データ駆動パラダイムに基づく
統合的システム記述体系に関する研究

1996年12月

岩田　誠

データ駆動パラダイムに基づく
統合的システム記述体系に関する研究

１９９６年12月

岩田　誠

内 容 梗 概

本論文は筆者が大阪大学大学院工学研究科在学中および大阪大学に在職中に行ってきたデータ駆動型並列・分散処理システムの構成方式に関する研究のうち、データ駆動パラダイムに基づく統合的システム記述体系に関する研究をまとめたものであり、次の6章をもって構成する。

第1章序論においては、本研究の目的ならびにその工学上の意義、および関連分野での研究の現状について述べ、本研究で得られた新しい成果について概説する。

第2章においては、ソフトウェアとハードウェアからなるシステム全体の統合的な開発・保守の方法論として、従来の逐次代入型の文章記述に代わり、データ駆動パラダイムを基礎とする図的な統合的システム記述の継承・保守を重要視したシステム開発手法について述べる。まず、これまでの統合的システム開発方法論の一つのアプローチである、統合化CASE (Computer-Aided Software Engineering) ならびにハードウェア・ソフトウェア協調設計手法における問題点が、アーキテクチャ上の欠点を強く反映したプログラム記述の困難さにあることを示すと共に、統合的なシステム記述の体系に求められる要件を明らかにする。さらに、データ駆動原理に基づくパラダイムがシステム設計のあらゆる水準で有効に機能することを明らかにし、統合的システム記述によるシステム設計環境の理論的基礎として非常に有望であることを示す。

第3章においては、図的表現を中心とした統合的なシステム記述体系AESOP(Advanced Environment for System Oriented Production) の概要について述べる。さらに、統合的システム記述のための多面的かつ図的な仕様記述手法、ならびに、その変換のために拡張した抽象データ駆動型処理モデルとして状態を伴う処理モデルを提案する。最後に、本体系の根幹となる技術として、多面的な図的システム仕様記述から抽象データ駆動型システム記述 (プログラム) をインクリメンタルに直接生成する手法について述べ、実現法に依存しない水準でのシステム記述の継承・保守を可能とする基盤を与える。

第4章においては、AESOP システムをユーザにとってより親しみやすい仕様記述環境として提供するために、システムとの対話を通してインクリメンタルに仕様を記述可能

とする、仕様記述支援手法について述べる。すなわち、多面的に定義されたシステム仕様記述相互間での変換を介した対話的な有効性確認・検証支援、ならびに、ソフトウェア/ハードウェア実現法の選択のために必須となる性能評価支援の一手法を提案する。

第5章においては、AESOP システムの実現法として、今後の拡張性を考慮した自己適用的な実現法を提案し、これに基づいて実現したプロトタイプシステムとして、信号流れ図からデータ駆動型プロセッサ上で実行可能なプログラムを直接生成するシステムの構成について述べる。さらに、本システムを実時間制御システムならびに画像信号処理システムの仕様に適用した結果として、多面的な図的記述の例とその変換例を示し、本体系によりシステム記述からデータ駆動型 VLSI プロセッサ Qv-x 上で実行可能なオブジェクトプログラムが直接生成されることを実証的に明らかにする。

第6章においては、本研究で得られた結果と、本研究の成果をより一貫した統合的なシステム開発・保守体系へと発展させる上での課題についてまとめる。

関 連 発 表 論 文

（1）論文誌等に発表した論文

[1] 寺田浩詔, 西川博昭, 岩田誠, 岡本俊弥, 宮田宗一, 小守伸史, 嶋憲司: "VLSI 向きデータ駆動プロセッサ:Q-x," 電子情報通信学会論文誌 D, Vol. J71-D, No. 8, pp. 1383–1390 (1988-08).

[2] Hiroaki Terada, Makoto Iwata, Souichi Miyata, and Shinji Komori: "Super-pipelined Dynamic Data-Driven VLSI Processors," In J.-L. Gaudiot L.Bic and G.R.Gao, editors, *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press, pp. 75–85 (March 1995), also presented at the 2nd International Workshop on Dataflow Computers (May 1992).

[3] Makoto Iwata and Hiroaki Terada: "Multilateral Diagrammatical Specification Environment Based on Data-Driven Paradigm," In G. Gao, J.-L. Gaudiot, and L. Bic, editors, *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press, pp. 103–112 (March 1995), also presented at the 2nd International Workshop on Dataflow Computers (May 1992).

