3S-4

ソフトウェア開発環境ｄｍＣＡＳＥ

－ＦＡＳＥＴ（４）宣言型記述による開発支援ツール－

松浦　佐江子　　大林　正晴　　上枝　泰祐

協同システム開発（株）

１．はじめに

ＦＡＳＥＴプロジェクトでは、形式的仕様記述によるソフトウェア・ライフサイクルにおける新しいソフトウェア開発環境の構築を目指して研究を行なっている。われわれは、オブジェクト指向的な設計方法論である「概念による設計法」と、宣言的言語ＭＬを基礎としたソフトウェア開発支援環境ｄｍＣＡＳＥを提案する。本稿においてはｄｍＣＡＳＥの基礎となる「概念による設計法」の説明と設計手順、および本ツールの構成ならびに特徴を報告する。

２．概念による設計法
(Design Method based on Concepts 略称　DMC)

ユーザの要求が与えられた時、まず、その問題を分析することから始める。与えられた問題を分析していく過程において、問題の性質を表す具体的な＜もの＞や、それらを抽象化した＜概念＞といったものが現れてくる。概念による設計法では、こうした抽象的な＜概念＞あるいはより具体的な＜もの＞をプログラムを設計する際の指針として取り上げる。実際には問題を表現するために必要な種々の単語がこの役割を果す。これらの＜概念＞や＜もの＞を使って問題のモデル化を行なう。すなわち、これらの相互関係を定め、さらに各＜概念＞や＜もの＞のもつ機能や作用の構造を明確にすることによって問題の構造を作り上げていくのである。＜概念＞や＜もの＞とそれらの関係による２次元の図が概念構造図である。概念構造図における関係は、ある＜概念Ａ＞から別の＜概念Ｂ＞へメッセージを送っていることを表している。このメッセージは＜概念Ｂ＞に付随する関数であり、＜概念Ａ＞は＜概念Ｂ＞にメッセージを送り、その結果を参照して自分の仕事を行なうことになる。このようにして作られた構造図と対応した個々の概念の仕様を形式的仕様として書いていく。すなわち、各＜概念＞に付随したデータの型やメッセージの内容を関数型言語ＭＬにより宣言的に記述していくのである。

つぎに、ＤＭＣにおけるプログラムの設計手順を擬人化して説明する。
何か仕事（＝問題）が与えられたとき、その仕事を分担する何人かの担当者（＝＜概念＞≒インスタンスと呼ぶ）を選ぶ。
そして、この中の主任（＝上位＜概念＞）を決め、この主任が自分の仕事を分担すべき部下の担当者（＝参照する＜概念＞）を選ぶ。はじめに登録した担当者だけでは足りない場合には、新たに担当者を登録していく（＝新しい＜概念＞を単語として登録する）。
さらに部下の担当者も自分の仕事を分担する担当者を選択する。各担当者は自分の部下だけを管理して行けばよい訳である（＝直接の下位の＜概念＞だけを用いてその＜概念＞を説明する）。
しかし、自分の直接の部下だけでなく、部下の部下の仕事も参照したいという事もあるだろう。この場合には、部下の担当者を選ぶときの関係を”参照”でなく”継承”という関係にしておけば良い。このように、部下を採用するときにどういう関係で採用するかを規定することで管理する部下の範囲を決めることができるのである。
このようにして、仕事の分担と部下の採用を行なって、この仕事を行なうための組織図（＝概念構造図）を作成して行く。

The Software Development System dmCASE
:Declarative Specification
Saeko Matuura,Masaharu Ohbayashi

各担当者が行なう仕事の内容を箇条書にして表わしたものが、本ツールにおけるインスタンス定義表である。そして具体的な仕事の内容（＝関数）や扱うデータの型（＝タイプ）の定義を構文誘導型のエディタで行なう。
上記の手順を繰返しながら設計を進めていくのが、概念による設計法（ＤＭＣ）である。

