解　説

3. プログラム設計環境のツール

3.3 プロトタイピング支援ツール†

伊藤　潔††　本位田真一†††

1. プロトタイピングサイクルの認識

ソフトウェアの開発を効率的に進めるために，1970年代の前半からソフトウェア生産技術の研究が盛んになり，開発を順序立てて後戻りなく進めるためにソフトウェアのライフサイクルの概念を導入した（たとえば文献[TEI 77],[ROS 77]）．ライフサイクル内の要求分析は，ソフトウェアのユーザや発注者の意図を漏れなく正確に把握し，それ以降の開発フェーズに誤りやあいまいさを伝えないようにするフェーズであり，設計は，要求分析の段階で明確となったユーザや発注者の要求仕様を逸脱しない範囲で，ソフトウェアの内部の実現方式や実行方式などをプログラミング以前に確定するフェーズである．

このライフサイクルに基づくソフトウェア生産技術の問題点は，プログラミング以前の要求分析や設計のフェーズにおけるソフトウェアの記述が実行可能ではない仕様文書であるため，プログラミングが行われるまでソフトウェアの動きがみえない点である．この実行可能ではない仕様文書を人が読んで要求分析や設計の正しさを確認するレビュには，一般に膨大な時間や手数がかかる．

仕様文書を作成しこれを確認するだけではなく，要求分析や設計のフェーズでも開発中のソフトウェアの動きを人にみせて開発をより効率的に進めようとするアプローチがプロトタイピング手法である（たとえば文献[ACM 82],[ARI 86],[ITO 87],[KAT 88]）．

プロトタイピング手法は，対象となるソフトウェアの開発のために，そのソフトウェアのプロトタイプを作成し，対象のソフトウェアの（模擬）実行環境の上でプロトタイプを実行させながらその振舞いを人に見せて確認し，かつ，そのプロトタイプを迅速にかつ徐々に進化，精密化しながら，ユーザや顧客の要求や設計者の実現方法を明確化する開発手法である．

ソフトウェアのプロトタイピングがもつべき性質は次の5点であると考える．プロトタイピング支援ツールはこの5点を支援しなければならない．

（1） 実行性 (executability)

プロトタイプは，その内部での実現方式（あるいは実行方式）は対象ソフトウェアの実現方式（実行方式）と異なっていてもよいが，最初から実行可能であること，その実行の際に対象ソフトウェアの機能や性能を調べることができること．

（2） 環境導入性 (target environment introducibility)

プロトタイプは，対象ソフトウェアの実行する実環境，あるいはその模擬環境上で実行可能であること．

（3） 構築や変更の迅速性 (rapid constructability/modifiability)

プロトタイプには，対象システムのもつべき機能項目のうち，着目した項目の機能を迅速に（短い工期で）かつ安価に（少ない工数で）装備可能であること．また対象システムに対する仕様の変更，誤りあるいは不明確な点に簡単に対応できて変更可能であること．

（4） 進化性 (step-wise refinability)

プロトタイプは，全面的な版の改変を行うのではなく，徐々に精練，精密化して版を進化可能であること．最終版のプロトタイプは，対象ソフトウェアが実行の効率を問わないものであれば，実際の対象システムとする場合もあること．（プロトタイプを要求分析/設計用にのみ用いるのか，あるいは最終のソフトウェアとしても用いるのかについては，対象ソフトウェアやプロトタイピングの目的に依存する．）

（5） 習得の容易性 (easy learnability)

プロトタイプの記述が容易であり，その記述方法の

† Tools for Prototyping for Developing Software by Kiyoshi ITOH (Faculty of Science and Technology, Sophia University) and Shinichi HONIDEN (Systems and Software Engineering Laboratory, Toshiba Corporation).

†† 上智大学理工学部・一般科学研究室情報科学部門
††† (株)東芝・システムソフトウェア技術研究所

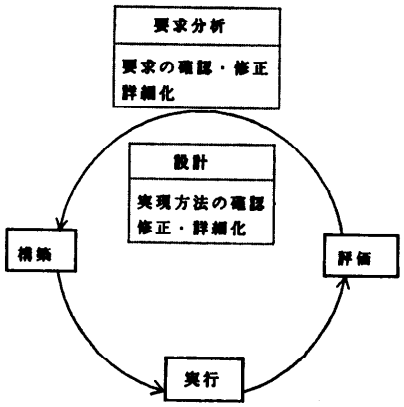

図-1 ソフトウェアプロトタイピングサイクル

習得が容易であること.

プロトタイピングの位置づけには，従来のライフサイクルモデルに対峙させる場合と，要求分析/設計段階での従来の開発手段の代替あるいは増強手段とする場合とがある.

このプロトタイピングの位置づけについての議論も重要である．しかし，最終のプログラミング時にプログラミング・テスト実行・評価というサイクルが存在することと同様に，プロトタイピングにもプロトタイプの構築・実行・評価の３フェーズからなるサイクルが存在する．このサイクルの各フェーズに効果的なツール群を装備することが，プロトタイピング手法をより有効なソフトウェア開発の手法とすることができると考える.