[4] 岩田誠, 寺田浩詔: "図的仕様記述からのデータ駆動型プログラムの生成手法," 情報処理学会論文誌, Vol. 36, No. 5, pp. 1203–1210 (1995-05).

[5] 唐沢圭, 新吉高, 岩田誠, 寺田浩詔: "信号流れ図からのデータ駆動型プログラムの対話的生成手法," 電気学会論文誌C, Vol. 116-C, No.11, pp.1307–1312 (1996-11).

[6] 上方輝彦, 岩田誠, 滝根哲哉, 寺田浩詔: "分散キューバッファを持つデータ駆動型プロセッサ Qv-x の性能評価," 電気学会論文誌C, Vol. 116-C, No.11, pp.1295–1300 (1996-11).

[7] Makoto Iwata and Hiroaki Terada: "Stream-Oriented Data-Driven Architecture with Functional Data Memory," *Transactions of Information Processing Society of Japan*, (submitted for publication).

（２）国際会議等に発表した論文

[1] Hiroaki Terada, Yan Xu, Makoto Iwata, Tetsuya Takine, and Koso Murakami: "Flow-Thru Processing Concept and its Applications to Soft-Computing," *Proc. 4th International Conference on Soft Computing*, pp. 105–108 (October 1996).

[2] Hiroaki Terada, Makoto Iwata, and Souichi Miyata: "600MOPS Super-Pipelined Data-Driven Processors and Their Application to HDTV Signal Processing," *Proc. Australasian Computer Architecture Workshop'96*, (to be published).

目 次

第1章

序　論

ソフトウェアとハードウェアが有機的に統合した情報通信システムの生産性・保守性の向上は、情報機器の社会への浸透とULSI技術の進展を背景として、非常に深刻な課題となっており、新しい統合的システム構成パラダイムへのシフトが望まれている[1, 2, 3, 4]。今後、LSI素子の微細化技術によってシステム内に集積可能な素子数が増加するにしたがって、システムに要求される機能も多様かつ高度なものが求められることが多くなることは必須である。また、システムの使用環境が十分に特定されていなかったり、未知の環境に適応することが可能なシステムが求められるようにもなる。このようなシステムの開発・保守においては、当初想定されていた基本機能に加えて、システムを漸増的に発展・成長させることが最も有効である。すなわち、システムへの要求仕様の追加に伴って、それがソフトウェアないしはハードウェア実現法にそのまま反映可能な柔軟なシステム構成方式の確立が望まれる。

システムの仕様記述は、少なくとも要求仕様定義の段階では、その機能が問題であり、実現法とは独立な機能記述の自然な形式での柔らかな定義が求められる。さらに、この機能記述の構造をそのまま継承し、最終的な実現法に直接に反映できれば、一貫した系統的なシステム開発・保守を多人数で協調的に行うことが可能になる。しかしながら、現状では、開発の比較的初期の段階で、ハードウェア実現部分とソフトウェア実現部分とに完全に分離され、以降はそれぞれの仕様が個別に設計 (実現) されることが多い。これは、特にソフトウェア仕様部分に関して、さらに、機能仕様とは全く異質な構造を持つ逐次代入型のプログラム仕様への構造変換が要求されるため、既に確立されている系統的なハードウェア設計技法とは相容れないことに起因している。