３．ｄｍＣＡＳＥの特徴

われわれの研究の主目的は、つぎのとおりである。第一には、与えられた問題に対して仕様を形式的に記述することである。第二には、その形式的仕様記述によって、ソフトウェア開発の早期過程において仕様の検証を行なうことである。また、形式的仕様を実行可能なプログラムへ変換することも試みた。
ｄｍＣＡＳＥではこれらをつぎのように実現している。まず、設計方法論ＤＭＣにより問題の分析から仕様の記述のプロセスを支援し、関数型言語ＭＬにより副作用のない簡潔な仕様を記述することで形式的仕様を構築する。そして、ＤＭＣの制約や関数型言語ＭＬの型によって形式的な検証を行なう。また、ＭＬがラムダ計算の理論に基づいていることから仕様記述をコンビネータに変換することで実行効率の向上を目指した。
上記に加え、このようなソフトウェア開発環境としてのツールに欠くことのできない要素として、視覚的ユーザ・インタフェースと一貫した操作性を重視して開発を行なった。
仕様の検証についてもう少し述べよう。仕様の検証としては静的解析による検証と動的解析による検証がある。静的解析は仕様を記述する段階においてのＤＭＣによる制約およびＭＬの型推論機構による検証である。また、動的解析とは、この形式的仕様が実行可能仕様であることである。すなわち、仕様のインタプリタによる実行によって、シミュレーションを行なうことができる。

４．ｄｍＣＡＳＥの構成

ｄｍＣＡＳＥでは主に仕様の記述・静的解析・動的解析・変換の工程を支援している。
ｄｍＣＡＳＥを構成する＜概念＞の内、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・回路図・スコープ・回路図マップがある。これらの＜概念＞が種々の工程に登場し、ユーザのインタフェースとなる。各＜概念＞の概要と特徴を表４－１に、＜概念＞と設計工程との関連を図４－１に示す。

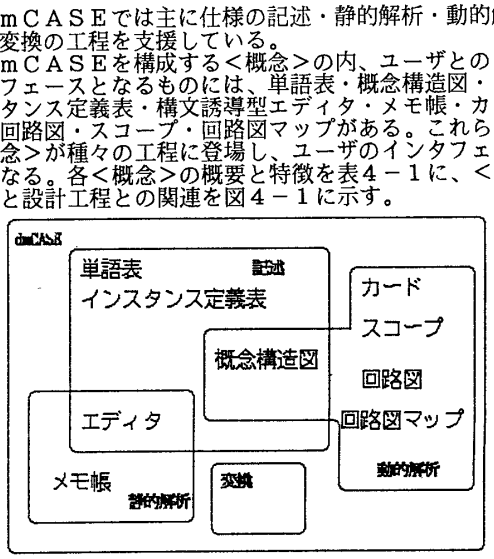

図４－１　ユーザ・インタフェース

表４－１　ｄｍＣＡＳＥの＜概念＞

＜概念＞	概要および特徴
単語表	要求または問題の中から抽出した具体的な＜もの＞あるいは抽象化した＜概念＞の名前を整理した表である．単語の使用目的に合わせて分類することによってその後の仕様記述においての矛盾を少なくする．単語の検索機能をもつ．
概念構造図	問題を説明するのに中心的な役割を果すインスタンス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．インスタンスの関連によって記述を規定する．関連には参照と継承の２種類がある．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．
インスタンス定義表	インスタンスの振る舞いを規定する単語の登録表である．単語を登録することによってそのインスタンスの振る舞いの概要を決定する．
構文誘導型エディタ	インスタンスの振る舞いをＭＬで記述するためのエディタである．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をすすめるので、段階的詳細化が可能である．型推論により個々の関数の検証を行なうことができる．
メモ帳	記述の過程において生じる矛盾を確認するための情報を提示する．ＤＭＣにそった記述は試行錯誤の過程であるので、記述の整合性が保たれることは重要である．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なえるようにすることを目的とする．
カード	記述した仕様を実行する環境である．複数の実行環境を設定できる．さらに未定義変数を含んだ実行が可能である．
回路図	記述した仕様の構造の図式表現である．ＭＬはラムダ計算の理論に基づくので、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．構造の再認識および確認の役割を果す．
スコープ	仕様の実行時における変数の値を覗く役割を果す．回路図から呼び出される．
回路図マップ	仕様の実行時の状態を表示する．回路図の情報を集約して構造全体を表現している．実行時の経路を表示すると共にトレースやブレイク時の変数の状態を観察する媒体となる．

４．１　記述工程

上記の＜概念＞の内、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介してＤＭＣの設計手順に従って記述を進める。この段階は試行錯誤的であるので、思考を妨げないようにほとんどの操作をマウスのみで行なうようにした。記述工程における仕様の整合性を保つために、記述作業を監視する機構を持つ。メモ帳がこの役割を果す。各＜概念＞間の制約の確認や、型推論機構による静的解析を行なって仕様を検証し、ＭＬによる形式的仕様を確立する。