プロトタイピング手法を導入したソフトウェアの開発過程を詳しく示すために，図-1 のプロトタイピングサイクル[TAM 86] [TAM 87] [ITO 87] をあげる．これはプロトタイプの構築・実行・評価の三つのフェーズからなるサイクルを繰り返して回りながら要求分析や設計を進めることを示す．プログラミングではこのような（テスト）実行をともなうサイクルは従来から存在するが，要求分析や設計ではプロトタイピングが注目されるようになって初めて導入された.

プロトタイプの実行フェーズでは，対象のソフトウェアに対するプロトタイプが実行され，計算機による各種の出力が得られる．プロトタイプの評価では，この出力をみて，要求分析の場合には，要求の確認，抽象的な要求の具体化，あいまいな要求の明確化，あるいは要求の修正を図ろうとする．また，設計の場合には，前段で明確にされたユーザの仕様を逸脱せずに，ソフトウェアの内部で採用する実現方法の確認・具体化・修正を図ろうとする.

対象のソフトウェアの規模があまり大きくない場合には，ソフトウェアのライフサイクルに沿って厳格に開発を進めると，過剰な工程管理というオーバヘッドをもたらしてしまうことがある．この場合には，プロトタイピング手法の導入がより有効となる．このプロトタイピングでは，図-1 の要求分析と設計の二つのサイクルを区別せずに任意に適切に切り換えながら回る.

2. プロトタイピングサイクルの必要事項

構築したプロトタイプがユーザや開発者の要求事項を満たすまで，プロトタイピングサイクルを繰り返す．プロトタイピングの重要な性質である迅速性を満たすためには，このプロトタイピングサイクルにかかる時間を最小化すること，すなわち，個々のフェーズにおいて可能なかぎり迅速性を満たすことと，プロトタイピングサイクルを回る回数を可能なかぎり減らすことが必要である.

実際には，すべてのフェーズを効率良く行うことは困難である場合が多い．このような時間のかかるフェーズは，他のフェーズで補うようにすることがより実際的な解決法である.

プロトタイプ評価にコストと時間のかかる項目は，プロトタイプ構築に多少負荷がかかったとしても，十分に行って，後の評価フェーズに対する負荷を減らすことにより，プロトタイピングサイクル全体の迅速性を保つことができる．たとえば，並行プロセスの同期部分について，プロトタイプ評価フェーズにおいてデッドロックを検出すると時間がかかる．この場合には，プロトタイプ構築フェーズにおいて定理証明手法などで，多少負荷をかけても，正しさの保証された同期部分をもつプロトタイプを生成するほうが，全体としてのコストダウンにつながる[HON 87].

このように，一般に簡単にプロトタイプを構築することを重視することがプロトタイピングであると捉えられているが，プロトタイピングサイクルにかかる時間の最小化という観点からは，必ずしも正しくない.

2.1 プロトタイプ構築フェーズ

仕様記述言語や部品検索ツールなどの種々の手法が，プロトタイプを効率良く構築するために有効である．このフェーズにおける必要事項は，それらの中のどの手法を採用するかに大きく依存する．たとえば，仕様記述言語ならば，おのずとモデリングの容易性が

必要とされる．また，プロトタイプを修正して再構築することの容易性も必要とされる．

仕様記述言語には，これまで要求分析や設計に用いられていたが実行を意識していなかった言語に，動的な意味を付加した言語—これは，実行可能な仕様記述言語 (executable specification language) と呼ばれる—，関数型・論理型・オブジェクト指向などの非手続き型の言語，豊富なデータ型や演算子をもつ手続き型言語，ペトリネットモデル，状態遷移モデル，データフローモデルなど，さまざまある．

実行可能な仕様記述言語とこの言語で書かれたプロトタイプの実行手法については，文献[SAE 87]に詳しく述べられている．

実際の複雑なシステムは，多くのソフトウェアモジュールから構成される．このような大規模なソフトウェアシステムを対象とするプロトタイピング手法の生産性を高めるためには，ソフトウェア再利用手法 (たとえば，文献[BOA 84],[KOM 87]) が有効な手法の一つとして考えられる．

ソフトウェア再利用手法は，プロトタイピングの観点（限りなく低コスト性，迅速性を追求する観点）からは，次の3点を検討する必要がある．

(1) あらかじめどのような部品を用意しておくかについて検討しなければならない．すべての対象ソフトウェアを包含することは，一般的に容易ではなく，対象ソフトウェアの分野をある程度限定して，その範囲でできるだけ汎用的な部品を用意することが必要である．

