計算機アーキテクチャ 107－8
（1994．7．21）

図的仕様記述からのデータ駆動プログラムの生成手法

岩田 誠　寺田 浩詔

大阪大学 工学部 情報システム工学科

機能間の情報の流れを表すデータフロー図は、仕様の自然な表現法の一つとして、多くの要求仕様分析・設計技法に導入されている。しかしながら、従来のプログラミング環境では、逐次処理機械の影響を受けた代入型の記述法を強要されるため、上流工程と下流工程とが乖離し、一貫したソフトウェア開発環境の提供が困難になっている。

これに対して、本稿では、要求に応じて処理が起動される、受動的なデータ駆動原理を目的プログラムの実行原理として採用して、データ (情報) の流れを基本とした機能的仕様をそのまま多重実行可能なプログラムに変換する手法を提案し、これを基礎にした仕様記述環境の構成法について述べる。

Transformation Scheme for Diagrammatical Specification into Data-Driven Program

Makoto IWATA and Hiroaki TERADA

Department of Information Systems Engineering,
Faculty of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565 JAPAN
E-mail: {iwata, terada}@ise.eng.osaka-u.ac.jp

The dataflow diagram is one of natural representations and has been adopted in many requirement analysis and design techniques. However, it is difficult to provide an integrated environment for software development under a conventional programming paradigm, because upper and lower streams are separated due to forcing programmers to translate specifications into assignment based descriptions derived from the sequential machine architecture.

A transformation scheme proposed in this paper generates executable programs from functional specifications, since the data-driven principle is adopted as an execution principle of the final object programs. The paper describes the transformation scheme and a specification environment based on the scheme.

1 はじめに

通信システムや制御システムなどに代表される多重処理ソフトウェアに顕著な特徴は、

- 多様な要求の発生ならびにこれに伴う個々の処理負荷が共に確率的に生起する状況において、指定された機能を、資源／時間制約下で、多重に処理しなければならないこと、
- 運用期間が長く、かつ、サービスの中断が許されないため、保守性・検証性に優れていなければならないこと、である。

現状では、このようなシステムを、イベント駆動、データ駆動、あるいは、要求駆動などを含むデータフロー的なパラダイムを用いて、仕様記述する試みが、いわゆる上流工程向きの CASE として開発され、実用に供される段階に達しているものも多い [1]。

しかし、これらのシステムモデルを実行可能なプログラムに変換する、いわゆる下流工程では、多重分散処理の概念を本来有していない手続き型の逐次代入型言語による記述によらざるを得ない。このために、上流工程でのシステムモデルをこれとは全く異なった逐次代入型処理モデルへ、人手によって、変換せざるを得ない状況にあり、これが上流工程と下流工程とを乖離させ、多重処理プログラムの開発・保守を非常に困難にする原因となっている。

このような上流・下流工程間の乖離を解消するには、上流工程での仕様記述に含まれるシステム構造や振る舞いを、そのまま実行可能プログラムの水準で実現できる、新しいソフトウェア・パラダイムの導入が必須である。すなわち、このようなパラダイムを確立すれば、実現法をほとんど意識しなくとも良い仕様記述の水準で、アルゴリズムの継承・保守を可能にする、ソフトウェア開発環境が容易に構築可能になる。

これに対して、本稿に述べるプログラム生成手法は、上流工程でのシステム仕様記述が、実行可能プログラムの水準では、最終的に機能間のデータ依存性に従ってデータ駆動型に動作するプログラムに帰着される事実に着目している。すなわち、最終的に生成すべき目的プログラムの実行原理にデータ駆動原理を採用することよって、上流工程での仕様記述を下流工程での記述である実行可能プログラムにほぼ同型のまま変換することを意図している。この着想に基づいて、筆者らは既に、仕様記述体系 AESOP(Advanced Environment for Software Production) の構想を提案し [2,3]、この構想を直接的に実証可能なプラットフォームとしてのデータ駆動型マルチプロセッサシステムの構成法を並行して研究している [4,5]。