これまでにも、ソフトウェア工学やVLSIシステム設計工学の分野で個別にではあるが、

統合的な開発・設計技術の研究が進められてきている。ソフトウェア工学の分野では、当初、仕様記述から実行可能プログラムを自動生成する自動プログラミング技術 [5] の確立が唱われたが、未だに確立されてはいない。これは、仕様記述の水準では、対象とする問題の構造・振舞いを視覚的に表現する、多様な図的記述が活用されているにもかかわらず、プログラムとして構想し記述する段階では、問題の持つ構造の素直な反映が非常に困難な、現行のプログラミング言語による文章型記述への写像を余儀なくされるためである。したがって現状では、プログラミング言語で表現可能な構造についての豊富な知識を持ち、かつ、対象とする問題の持つ構造に深い洞察力を持った、経験豊富なシステムエンジニアの助けを借りなければならないことが、多くの深刻な問題を生み出している。これらのシステムエンジニアを支援する目的で、仕様記述水準からコード化に至るまでを統合的に支援する統合化 CASE が提案されている [1, 6] が、上述の観点から、原理的な解決がなされている訳ではない。一方、VLSI システム設計工学の分野では、昨今、ハードウェア/ソフトウェアを同時に協調して設計する手法が着目されている [7, 4, 8]。しかしながら、ハードウェアとソフトウェアからなるシステム全体の仕様を統一的に記述するツール (記述) は、現在のところまだ実用的なものは存在しないと言われている [9]。その根本的な原因は、いずれもノイマン型処理方式を基礎としたプロセッサ・アーキテクチャを前提として、既存の CPU コアを流用したり、プロセッサ自身を設計するアプローチを採っていることにある。すなわち、ノイマン型プロセッサ上で制御駆動型に実行される逐次代入型プログラム構造の設計と、階層的な動作記述・構造記述を用いた系統的なハードウェア設計技法とでは、両者の動作原理が構造的に異なるため、統一的に扱うことが困難であることに起因する。

このような課題に対応するために、本研究は、ハードウェア論理とプログラム処理とを柔らかなシステム仕様として統合的に記述し、これらを系統的に漸次詳細化する過程を通して、自律分散的システムを ULSI 実現するための、新しいシステム構成パラダイムの確立を目指している。すなわち、

a) ハードウェア/ソフトウェア実現法とは独立な水準で、了解性に優れた図的表現を用いたシステムの機能や振舞いの多面的な記述によって、多人数での協調的なシステム仕様定義・設計を可能とする統合的システム記述体系、ならびに、

b) 分散的な自己タイミング型ハードウェア処理機能と多数のプログラム処理機能とが

チップ上で有機的に統合され、また複数のチップが自由に直接的に結合できる、柔軟なシステム構成方式、

を、了解性・安全性に優れたデータ駆動原理を基礎として、確立することを目的としている。

本論文は、第一の目的である統合的なシステム記述体系に関して、その根幹となる、図的な仕様記述からデータ駆動型システム記述を直接生成する体系 [10, 11, 12] とその記述支援環境 [13, 14, 15, 16] についての研究結果をまとめたものである。第二の目的に関してはこれまでに、筆者らの共同研究グループによって、自己タイミング型パイプライン機構による専用ルータ機能とこれによって相互接続された動的データ駆動型マルチプロセッサが単一チップ内に集積化された実績が報告されている [17, 18]。本論文では、このようなシステム構成方式が基本的に実現可能であることを踏まえて、統合的システム記述体系が提案されている。

一般に、基本的なシステム構想の段階では、システムの機能と構成の直観的な理解が要求されることが多いため、システム仕様記述には、多くの場合、図的な表現が多用される。本提案は、このような図的な記述をデータ駆動原理に従って解釈すれば、副作用のない自律分散処理が規定されるため、機能記述の各階層で、機能間の相互依存関係を自律分散型の (ソフトウェア/ハードウェア) アルゴリズムとして解釈実行可能なことを基礎としている。

この目的を達成するために、本研究では、システムの発注者にも容易に理解可能な半形式的な図的記述により多面的に要求仕様を記述できる手段を導入し、以後データ駆動パラダイムに基づいて、これらの仕様に含まれる機能・振舞いを実現法の水準まで一貫的に保存したまま変換する手法を提案する。本手法は、実現法にそのまま写像可能なデータ駆動型プログラム記述を仕様記述から直接変換することによって、応用分野の専門技術者が、その専門領域で一般的に用いられる図的仕様記述形式を用いて、システムの設計・開発を多人数で協調的に行える環境を提供しようとするものである。これによって、システム開発者は、特定の形式的仕様記述形式やプログラミング言語に煩わされることなく、所望の機能を実現するアルゴリズムの探求に純粋に専念することが可能になる。

