルを作成できた.問題の中に明記されている機能を実現するためのプロセスを記述することはできたが,リフト問題のようにそのプロセスが互いに同期を取ることを考慮しなくてはならない場合には並列プロセスとしてモデル化を行おうとすると,現在の枠組みでは対処できないことがわかった.リフト問題の場合はプロセスが状態をもって動作する,すなわち状態遷移と捉えることでモデル化ができた.「概念構造図」で表される構造は静的なものであって,自立的に動作する並列プロセスをシミュレートするためには言語自体を拡張するか,イベント系列を記述する枠組みを付加することが必要であると考えている.

(2) 仕様の読みやすさ

形式的仕様は「概念構造図」と「インスタンス」仕様である.「概念構造図」は問題全体のモデルを把握するのに便利である.本実験では被験者は1人であるが,作成者によってかなり異なったモデルが作成されることが予想される.設計途中でシステムのモデルを議論する材料としては「概念構造図」は適当である.すなわち,あまり詳細な資料では議論しにくいが全体を見渡せる「概念構造図」で説明をすれば,理解しやすいだろう.また,「インスタンス」仕様は各「インスタンス」の振る舞いを記述したものであるから,これを読むことでインスタンスの意味を理解できるようにしたい.dmCASE では日本語による記述が可能である.そこで,単語を巧く使用すれば,ML の構文を自然語のパターンに置き換えて文書化も可能であると考える.つまり単語の選択・使用がキーポイントになるので,始めに与えられた非形式的な要求の意図を汲み取るように選択し適切に使用することが望まれる.さらに,単語の名前変更の機能によって後からでもより適切な命名が可能であるので,これによっても理解性を高めることができる.

(3) ユーザインタフェース

記述を支援するための制約やほとんどの操作がマウスのみで可能であることは,人間の思考過程に適していると思われた.直接にMLで記述する前段階としてプロセスのアルゴリズムやデータ構造をMLの枠組み内でのガイド情報を付加したメモ帳に記述して,情報整理を支援する仕組が必要である.そうすることによって ML への理解が容易になるだろう.

8. おわりに

ソフトウェアの生産性・信頼性の向上に対する対策は従来の職人的な方法を単にシステム化するだけでは解決されない.信頼性のあるソフトウェアを開発するためには明確な設計の枠組みをもった新しい環境を構築しなければならない.そのような環境には生産性の向上も期待できると考える.設計方法論・形式的仕様記述言語・ツールによる支援の3つの柱をもつ新しい環境の1つとして dmCASE の提案を行った.「ボード」の多層化による空間設計・記述された形式的仕様から非形式的仕様への文書化,抽象的に記述した ML から実プログラムへの変換についてはまだ,実現していない.7章で述べた問題点を含めて今後の研究課題として dmCASE を発展させたい.

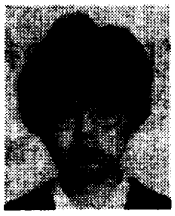
謝辞 本研究の一部は IPA の委託による協同システム開発(株)のソフトウェア環境統合化技術開発計画によるものである.

参考文献

1) Balzer, R., Cheathm, T. and Green, K.: Software Technology in the 1990's: Using a New Paradigm, *IEEE Comput.*, Vol. 16, No. 11, pp. 39-45 (Nov. 1983).
2) Burstall, R. M. and Goguen, J. A.: Putting Theories Together to Make Specifications, *Proc. of 5th IJCAI*, pp. 1045-1058 (Jul. 1977).
3) Futatsugi, K., Goguen, J. A., Jouannaud, J.-P. and Meseguer, J.: Principles of OBJ 2, *Proc. Symposium on Principles of Programming Languages*, ACM, pp. 52-66 (1985).
4) Zave, P.: An Overview of the PAISLey Project-1984, *SIGSOFT SEN*, Vol. 9, No. 4, pp. 12-19 (1984).
5) Wikström, A.: *Functional Programing Using Standard ML*, Prentice-Hall International, Hemel Hempstead (1986).
6) Milner, R.: A Proposal for Standard ML, *Conference Record of 1984 ACM Symposium on LISP and Functional Programming*, ACM, pp. 184-197 (Aug. 1984).
7) MacQueen, D.: Modules for Standard ML, *Conference Record of 1984 ACM Symposium on LISP and Functional Programming*, ACM, pp. 198-207 (Aug. 1984).
8) 横田:型推論と ML, *bit*, Vol. 20, No. 3, pp. 74-83 (1988).
9) Cousineau, G., Curien, P-L. and Mauny, M.: THE CATEGORICAL ABSTRACT

MACHINE, *Lecture Notes in Computer Science 201*, pp. 50-64, Springer-Verlag (1985).
10) Problem Set for the 4th International Workshop on Software Specification and Design, *Proc. of 4th International Workshop on Software Specification and Design*, pp. ix-x (1987).
11) 関根, 大林: 新しいプログラム設計法—概念による設計法 (DMC), *bit*, Vol. 14, No. 7, pp. 102-114 (1982).
12) 大林: 概念による設計法 (DMC) による在庫管理システムの記述, ソフトウェア工学, Vol. 58, No. 5, pp. 33-40 (1988).
13) 松浦, 中里, 大林: 概念による設計法 (DMC) に基づくプログラム開発環境の実現, ソフトウェア工学, Vol. 58, No. 6, pp. 41-48 (1988).
14) 松浦, 大林: プログラム開発環境 dmCASE における解析環境, CASE 環境シンポジウム, pp. 117-124 (1989).

(平成元年8月31日受付)
(平成2年4月17日採録)



松浦佐江子 (正会員)

1955年生. 1979年津田塾大学学芸学部数学科卒業. 1982年同大学院理学研究科数学専攻修士課程修了. 1985年同大学院理学研究科数学専攻博士課程単位取得退学. 同年4月(株)管理工学研究所入社, 研究員. 以来, 人工知能に関する調査研究および, ソフトウェア開発環境の研究開発に従事. ソフトウェア開発環境および設計方法論におけるフォーマルなアプローチに興味を持つ. 人工知能学会会員.

大林 正晴 (正会員)

昭和27年生. 昭和51年東京教育大学理学部応用数理学科卒業. 同年, (株)管理工学研究所入社, 研究員. 抽象データ型, 代数的仕様記述システムなどの研究開発に従事. 現場でのプログラム開発経験をもとに DMC (概念による設計法) を提唱. DMC に基づく支援環境の研究開発に従事. VLSI 用のCAD システムの研究開発, 人工知能の応用にも取り組んでいる. ソフトウェア工学, CASE システム, 特に, 設計方法論, 関数型言語, 協調システム等に興味を持つ.