(2) 部品の検索仕様の記述方法と，所望の部品を検索するために必要な検索仕様の記述量を検討しなければならない．部品の検索仕様の記述の方法としては，論理式，格文法，キーワードなど，種々のものが提案されている．いずれの手法を用いても，部品を検索するためには一定量の検索仕様の記述が必要であり，また一つの検索仕様の記述では一つの部品しか検索できないことが普通である．すなわち，所望の部品数が増えると，それにともない，仕様記述量が増加する．このため，プロトタイピングのためにソフトウェア再利用手法を用いる場合には，できるだけ検索仕様の記述量を少なくすることの検討が必要である．

(3) 部品ライブラリがよく用いられているが，そのライブラリに関してあまり知識のない人にとっては，ライブラリによっては必ずしもプロトタイプ構築の迅速性が実現されているとは限らない．このような場合には，たとえば，部品ライブラリに関する知識を知識ベース化することにより，そのライブラリの専門家以外のユーザにとっても迅速な部品の再利用が可能となるような方策が必要である．

仕様記述言語を用いるにしろ，部品の再利用を行うにしろ，プロトタイプの記述手法の構成要素として，次の項目が必要になると考えられる．

(1)′ 効果的なマンマシンインタラクションを支援するインテリジェントなエディタ（視覚に訴えやすい図形的シンボル，いわゆるアイコンや，複数の表示ができるマルチウィンドウなどの機能をもつ）．

(2)′ プロトタイプの進化や変更の軌跡を記録し，必要に応じて必要な版のプロトタイプを検索し再利用できるための，プロトタイプ版管理システム．

2.2 プロトタイプ実行フェーズ

プロトタイピングの性質の中でもっとも重要なものは，実行性である．この性質は次の必要事項を満たすことを意味する．

(a) 迅速に実行できること
(b) 不完全な（いいかげんな）設定でも動くこと
(c) 実行状態を視認できること
(d) 適当なところで実行を中断，再開できること
(e) 評価のしやすい形で実行結果を表示できること

プロトタイプの実行手法の構成要素として，次の項目が必要である．

- シミュレータ的なものやインタプリタ的なものによって，プロトタイプの記述のソースのレベルで実行させる処理系，
- コンパイラやリンカをとおして実行可能プログラムに変換して実行させるもの，あるいは
- 宣言的な記述がされている場合には，推論エンジンなどの実行系．

2.3 プロトタイプ評価フェーズ

一般に，プロトタイピングといえば，プロトタイプの構築と実行という二つのフェーズが重要視され，この評価フェーズに関する議論はきわめて少ない．しかし，前の二つのフェーズが迅速に処理が進んでも，この評価フェーズで時間やコストがかかったり，意味のない評価をした場合には，無意味なプロトタイピングサイクルを繰り返すことになり，プロトタイピングの迅速性を損なうことになる．

事務処理ソフトウェアなどでは，そのプロトタイプを実行させて振舞いを視認するだけで，比較的容易に

その正しさを評価できる．しかし，より複雑なソフトウェアの場合には，振舞いの単純な視認だけでは不十分である．プロトタイピングの迅速性を失わないように評価フェーズに効率的なツール群を整備して，プロトタイプの振舞いや実行結果を詳細に調べる必要がある．

プロトタイプの評価のためには，プロトタイプが機能面の事項を満たすことを調べる評価系と，性能面の事項を満たすことを調べる評価系が必要である．

プロトタイプの機能面の評価系として，プロトタイピングを行う開発者がプロトタイプの実行を会話的に制御できるオンラインモニタや，実行の過程を必要に応じて逐一たどることを可能とするビジュアルなトレーサなどが考えられる．また，たとえば，デッドロックの検出なども含まれる．

プロトタイプの性能面の評価系として，グラフや図表を表示しながら，設計者の分析の意図に沿う出力を行う出力エディタなどが考えられる．また，たとえば，ボトルネックの検出なども含まれる．

プロトタイプの実行はたとえ効率化したとしても，コストと時間のかかる作業である．したがって，いかにプロトタイプの評価作業を計算機で支援できるかが，無駄なプロトタイプの実行を繰り返すことを防ぐことになる．そのためには，このプロトタイプ評価フェーズにおいて「迅速なバグの検出と適切な改善プランを生成」することが重要である．

また，プロトタイプの実行はシミュレーションにすぎないため，すべての場合をつくすことはきわめて困難である．そこで，ある実行結果例から，あるいはそのいくつかの組合せから，他の結果をある程度予測する機能も必要とされる．これは，動的な実行結果を静的に吟味する機能によっても実現することができる．

3. プロトタイピング事例

最近のプロトタイピング事例を概観する．これ以前の事例については文献[ITO 87]に収めている．

2. でプロトタイピングのための構築・実行・評価のツールの必要事項を整理した．以下の事例はこの必要事項をすべて満足しているわけではない．特に，評価ツールが必要であるとの認識はまだ一般的でないため，評価ツールの項目としては空白のものが多い．表-1 に示すとおり，プロトタイプの評価手法には，すべて人による視認が入る．視認以外の評価手法がある場合に限り，評価手法の項目として特記している．プロトタイピングが，今後より複雑なソフトウェアを対象とするようになると，この評価項目に記入される事例が増えると考えられる．