本稿では、これまで、自動プログラミングが最も困難とされてきた実時間処理分野に典型的に現れる、状態遷移プロセスが多重に実行されるシステムを対象に、自然な仕様記述から多重処理プログラムを直接生成する手法を提案する。2 章ではまず、仕様記述体系の考え方を述べ、変換対象となる仕様記述形式と目的プログラムとなるデータ駆動型プログラムを定義し、3 章では、これらを対象にした変換手法を述べる。

2 AESOP の仕様記述処理体系

2.1 AESOP の概要

ソフトウェア開発過程では一般に、顧客と設計者が協調して、対象システム (ドメイン) を分析し、これを簡単化・抽象化することによってシステムモデリングを行う。どのような観点からモデル化を行うかには、モデル化対象システムのレベル、ユーザの嗜好など様々な要因によって多様性がある [6]。したがって、AESOP では、ユーザが着目したい側面から多面的に仕様化対象システムを定義できるよう、図 1に示すように、主として機能 (構成) を表現する機能ブロック図 (Functional Block Diagram: FBD) やデータ構造を表現するデータ構造図 (Data Block Diagram: DBD) に加えて、動的な振る舞いを表現するシーケンス図 (Sequence Chart: SC)、状態遷移図 (State Transition Diagram: STD)、決定表 (Decision Table: DT) をユーザに提供している。

一方、実行機械として想定している動的データ駆動型プロセッサシステムにも、用途に応じて様々なアーキテクチャやマルチプロセッサ構成が

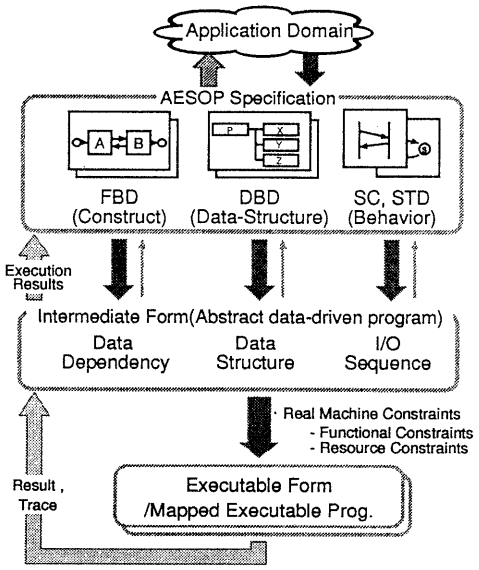

図 1: AESOP の仕様記述処理体系

採られる。例えば、命令セット、タグ処理方式、構造体データ処理方式によって多様なアーキテクチャが存在する。さらに、メモリ容量、マルチプロセッサ構成のネットワークトポロジやプロセッサ数などにも多様性があるため、これらを仮想化して、仕様記述に可搬性を付与することが重要になる。

AESOP の仕様記述処理体系では、以上のような仕様記述、ならびに、実行機械の多様性を吸収するために、処理実行に本質的に必要な情報のみを集約した、中間的なプログラム形式を導入している。すなわち、細粒度の並列多重処理を簡明かつ自然に表現できる、抽象データ駆動型プログラム *ADP(Abstract Data-Driven Program)* 形式を導入し、この水準で、仕様の論理的検証や再利用可能な部品の蓄積などを実現する処理体系を採用した。本処理体系では、図 1の概要図に示すように、AESOP 仕様記述を中間形式 (Intermediate Form)、実行形式 (Executable Form) の順に変換し、プロトタイプ実行の状況や結果をその逆順に変換して、仕様記述に反映する。このようなフィードバック機能によって、ユーザが特定のプログラミング言語の知識を必要としないで、仕様記述水準で直接にソフトウェアの定義・検証を行なえるよう構想されている。