以下に本論文の構成、ならびに新しく得られた成果を述べる。

第 2 章においては、ソフトウェアとハードウェアからなるシステム全体の統合的な開発・

保守の方法論として、従来の逐次代入型の文章記述に代わり、データ駆動パラダイムを基礎とする図的な統合的システム記述の継承・保守を重要視したシステム開発手法について述べる。まず、これまでの統合的システム開発方法論の一つのアプローチである、統合化CASE (Computer-Aided Software Engineering) ならびにハードウェア・ソフトウェア協調設計手法における問題点が、アーキテクチャ上の欠点を強く反映したプログラム記述の困難さにあることを示すと共に、統合的なシステム記述の体系に求められる要件を明らかにする。さらに、データ駆動原理に基づくパラダイムがシステム設計のあらゆる水準で有効に機能することを明らかにし、統合的システム記述によるシステム設計環境の理論的基礎として非常に有望であることを示す。

第3章においては、図的表現を中心とした統合的なシステム記述体系 AESOP(Advanced Environment for System Oriented Production) の概要について述べる。さらに、統合的システム記述のための多面的かつ図的な仕様記述手法、ならびに、その変換のために拡張した抽象データ駆動型処理モデルとして状態を伴う処理モデルを提案する。本処理モデルに基づくプログラム記述は、複数の入力ストリームを副作用なく多重に解釈実行可能とするものである。次に、本体系の根幹となる技術として、多面的な図的システム仕様記述から抽象データ駆動型システム記述 (プログラム) を加法的 (インクリメンタル) に直接生成する手法について述べ、実現法に依存しない水準でのシステム記述の継承・保守を可能とする基盤を与える。

第4章においては、AESOP システムをユーザにとってより親しみやすい仕様記述環境として提供するために、システムとの対話を通してインクリメンタルに仕様を記述可能とする、仕様記述支援手法について述べる。まず、多面的に定義されたシステム仕様記述相互間での変換を介した対話的な有効性確認・検証支援手法を示す。さらに、ソフトウェア/ハードウェア実現法の選択のために必須となるソフトウェア実現法の性能評価の一支援手法として、理論的な性能見積もり手法ならびに実験的な性能評価のための視覚的支援手法を示す。

第5章においては、AESOP システムの実現法として、今後の拡張性を考慮した自己適用的な実現法を提案し、これに基づいて実現したプロトタイプシステムとして、信号流れ図から2種類のデータ駆動型 VLSI プロセッサ上で実行可能なプログラムを直接生成する試作システムの構成について述べる。さらに、本システムを実時間制御システムならびに

画像信号処理システムの仕様に適用した結果として、多面的な図的記述の例とその変換例を示し、本体系によりシステム記述からデータ駆動型 VLSI プロセッサ Qv-x 上で実行可能なオブジェクトプログラムが直接生成されることを実証的に明らかにする。

第 2 章

統合的システム記述とデータ駆動パラダイム

2.1 緒言

情報通信システムの設計・開発、ならびに保守・運用のために、ソフトウェアの要求分析から VLSI チップの設計手法に至るまで、これまでに様々な方法論が提案されている。ソフトウェア工学の分野では、構造化分析・設計手法やオブジェクト指向分析・設計手法などが提案され、実用に供されている手法もある。また、VLSI 設計に関しては、計算機支援設計手法の研究の成果を受けて、多くの EDA(Electronic Design Automation) ツールが商用化されている。

これに対して、ソフトウェア開発の上流工程と下流工程を統合的に支援する統合化 CASE (Computer Aided Software Engineering) や、主として組み込み用途のシステムの開発を統合的に支援するハードウェア・ソフトウェア協調設計手法など、システム開発の様々な水準で、各開発工程を統合的に支援する試みがなされている。最終的には、システムに対する要求を分析する工程から VLSI 設計の工程に至るまでを、一貫して統合的に実施可能なシステム構成パラダイムの確立が必須である。

以下本章ではまず、これらの統合的システム開発方法論の主要なアプローチである統合化 CASE ならびにハードウェア・ソフトウェア協調設計手法について概観し、これらの方法論がノイマン型プロセッサあるいは逐次代入型処理モデルの概念から出発したものであり、統合的なシステム構成パラダイムとして、そのまま発展しうるものではないことを明らかにする。同時に、統合的なシステム構成パラダイムの満たすべき要件として、1) システムの構造・動作を自然に表現でき、かつ、解釈実行可能な処理モデル、2) この処理モデルを直接実行可能なプロセッサ・アーキテクチャ、ならびに、3) このアーキテクチャを ULSI 実現可能なハードウェア構成原理、に関する基本的性質を明確にする。次に、統