以下の事例では，表-1 に従い，プロトタイピングの対象ごとに順に概観する．

Wasserman, A. I. は，対話型システムの要求分析を支援する自動化ツール USE (User Software Engineering) を提案している[WAS 82,85,86]．USE では，グラフィックスを用いて状態遷移モデルの形式，あるいはテキスト形式でプロトタイプを記述し，これを関係データベース管理システムが管理可能な関係式に変換した後，TDI (Transition Diagram Interpreter) と呼ぶインタプリタで実行させる．

Boar, B. H. は，対話型システムを対象として宣言的な仕様化に基づくプロトタイピングを提案している[BOA 84]．プロトタイプは部品パッケージとデータディクショナリなどによって実行される．

Stelovsky, J. は，対話型システムの仕様記述，プロトタイピング，実際の使用を統合する XS-2 を提案した[STE 88]．XS-2 では，対話型システムで使用されるコマンドの文法を定義する機能，コマンド間の関係をコマンドツリーとして定義する機能をもつ．コマンドツリーは Modula-2 のモジュールに自動的に変換され，コンパイル・リンクを経て実行可能なプログラムになる．

Mason, R. E. A. は，対話型システムのスクリーンをベースにした対話型システムの設計開発を援助するシステム ACT/1 を提案した[MAS 82,83]．ここでは，対話のシナリオを記述すると，自動的にそのシナリオに従って実行するプロトタイプが生成され，表示項目・常時順序，対話の実際などがユーザに提示される．プロトタイプを精練すると最終的に動くソフトウェアになる．

Lewis, T. G. は，ユーザインタフェースプログラムを対象として OSU (Oregon Speedcode Universe) と呼ぶ UIMS (User Interface Management System) を提案している[LEW 89]．

Urban, S. D. は，仕様記述言語として Descartes 言語とそのインタプリタを開発し，情報蓄積検索システムの機能面のプロトタイピングに適用した[URB 85]．Descartes によるプロトタイピングでは，モジュールの入出力だけの記述にデータを与えて実行させる抽象実行 (abstract execution) と，モジュールの実体を

表-1 プロトタイピング事例
評価手法の項目にはすべて，「視認」が入る．視認以外の評価手法がある場合に限り，それを特記する．

作成者	対象	ツール名	構築手法	実行手法	評価手法
Wasserman	対話型システム	USE	状態遷移モデル	インタプリタ	
Boar	対話型システム		宣言的な仕様	データディクショナリ＋部品パッケージ	
Stelovsky	対話型システム	XS-2	コマンドツリー	Modula 2 へのトランスレータ	
Mason	対話型システム	ACT/1	シナリオ	部品	グラフィカル出力
Lewis	ユーザインタフェース	OSU	アイコン	ジェネレータ	
Urban	情報蓄積検索システム	Descartes	モジュール入出力	抽象実行	
Belkhouse	図書データベース		抽象データ型言語	トランスレータ	
Herndon	言語トランスレータ	Kodiyak	属性文法	トランスレータ	
Ceri	Ada コンパイラ・インタプリタ	PCDB	関係モデル	プログラム構築データベース	
Wang	電話システム	SXL	状態遷移モデル	インタプリタ	
Tsai	オフィス間電話システム	FRORL	フレーム・ルール	Prolog へのトランスレータ	
Jard	通信プロトコル	Veda	ISO標準プロトコル言語	シミュレータ	
Balzer	パケット経路決定ソフトウェア	Gist	状態遷移モデル	マッピング・記号実行	
本位田・松本	実時間システム	QPIT	状態遷移モデル	Prolog へのトランスレータ	
Luqi	実時間システム	PSDL	Syntax 指向エディタ	トランスレータ・Ada ソフトウェア部品・スケジューラ	
Diaz-Gonzales	実時間システム	ENVISAGER	オブジェクト指向言語＋ITL	シミュレータ	
Duncan	組込み型実時間制御ソフトウェア		Ada	コンパイラ＋部品	
Terwilliger	組込み型実時間制御ソフトウェア	PK/c++	オブジェクト指向＋論理型言語	c++ のプリプロセッサ	
Bruno	プロセス制御ソフトウェア	PROT	拡張ペトリネット	シミュレータ	アニメーション
Zave	プロセス制御ソフトウェア・患者監視システム	PAISLey	関数型言語	インタプリタ	
伊藤・田村	並行処理ソフトウェア	P-Flots	メッセージフロー	シミュレータ	デッドロック検出ボトルネック改善
本位田・内平	並行処理ソフトウェア	MENDEL	並列型オブジェクト指向言語	Prolog インタプリタ	
Hooper	分散型データ処理	ADS	グラフィカルユーザインタフェース	VMS 環境＋模擬環境	
Witten	分散型ソフトウェア	Jade	並行プロセス	シミュレータ	
Francesco	分散型ソフトウェア	ESL	拡張 CSP	インタプリタ	対話型デバッグシステム