2.2 状態を伴う処理の多面的仕様記述手法

多数の状態遷移プロセスが協調して動作するシステムは、一般に制御分野や通信分野に頻出する基本的な処理形態である。

このような処理形態の仕様化においては、適切なモジュール分割だけでなく、(i) 副アルゴリズムの選択に必要な記憶 (状態) の抽出、(ii) 入出力イベントの抽象化あるいは統合化、(iii) 状態の階層的分割あるいは直並列分割により、見通しの良いシステムとしてモデル化することが極めて重要である。したがって、AESOP 仕様記述手法には、これらの記述要素を簡潔かつ明示的に表現できる記法を導入している。

状態抽出のための表記 ：機能ブロック図 (FBD) によるモジュール表現として、通常の関数的処理モジュールに加えて、ファイルなどの共用記憶へのアクセス・モジュール、ならびに、副アルゴリズムを選択するための状態を持つモジュールを導入し、記憶の影響範囲の明示的定義を可能にした。

イベント抽象化のための表記 ：いわゆるデータ構造の構成子により複合データ構造を図的かつ階層的に表現できる記法をデータブロック図 (DBD) に導入した。

状態分割のための表記 ：上述の状態モジュールの振舞を状態遷移図 (STD) により定義し、さらに、これらの詳細な振舞を階層的に FBD および STD により定義できるよう、表現形式間の対応関係を明確に定義した。

これらの仕様記述の表記法を用いれば、図 2 に示すように、状態遷移プロセスが多面的かつ階層的に定義できる。左上の FBD 記述 α は、モジュール i が有限状態機械 (FSM) により制御されることを明示している。この FSM はまた、右

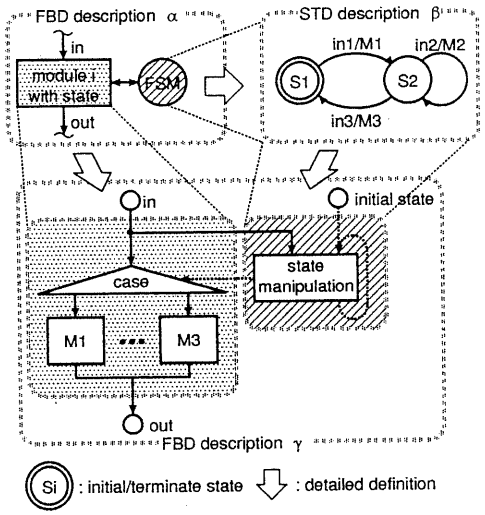

図 2: 状態遷移処理の多面的図的仕様記述手法

上図の STD 記述 β により詳細に定義される。そして、STD 記述に定義された状態遷移論理に従って、副アルゴリズムを選択する構造として、下図のような FBD 記述 γ として定義される。

このように状態遷移に関与する箇所の明示的表記法をユーザに提供することによって、仕様記述処理系が状態操作部を容易に分離・抽出でき、次節に述べる、抽象データ駆動型プログラム *ADP* への直接的変換が可能になる。

2.3 状態を伴う抽象データ駆動型処理モデル

純粋なデータ駆動原理は履歴依存性のある一般的な処理を扱えない。そのため、AESOP では、関数透過性を保存したまま、(副) アルゴリズムの選択のための状態遷移処理、および、履歴依存性のある構造体データ処理を規定できるよう、拡張した動的データ駆動型処理モデルを導入した。

本節では、前者の状態遷移処理を中心に、その解釈実行規則を提案する。

2.3.1 ハイブリッド・データ駆動型実行原理

本処理モデルでは、状態ならびに入力イベント系列による一連の状態遷移の実行シーケンスを抽象化するために、ストリーム概念を導入して、自然に状態遷移プロセスの多重実行が実現される、以下のような、実行規則を採用した。

定義 1 (Stream) 線形順序を持つ要素からなり、すべての要素が必ずしもそろっている必要がないという性質を持つデータ構造をストリームと呼ぶ *[7]*。 □

定義 2 (Firing rule) あるノードの全ての入力にトークンが揃い、かつ、入力アーク上の利用可能なトークンと同一のストリームに属するトークンが出力アーク上に存在しない時のみ、そのノードは発火可能である。 □