合的なシステム構成パラダイムの確立において、簡明でかつ厳密な解釈によって、システムの (並列処理) 構造ならびに動作を自然に表現可能なデータ駆動原理が有効であることを示す。すなわち、システム記述の水準からハードウェア構成の水準に至るまで一貫してデータ駆動原理を理論的基盤として導入でき、あらゆる水準で乖離のない、シームレスなシステム開発・保守に対して有効であることを示す。

2.2 統合的システム記述とその実現法

現状のシステム開発方法論の統合化の試みは、ソフトウェア工学の分野とハードウェア、すなわち、VLSI システム設計工学の分野に個別に見受けられる。これらを、対象とする基本構成要素の機能レベルと、利用されるハードウェア機器 (部品) とを軸として整理すると、**図 2.1**のようになる。この図からもわかるように、これまで比較的個別に発展してきた、上流 CASE 環境、プログラミング環境、および CAD 環境に対して、これらを相互に一貫して支援するための統合化 CASE 環境ならびにハードウェア・ソフトウェア協調設計技術の研究が進められてきた。

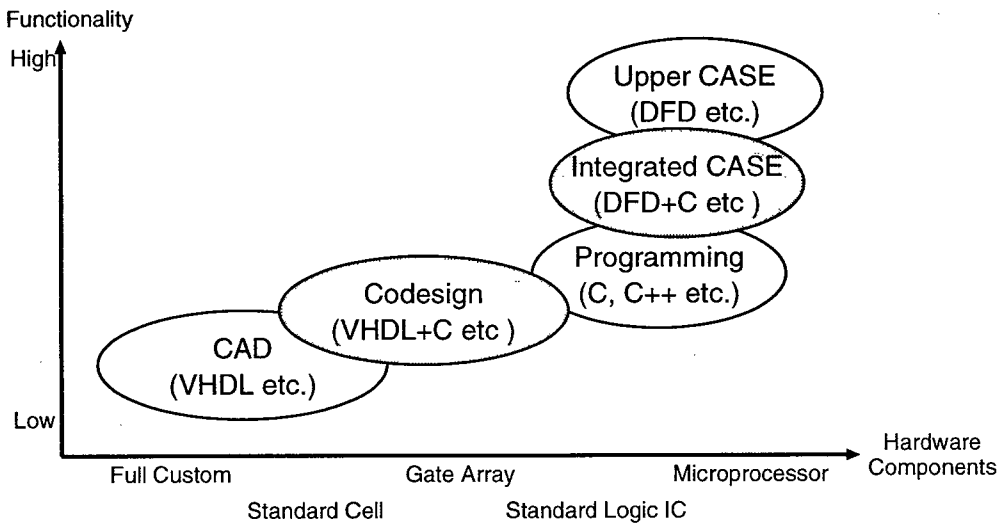


図 2.1 現状のシステム開発環境

これらの研究動向に対して、本研究の最終目的は、**図 2.1**に示す 2 次元の全空間を支援可能な統合的なシステム開発の方法論を確立し、多様な要求に柔軟に対処でき、かつ、系統的にシステム開発・保守が可能な環境の提供によって、情報・通信システムの生産性・

保守性を向上させることにある。以下では、現状の統合化の主要な試みを概観し、統合的なシステム記述とその系統的実現法の要件を明らかにする。

2.2.1 システム記述とその処理実行モデル

ソフトウェアの品質と生産性を向上するために、1980年代後半から、CASE (Computer Aided Software Engineering) ツールに大きな期待がかけられてきた。CASEツールは当初は要求分析作業の支援から始まり、現状では、対象とするシステムの要求仕様を、イベント駆動、データ駆動、あるいは、要求駆動などのパラダイムを用いて、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]。そして、次第に支援範囲が拡大され、ソフトウェア開発の上流から下流までを一貫して支援する統合化CASEへと発展させる研究もみられる [6]。