徐々に付加しながらの実行を可能とする．

Belkhouse, B. は，大学の図書データベースを例として，抽象データ型言語を用いたプロトタイピング手法を提案している[BEL 86]．抽象データ型言語で書かれた仕様は，プログラム合成トランスレータにより実行可能なプログラムに自動変換される．

Herndon, Jr. R. M. は，属性文法に基づいて定義されたプログラミング言語のトランスレータを作成する kodiyak を提案した[HER 88]．kodiyak をプログラミング言語のプロトタイピングツールであると位置づけている．

Ceri, S. は，PCDB と呼ぶプログラム構築データベースを用いて関係モデルによってプロトタイピングを行っている[CER 88]．対象は，Ada のコンパイラ・インタプリタとマルチマイクロプロセッサのプログラムである．

Wang, Y. は，SXL と呼ぶ状態遷移記述言語で電話システムのプロトタイピングを行った[WAN 88]．

Tsai, J. J. P. は，FROL と呼ぶフレーム・ルールに基づく言語によってオフィス内電話システムのプロトタイプを記述し，Prolog へ変換している[TSA 88]．

Jard, C. は，通信プロトコルを対象として，ISO 標準プロトコル言語によるプロトタイピングを提案している[JAR 88]．プロトタイプは Veda と呼ぶシミュレータによって実行する．

Balzer, R. M. は，仕様記述言語として状態遷移モデルに基づく Gist を提案している[BAL 82]．Gist による仕様を実行させるための方法として，マッピングと記号実行の二つを用いる．Gist の各構文は，マッピングルールにより実行可能な文に変換されるため，Gist で書かれた仕様はプロトタイプになる．このプロトタイプを記号実行によって実行させる．例としてパケットの経路を決定するプログラムをあげている．この基本的な考え方は，Transformational Implementation という方法論の下に，抽象的プログラム仕様から，ルールを用いて実行可能なプログラムを生成する方法（たとえば[BAL 85]）に基づいている．

本位田・松本は，実時間システム開発向きプロトタイピング手法 QPIT (Qualified Performance Integrated Tool) を提案している[HON 85a]．要求分析レベルで，SRL (State transition based Requirement Language) と呼ぶ，状態遷移モデルにおける条件の定義，現在の状態と成立条件に対する次状態の定義，各状態の出力を定義する言語を用意し，SRL で書かれた要求記述を Prolog 記述に変換した後，実行させる．

Luqi は，実時間システムのソフトウェアのプロトタイピングのために PSDL (Prototyping System Description Language) を提案した[LUQ 88ab]．PSDL は実時間システムに必須のタイミング制約を記述でき，Ada で書かれたソフトウェア部品を取り込むことができる．プロトタイプの構築には PSDL 向きの Syntax 指向エディタが用意されている．

Diaz-Gonzales, J. P. は，実時間システムを対象として ENVISAGER と呼ぶオブジェクト指向言語と ITL（区間時制論理）を組合せた環境を提案している[DIA 89]．ITL は時間的制約を記述するために用いられており，シミュレーションによって実行している．

Duncan, A. G. は，小規模な組込み型実時間制御システムの設計を，開発用計算機上で Ada を用いたプロトタイピングによって行った[DUN 82]．同じ分野であるが少しずつ異なる，開発対象のいくつかのシステムの機能上の違いを整理し，それを汎用的に吸収できる Ada によるパッケージを開発用計算機上に用意している．詳細化を施すプロセスは，まず，Ada のプログラム仕様部のみを記述し，次にプログラム本体部をラフに記述し，徐々にその本体部をきちんと記述していくプロセスである．

Terwilliger, R. B. は，組込み型実時間ソフトウェアを対象として，PK/c++ と呼ぶオブジェクト指向と論理型言語をベースとした実行可能仕様記述言語を提案している．PK/c++− は c++ のプリプロセッサとして位置付けられる[TER 89]．

Bruno, G. は，拡張したペトリネット PROT を用いてプロセス制御システムのプロトタイピングを行った[BRU 85,86]．「並行プロセス」と「同期」の概念に基づくオペレーショナル仕様記述，ペトリネットのデッドロックや有界性の検出などの正当性の検査，解析的あるいはシミュレーションによる評価，簡単なアニメーションを導入したプロセスの並行的実行の表示，Ada で記述されたスケルトンプログラムの生成などの特徴をもつ．

Zave. P. は PAISLey (Process-oriented, Applicative, Interpretable Specification Language) と呼ぶ関数型言語で，プロセス制御システムや病院の患者監視システムの要求仕様の定義を行い，システムの実行可能なモデルを記述した[ZAV 81,82,84,86]．これをオペレーショナルなアプローチと呼んでいる．プロセスを関数で仕様化し，また，プロセス間通信も関数で仕様化する．この仕様はインタプリタによって実行される．仕様の矛盾は自動的に検出され，また仕様が不完全でも実行できる．