このように、本処理モデルは、単一のストリームに関する実行を静的データ駆動型発火規則により規定し、複数の独立なストリームに関する同時並行型多重実行を動的データ駆動型発火規則により規定する。これによって、

i. ストリームに関する識別子だけがいわゆるタグ処理の対象になるため、不要なタグ処理を排除した本質的な処理構造を表現できる。

ii. マルチストリームに対する多重処理プログラムの検証が、データ駆動グラフの構造の検証問題と、タグ処理の首尾一貫性の検証問題に分離でき、探索空間の縮退が可能になる。

2.3.2 状態を伴うデータ駆動型処理構造

上述のような静的・動的データ駆動による混合型実行規則を導入した処理モデルでは、以下の性質が満足される。

定理 1 (validity(acyclic)) 対象とするデータ駆動型プログラムが非巡回グラフであり、かつ、*well-structure[8]* である場合、グラフの入力ストリームの各要素に対応した要素を持つストリームが出力される。 □

すなわち、入力ストリームの要素トークン数と同数の要素を持つストリームがプログラムの出力

となるため、原理的に並列処理に伴う副作用のない安全な実行が可能である。

しかしながら、状態遷移処理のデータ駆動表現には、帰還ループが必須であり、巡回グラフを含めた検証性の保証が必要になる。このため、状態遷移論理を実現するデータ駆動型処理構造(以下、FSM構造と呼ぶ)を図3に示すように定義し、これをトークンに関する生成・消費に着目した関数性を満たす処理ノードとして局所化する形態で状態を伴う処理を規定する。

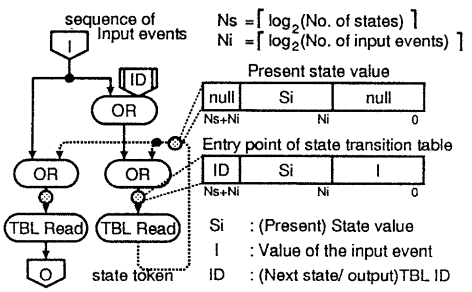

図3: FSMの抽象データ駆動型プログラム構造

図3のFSM構造は、仕様化された状態遷移論理に応じて、パラメトリックに定数および参照テーブル(次状態テーブルと出力テーブル)を変更するだけでよいので、仕様記述からの直接的な変換が可能である。この処理構造によって、状態更新処理ループ(図中点線)を局所化・極小化でき、部品化、ならびに、単位時間あたりの処理率の極大化が可能になる。すなわち、次の性質を満たすため、安全な実行が保証される。

定理 2 (validity(cyclic)) 対象とするデータ駆動型プログラムが巡回グラフである場合、状態遷移論理モジュールを除いたグラフが非巡回グラフで、かつ、*well-structure* であれば、定理 *1* の性質が保証される。 □

さらに、図4に示すように、動的データ駆動原理により、文脈の切替え処理がなくとも複数の入力データストリームを受理できるので、複数の状態遷移系列が非同期に独立に進行する処理に関して、実行機械の命令セットならびにタグ処理方式が隠蔽された、抽象データ駆動型プログラムが定義可能になる。また、構造体データ処理方式に関しても、構造体データを多次元ストリームとして捉えれば、本処理モデルの拡張として規定できることが期待される。

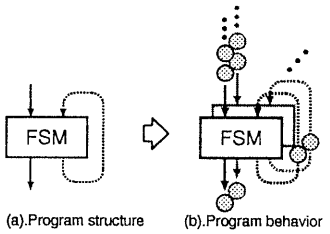

図4: FSM構造の動的データ駆動型多重実行

3 データ駆動型プログラムの生成手法

本章ではまず、多面的かつ図的なAESOP仕様記述からの抽象データ駆動型プログラムへの直接生成手法を提案し、本手法によれば、図的仕様記述からデータ駆動型プロセッサQv-x[4][5]上で高度並列に実行可能なプログラムが原理的に直接生成可能なことを示す。