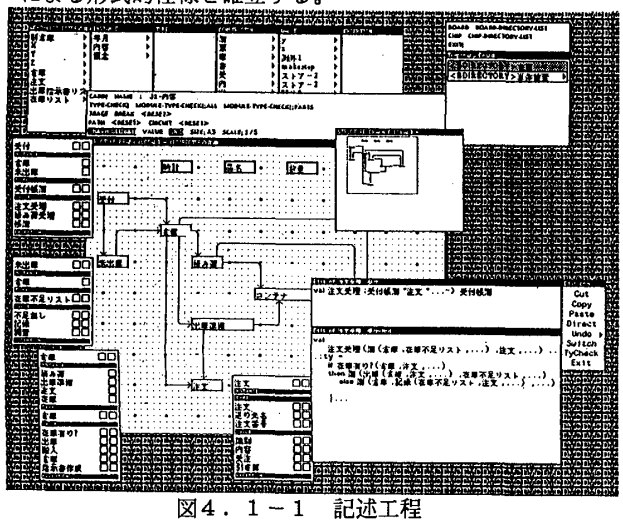

図４．１－１　記述工程

４．２　解析工程

記述工程で作成された仕様の実行による検証を行なうことを目的とする。上記の＜概念＞の内、カード・回路図・スコープ・回路図マップを介して実行およびデバッグ環境を支援する。概念構造図をデータ構造や関数の経路といった様々な切り口で捉えて構造の確認を行なう。インタプリタによる未定義変数をも含んだ実行を行ない、その時の変数の状態を回路図で詳細に観察したり、回路図マップによって実行状態を全体的に把握する。

４．３　変換工程

変換工程は検証の終了した仕様をカテゴリカル・コンビネータ理論に基づき変数のないコードに変換し、実行する過程である。仕様記述言語であるＭＬはラムダ計算の枠組みの中で考えられるので、ラムダ式をコンビネータに置き換える理論に基づきＭＬを変換する試みがなされている。そこで、本ツールではこれを拡張して環境に取り入れた。この変換したコードを実行する抽象機械をツール上に実現し、このコードをインタプリトすることで完成した仕様の実行効率の向上とコードの移植性が確かめられた。

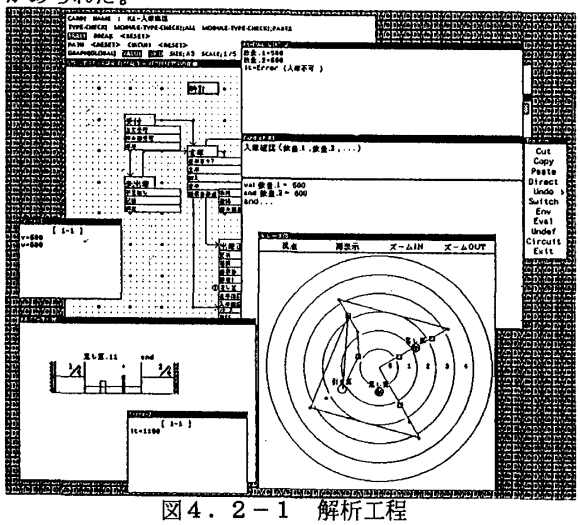

図４．２－１　解析工程

５．まとめ

形式的仕様記述には検証の手段を与えるという利点に反して、記述言語の難解さという問題がある。ＭＬは検証の意味では役立つが理解性ではまだ問題がある。そこで、ＭＬによる記述に対する一層の支援が必要であると考える。

参考文献

[1]R.Millner"A proposal for standard ML"Conference Record of 1984 ACM Symposium on LISP and Functional Programming, ACM Aug. 1984, pp 184-197
[2]D.MacQueen"Modules for standard ML" Conference Record of 1984 ACM Symposium on LISP and Functional Programming, ACM Aug. 1984, pp 198-207
[3]横田，型推論とＭＬ，bit,Vol.20,No.3,pp74-83
[4]G.Cousineau,P-L.Curien,M.Mauny,"THE CATEGORICAL ABSTRUCT MACHINE" Lecture Notes in Computer Science 201,Springer-Verlag 1985
[5]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 1982
[6]大林，”標準ＭＬを用いたプログラム設計支援環境”，ソフトウェア工学 51-5 1986
[7]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 58-6 1988
[8]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 1989