伊藤・田村は，多数の相互に関連する並行処理モジュールが多数のトランザクションを処理する並行処理ソフトウェアのプロトタイピングを進める方法 SPM (Stepwise Prototyping Method) を考案し，この SPM に基づくプロトタイプの構築および実行ツール群 P-Flots (Prolog based FLOW and Task Simulator) を開発した[ITO 84a,b,89],[TAM 86,87]．SPM では，機能面の設計で，互いに並行的に実行する並行処理モジュール群を定める．さらに，資源の利用度合や並行処理モジュールの実行時間などの見積りを正しく導入して性能面の設計も行う．P-Flots は，SPM で構築されたプロトタイプを実行させてその振舞いを視認させ，さらに性能評価データを提示する．単に視認させるだけではなく，プロトタイプの振舞いや性能

データを診断するエキスパートシステムにより，設計者を支援する方法を導入することが有効であると考える．

対象をソフトウェアに限らない一般的な待ち行列ネットワークによる性能評価データを調べて，そのボトルネックとその要因を定性的に診断し，定量的な改善プランを提示するツール BDES (Bottleneck Diagnosis Expert System) を開発している[SAW 89]．並行処理ソフトウェアは，モジュールをサーバに対応させた待ち行列ネットワークとみなすことができる．

本位田・内平は，推論型システム記述言語 MENDEL (MEta iNferential system DEscription Language) とその処理系を開発し，これにより，並行処理ソフトウェアのプロトタイピングを行っている[HON 85b, 86a]．MENDEL は，論理型プログラミング言語を基本に並列型オブジェクト指向言語の要素を統合した言語である．

本位田は，画像処理技術者向きに，プログラミングせずに画像処理パッケージ内の部品群を組合せて所望の画像処理アプリケーションを構成できる画像処理エキスパートシステムを，プロトタイピングシステムとして位置づけている[HON 86b]．

Hooper, J. W. は，分散型データ処理システムのプロトタイピング手法のための実行環境を述べている[HOO 85]．プロトタイピング手法は ADS (Architecture Development System) と名づけられ，分散型データ処理システムのコンフィギュレーションのためのグラフィカルなユーザインタフェースと分散型データ処理システムのプロトタイプの実行環境に特徴をもつ．プロトタイプの実行環境は，複数の VAX 上に置かれた，分散型計算機システムのネットワークインタフェース機能，時間管理機能，タスク管理機能，メモリ管理機能などの諸機能を，通常の VMS 環境と ADS に用意された模擬環境を混在させて実現可能にしている．

Witten, I. H. は，Jade プロジェクトで，分散型計算機システムのソフトウェアの設計とテストのためのプロトタイピングを行っている[WIT 83]．開発用計算機を用いて，この上で分散型ソフトウェアの設計とテストを行った．Jade では，対象計算機のおのおのに載せられるソフトウェアをプロセスと呼び，プロセスの振舞いとプロセス間通信を記述したものをシミュレーションコードと呼んでいる．

Francesco, N. D. は，CSP (Communicating Sequential Process) を拡張した ECSP に基づいた分散型ソフトウェアのプロトタイプを作成する言語 ESL (Event based Specification Language) を提案した[FRA 88]．プロトタイプの実行中に，ユーザが変数の値を変えたり，以前の時点に戻って実行を再開できる対話型デバッグシステムを導入している．

4. おわりに

現段階でのプロトタイピングツールの多くは，それぞれなんらかの考え方に基づいてプロトタイプ構築用の言語を用意し，実行ツールを提供することにより，プロトタイプの構築と実行を計算機により支援するものがほとんどである．3. でみたとおり，このようなツールとして使えるものはほぼ出尽くしているように思われる．最近，このようなプロトタイピングツールやシステムの共通機能に対する要求事項の草案をまとめる作業が行われている[BAL 89]．

一方，プロトタイプがユーザや開発者の要求を満足しているかどうかを調べること，すなわち，プロトタイプの評価自体は人間に頼っているものがほとんどである．2. で述べたとおり，このプロトタイプの評価フェーズに対しても，計算機による支援ツールが整備する方向で今後進むと考える．プロトタイピングが真に実用的なソフトウェア開発手法として定着するためには，この整備が必要であると考える．

参考文献

[ACM 82] ACM SIGSOFT SE Note, Vol. 7, No. 5 (Dec. 1982).

[ARI 86] 有澤 誠：ソフトウェアプロトタイピング，近代科学社 (1986).

[BAL 82] Balzer, R. M., Goldman, N. M. and Wille, D. S.: Operational Specification as Basis for Rapid Prototyping, ACM SIGSOFT SE Notes, Vol. 7, No. 5, pp. 3-16 (Dec. 1982).