さらに、本手法を基礎にすれば、多面的に定義された仕様記述相互間での変換を介した対話的な仕様記述支援環境が容易に構築できることを示す。

3.1 抽象データ駆動型プログラム要素の生成

AESOP仕様記述は、前述したように、機能とその間のデータ依存性に関する情報を主として表現する機能ブロック図を中心として、データ構造図やシーケンス図などにより多面的に定義される。したがって、この仕様記述からの抽象データ駆動型プログラムの生成の問題は、機能とその間のデータ依存性を核にして、これにデータ集合や振る舞いの情報を矛盾無く統合する規則の定式化の問題に帰着する。

AESOPでは、記述能力や抽象データ駆動型処理モデルの将来的な拡張性を残すために、プリミ

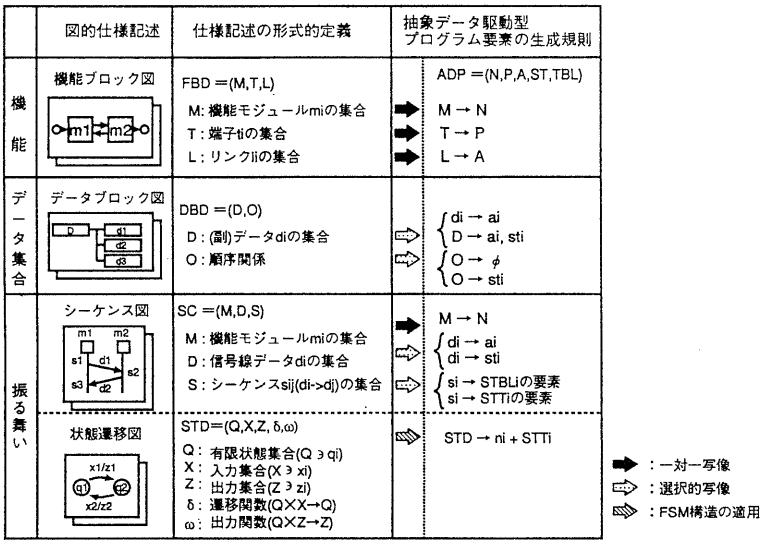

図 5: 図的仕様記述からの抽象データ駆動型プログラム要素の生成規則

ティブな図的記述要素に対する生成規則を定め、これらの規則により生成されたプログラムの構成要素を、その時点で既に生成されているプログラム情報に統合する手法を採用した。

図 5に示したADP の生成規則の一覧は、以下のADP の定義に基づき、集合 (の要素) 間の写像として定式化されている。

定義 3 (Abstract Data-Driven Program) 抽象データ駆動型プログラム *ADP* は、5 つ組 $ADP = (N,P,A,ST,TBL)$ で表現される。

N: ノード n_iの集合
P: ポート p_iの集合
A: アーク a_iの集合
ST: ストリーム st_iの集合
TBL: スタブ表 $(STBL_i)$ および
状態遷移表 (STT_i) の集合 □

[変換手法 a] 機能とその間の接続関係の変換 : 機能ブロック図中の、モジュール m、端子 t、および、リンク l は、それぞれ *ADP* のノード n、ポート p、および、アーク a にほぼ一対一に変換可能である。

[変換手法 b] データ集合の変換 : タグを付与されたトークンストリーム *ST*、あるいは、データ依存アークの集合 *A* として変換される。また、ファイルアクセスが明示される場合には、永続的記憶のためのデータ構造として直接的に変換される。

[変換手法 c] 振舞情報の変換 : 下位階層が未定義の機能モジュールに対して、その入出力の因果関係を抽出し、これからスタブの入出力情報 *(Stub TBL)* を構成し、早期プロトタイピングを可能にする。さらに、状態が明示されるか、あるいは、状態が存在する可能性のあるシーケンス集合が検出された時点で、図 3 に示した処理構造からアクセスされる状態遷移表 *(STT)* として再構成する。