[BAL 85] Balzer, R. M.: A 15 Vear Perspective on Automatic Programming, IEEE Trans. SE, Vol. 11, No. 11, pp. 1257-1268 (Nov. 1982).

[BAL 89] Balzer, R. M. et al.: Draft Report on Requirements for a Common Prototyping System, SIGPLAN Notices, Vol. 24, No. 3 (March 1989).

[BEL 86] Belkhouse, B. and Urban, J. E.: Direct Implementation of Abstract Data Types from Abstract Specifications, IEEE Trans. SE, Vol. 12, No. 5, pp. 649-661 (May 1986).

[BOA 84] Boar, B. H.: Application Prototyping: A Requirements Definition Strategy for

the 80s, A Wiley-Interscience Publication (1984). 邦訳：前川 守, 伊藤 潔：プロトタイピングの新応用技術と導入法, 日本技術経済センタ (Dec. 1984).

[BRU 85] Bruno, G. and Marchetto, G.: Rapid Prototyping of Control System Using with High Level Petri Net, 8 th ICSE, pp. 230-235 (Aug. 1985).

[BRU 86] Bruno, G. and Marchetto, G.: Process-Translatable Petri Nets for the Rapid Prototyping of Process Control System, IEEE Trans. SE, Vol. 12, No. 2, pp. 346-357 (Feb. 1986).

[CER 88] Ceri, S. et al.: Software Prototyping by Relational Techniques: Experience with Program Construction Systems, IEEE Trans. SE, Vol. 14, No. 11, pp. 1597-1609 (Nov. 1988).

[DIA 89] Diaz-Gonzales, J. P. et al.: Prototyping Conceptual Models of Real-Time Systems: A Visual Perspective, HICSS '89, pp. 358-367 (Jan. 1989).

[DUN 82] Duncan, A. G.: Prototyping in ADA: A Case Study, ACM SIGSOFT SE Notes, Vol. 7, No. 5, pp. 54-60 (Dec. 1982).

[FRA 88] Francesco, N. D. et al.: Description of a Tool for Specifying and Prototyping Concurrent Programs, IEEE Trans. SE, Vol. 14, No. 11, pp. 1554-1564 (Nov. 1988).

[HER 88] Herndon, R. M. et al.: The Reliable Benefits of a Language Prototyping Language, IEEE Trans. SE, Vol. 14, No. 6, pp. 803-809 (June 1988).

[HON 85 a] 本位田真一, 松本吉弘：リアルタイムシステムにおけるプロトタイピングの1手法, 情報処理学会論文誌, Vol. 26, No. 5, pp. 946-953 (Sep. 1985).

[HON 85 b] 本位田真一, 内平直志, 大須賀昭彦, 柏谷利彦：推論型システム記述言語 MENDEL, 情報処理学会論文誌, Vol. 27, No. 2 (Feb. 1986).

[HON 86 a] Honiden, S., Uchihira, N. and Kasuya, T.: Software Prototyping with MENDEL, Lecture Notes in Computer Science 221 (July 1986).

[HON 86 b] Honiden, S., Sueda, N., Hoshi, A., Uchihira, N. and Mikami, K.: Software Prototyping with Reusable Components, Journal of Information Processing, Vol. 9, No. 3, pp. 123-129 (1986).

[HON 87] 本位田真一, 内平直志, 中村英夫：制御分野における自動プログラミング, 情報処理, Vol. 28, No. 10 (Oct. 1987).

[HOO 85] Hooper, J. W., Ellis, J. T. and Johnspson, T. A.: Distributed Software Prototyping with ADS. 8th ICSE, pp. 216-224 (Aug. 1985).

[ITO 84 a] Itoh, K. et al.: Software Design Process: Chrysalis Stage under the Control of Designers, Journal of Information Processing, Vol. 7, No. 1, pp. 5-14 (Mar. 1984).

[ITO 84 b] 伊藤 潔, 田畑孝一：ソフトウェア設計におけるプロトタイピング, bit 臨増, pp. 166-179 (July 1984).

[ITO 87] 伊藤 潔, 本位田真一, 内平直志：ソフトウェア開発のためのプロトタイピングツール, 啓学出版 (1987).

[ITO 89] 伊藤 潔, 本位田真一, 田村恭久, 沢村淳：シミュレーション手法に基づいた並行処理ソフトウェアのプロトタイピング手法：日本シミュレーション学会学会誌, 近く掲載.

[JAR 88] Jard, C. et al.: Development of Veda, a Prototyping Tool for Distributed Algorithms, IEEE Trans. SE, Vol. 14, No. 3, pp. 339-352 (Mar. 1988).

[KAT 88] 片岡雅憲：ソフトウェアモデリング, 日科技連 (1988).

[KOM 87] 古宮誠一, 原田 実：部品合成による自動プログラミング, 情報処理, Vol. 28, No. 10, pp. 1329-1345 (Oct. 1987).

[LEW 89] Lewis, T. G. et al.: Prototypes from Standard User Interface Management Systems, HICSS '89, pp. 397-406 (Jan. 1989).