このように本手法では、個々のプログラム要素への一対一変換により、抽象データ駆動型プログラム構造が直接的に生成されるために、いわゆる、

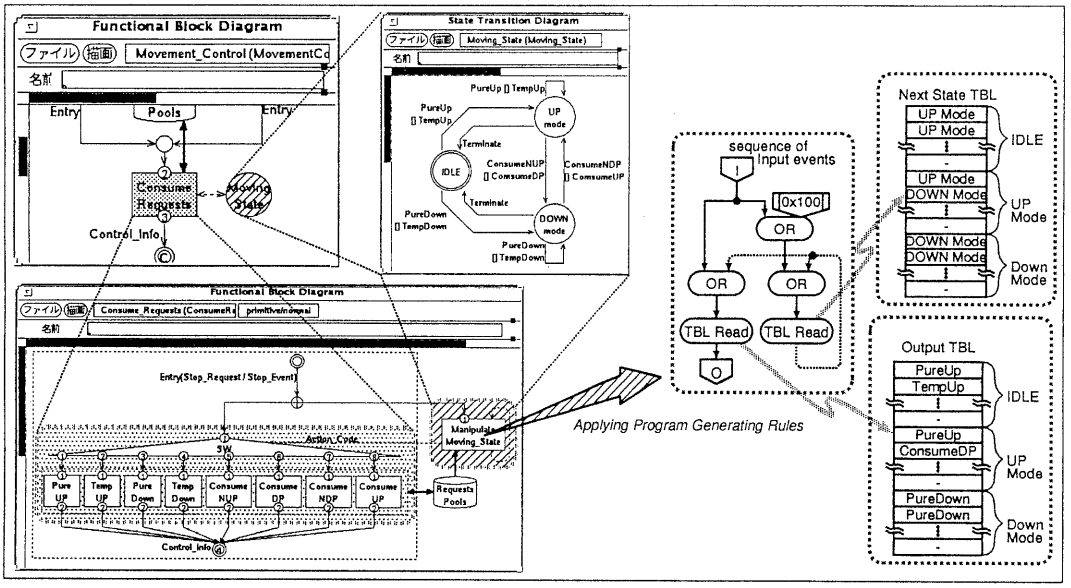

図 6: リフト制御問題の仕様記述例とその ADP の生成例 (一部)

加法的変換 (インクリメンタルな変換) が可能である。また、新たな表現形式を追加した場合にも、その表現形式と抽象データ駆動型プログラムとの対応を生成規則として、追加すればよく、既存の生成機構には何ら影響を与えない利点がある。さらに、本生成手法を基礎にして、各生成規則の逆変換規則を定義しさえすれば、後述する多面的仕様記述間の相互変換が容易に定式化できる。

3.2 図的仕様記述過程の対話的支援手法

対象システムの初期開発あるいは保守に限らず、利用者が当初から明確な要求仕様をイメージしていることは極めて稀であるため、利用者の仕様記述を完結する方向に誘導することがユーザフレンドリな環境の実現には必須である。したがって、本環境では、適切なアドバイスあるいは問合せを利用者に発生する観点から、

i. 仕様記述において、データ駆動的に解釈できない箇所、すなわち、実行不可能な箇所を検出して、

ii. これに対して、異なる側面を表現する仕様記述形式の相互間の変換、(相互変換) ならびにダイアログを通して、ユーザに問い合せて、

完全な仕様記述へと誘導する手法を採用した。

具体的には、以下の情報を検出して、これらの箇所に対して図 5の生成規則を逆に適用して、相互変換を施し、利用者にその定義を促す。

機能 機能の駆動条件の未定義・矛盾箇所 (巡回グラフ構造など)。

情報 (データ) 機能に関する情報との対応関係の未定義・矛盾箇所。

振る舞い 断片的な振る舞い情報をスタブの入出力表あるいは状態遷移表として統合する際の、機能に関する情報との対応関係、ならびに、表中の未定義・矛盾箇所。

3.3 リフト制御問題への適用

AESOP の仕様記述手法ならびにプログラム生成手法の有効性確認のため、仕様記述のベンチ

マーク問題であるリフト問題 [9] を取り上げた。仕様化する際に留意した点は、(i). リフト稼働要求イベント群を対象とした典型的な生産者・消費者問題としてモデル化すること、ならびに、(ii). 各リフトを制御する状態遷移処理が各リフトの状態とリフト稼働要求イベントに応じて同時並行に多重実行すること、である。