[LUQ 88 a] Luqi and Berzins, V.: Rapidly prototyping Real-Time System, IEEE Software, pp. 25-36 (Sep. 1988).

[LUQ 88 b] Luqi and Berzins, V.: A Prototyping Language for Real-Time Software, IEEE Trans. SE, Vol. 14, No. 10, pp. 1409-1423 (Oct. 1988).

[MAS 82] Mason, R. E. A., Carey, T. T. and Benjamin, A.: ACT/1: A Tool for Information Systems Prototyping, ACM SIGSOFT SE Notes, Vol. 7, No. 5, pp. 120-126 (Dec. 1982).

[MAS 83] Mason, R. E. A.: Prototyping Interactive Information Systems, CACM, Vol. 26, No. 5, pp. 347-354 (May 1983).

[ROS 77] Ross, D. T.: Structured Analysis (SA): A Language for Communicating Ideas, IEEE Trans. SE, Vol. 3, No. 1, pp. 16-34 (Jan. 1977).

[SAE 87] 佐伯元司：実行可能な仕様記述, 情報処理, Vol. 28, No. 10, pp. 1346-1358 (Oct. 1987).

[SAW 89] 沢村 淳, 志田圭介, 本位田真一, 伊藤 潔：定性推論を導入した待ち行列ネットワークのボトルネック改善法：情報処理学会知識工学と人工知能研究会, 62-7 (Jan. 1989).

[STE 88] Stelovsky, J. et al.: A System for Specification and Rapid Prototyping of Appli-

cation Command Languages, IEEE Trans. SE, Vol. 14, No. 7, pp. 1023-1032 (July 1988).

[TAM 86] Tamura, Y. and Itoh, K.: Software Prototyping Using Simulation Language, Proc. JSST Conference on Recent Advances in Simulation of Complex Systems, pp. 41-51 (July 1986).

[TAM 87] 田村恭久, 伊藤 潔, 本位田真一: 並行処理ソフトウェアシステムの設計向きプロトタイピング手法とそのツール, 情報処理学会論文誌, Vol. 28, No. 9, pp. 923-932 (Sep. 1987).

[TAV 85] Tavendale, R. D.: A Technique for Prototyping Directly from A Specification, 8 th ICSE, pp. 224-229 (Aug. 1985).

[TEI 77] Teichrow, D. et al.: PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis for Information System, IEEE Trans. SE, Vol. 3, No. 1, pp. 41-48 (Jan. 1977).

[TER 89] Terwilliger, R. B. et al.: PK/c++: An Object-Oriented Logic-Based, Executable Specification Language, HICSS '89, pp. 407-416 (Jan. 1989).

[TSA 88] Tsai, J. J. P. et al.: Rapid Prototyping Using FRORL Language, Compsac 88, pp. 410-417.

[URB 85] Urban, S. D., Urban, J. E. and Dominick, W. D.: Utilizing an Executable Specification Language for an Information System, IEEE Trans. SE, Vol. 11, No. 7, pp. 598-605 (July 1985).

[WAN 88] Wang, Y.: A Distributed Specification Model and Its Prototyping, IEEE Trans. SE, Vol. 14, No. 8, pp. 1090-1097 (Aug. 1988).

[WAS 82] Wasserman, A. I. and Shewmake, D. T.: Rapid Prototyping of Interactive Information Systems, ACM SIGSOFT SE Notes, Vol. 7, No. 5, pp. 171-180 (Dec. 1982).

[WAS 85] Wasserman, A. I.: Extending State Transition Diagrams for the Specification of Human-Computer Interaction, IEEE Trans. SE, Vol. 11, No. 8, pp. 699-713 (Aug. 1985).

[WAS 86] Wasserman, A. I., Pircher, P. A., Shewmake, D. T. and Kersten, M. L.: Developing Interactive Information Systems with the User Software Engineering Methodology, IEEE Trans. SE, Vol. 12, No. 2, pp. 326-345 (Feb. 1986).

[WIT 83] Witten, I. H.: JADE: A Distributed Software Prototyping Environment, ACM SIGOS, Vol. 17, No. 3, pp. 10-23 (July 1983).

[ZAV 81] Zave, P. and Yeh, R. T.: Executable Requirements for Embedded System, 5 th ICSE, pp. 295-304 (Mar. 1981).

[ZAV 82] Zave, P.: An Operational Approach to Requirements Specification for Embedded Systems, IEEE Trans. SE, Vol. 8, No. 3, pp. 250-269 (May 1982).

[ZAV 84] Zave, P.: An Overview of the PAISLey Project, ACM SE Note, Vol. 9, No. 4, pp. 12-19 (1984).

[ZAV 86] Zave, P. and Schell, W.: Salient Features of an Executable Specification Language and Its Environment, IEEE Trans. SE, Vol. 12, No. 2, pp. 312-325 (May 1982).

(平成元年1月20日受付)