このリフト制御問題の AESOP 仕様記述の例を図 6に示す。この仕様記述例は、図 2に示した、仕様記述モデルにしたがって、状態遷移を伴う消費者側の機能を仕様化した例である。リフト停止要求 (*Stop_Request*) を入力イベントとして、現状態に応じて、副アルゴリズムを選択する機能 (*Consume_Requests*) を定義している。

図 6の右図は、この状態遷移モジュールの仕様から生成される抽象データ駆動プログラムの一部である。ここでは、図 5の生成規則を適用して、図 3に示した FSM 構造を生成した例を示している。このプログラムは、データ駆動型プロセッサ RAPID[5] の命令セットへの変換、ならびに、タグ処理の挿入によって、多重実行可能なオブジェクトに容易に変換可能であることが確認された。

4 おわりに

本稿では、データ駆動原理のシステム表現能力ならびに並列処理原理の自然さに着目して、図的表現を許す仕様記述から直接的に高度並列に実行可能なデータ駆動型プログラムを生成する手法を提案し、本手法に基づく仕様記述支援手法の素案を示した。

実際に本処理モデルに基づくプログラム生成手法をリフト制御問題に適用した結果、DBD, STD により情報を補完すれば、仕様記述中の FBD から、ほぼ一対一に対応したプログラムが直接生成可能なことを確認した。

本手法は、ソフトウェアに限らず、ハードウェア機能の仕様記述環境にも原理的に適用可能であるため、ソフトウェア・ハードウェアを含めたシステム仕様記述環境の実現にも有効な手法に発展することが期待できる。

一方、本稿に述べたような仕様記述環境では、性能対費用比の削減を支援する機能もまた重要である。このため我々は、実行機械の特性にチューンした実行形式プログラム部品の開発・蓄積を効果的に支援できる環境を提供するために、データ駆動型プロセッサシステムの視覚的評価環境の構成法もまた並行して検討している [10]。

謝辞 御指導、御支援頂いた AESOP 研究の関係各位ならびに御協力頂いた寺田研究室の各位に深く感謝する。

参考文献

[1]. A.Fuggetta: *A Classification of CASE Technology, IEEE Computer*, Vol.26, No.12, pp.25-38 (1993).

[2]. 西川、寺田 他: 超高位図的仕様記述環境 (AESOP) の構想", 情報処理学会計算機アーキテクチャ研究会, 90-ARC-83-2, pp.7–12 (1990).

[3]. M.Iwata and H.Terada: *Multilateral Diagrammatical Specification Environment based on Data-Driven Paradigm, Proc. ACM ISCA Workshop on Data-Flow Computing*, (1992) (to be published).

[4]. H.Terada, M.Iwata et al.: *Superpipelined Dynamic Data-Driven VLSI Processors, Proc. ACM ISCA Workshop on Data-Flow Computing*, (1992) (to be published).

[5]. S. Komori et al.: *A 50 MFLOPS Superpipelined Data-Driven Microprocessor, Proc. ISSCC '91*, pp.92–93 (1991).

[6]. C.A.Vissers et al.: *Architecture and Specification Style in Formal Descriptions of Distributed Systems, Proc. the 8th International Workshop on Protocol Specification, Testing and Verification*, pp.189–204 (1988).

[7]. 寺田、西川 他: VLSI 向きデータ駆動型プロセッサ:Q-x, 信学論 (D), Vol.J71-D, No.8, pp.1383–1390 (1988).

[8]. J.B.Dennis: *Dataflow Schemas, Project MAC, M.I.T.*, pp.187-216 (1972).

[9]. *Problem Set for the Fourth International Workshop on Software Specification and Desgin, Proc. 4th International Workshop on Software Specification and Desgin* (1987)

[10]. 藤生、岩田、寺田: 動的データ駆動型プロセッサシステム Q-x の視覚的評価環境, 情処全大第 47 回, 6G-6 (1993).