しかしながら、現在のソフトウェア生産の種々の問題は、仕様記述に始まるいわゆる上流工程とコーディングに終わるいわゆる下流工程の間にある、越え難い溝に起因するものが多い。たとえば、仕様記述の水準では、対象とする問題の構造・振舞いを視覚的に表現する、多様な図的記述が活用されているにもかかわらず、プログラムとして構想し記述する段階では、問題の持つ構造の素直な反映が非常に困難な、現行のプログラミング言語による文章型記述への写像を余儀なくされる。この過程では、多くの場合、プログラミング言語で表現可能な構造についての豊富な知識を持ち、かつ、対象とする問題の持つ構造に深い洞察力を持った、経験豊富なシステムエンジニアの助けを借りなければならないことが、多くの深刻な問題を生み出している。特に現行では多くの場合、下流工程で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ため、上流工程でのシステムモデルとは全く異なった処理モデルに、人手によって、変換せざるを得ない状況にある。これが上流工程と下流工程とを乖離させ、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。

たとえば、これまでに、ソフトウェア開発手法として、**図 2.2**に示すようなデータフロー図 (DFD: Data Flow Diagram) を中心とした構造化分析・設計技法 [19, 20] やジャクソン法 [21] などが提案されているが、これらのほとんどが最終的な目的プログラムの実行原理に逐次代入型処理モデルを想定しているため、要求仕様定義からプログラミングに至る一貫した、ソフトウェアの生産・保守が不可能である。また、要求仕様定義からプログ

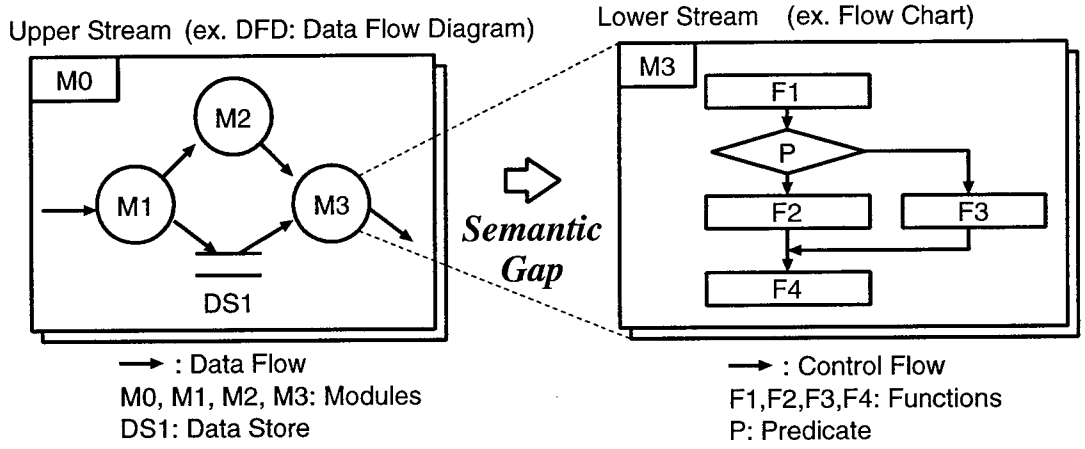


図 2.2 構造化分析・設計の流れ

ラミングに至るまで一貫して、同一のパラダイムを用いる方法論として、オブジェクト指向開発手法 [2, 3] があるが、(i) パイプライン型並列処理を自然に定義できないこと [22]、(ii) 逐次代入型言語をオブジェクト指向風に拡張したプログラミング言語を用いることが多いため、副作用の回避が原理的に困難なこと、(iii) 最終的な実行可能オブジェクトが、ノイマン型逐次処理マシン上のプログラムであるため、ii と同様の困難があること、という問題がある。

このような上流・下流工程間の乖離を解消するためには、下流工程の問題を上流工程に適合するように解決するのが正しい方法である。すなわち、上流工程でのシステム仕様記述として素直に定義された処理構造とその振舞いをそのまま継承して、これを解釈実行可能なハードウェアの導入によって、上流工程と下流工程の区別なくどの水準でも一貫して同一の方法論を適用可能なソフトウェア開発・保守パラダイムを確立することが望ましい。

2.2.2 システム記述の処理実行モデルとハードウェア実現法

ソフトウェアとハードウェアを一体化した新しいディジタルシステム設計のために、1990年代に入って、ハードウェア/ソフトウェア協調設計 (Hardware/ Software Codesign) の確立をめざした研究が進められている [7, 4, 8]。ハードウェア/ソフトウェア協調設計は、特定用途向け集積回路 (ASIC: Application-Specific Integrated Circuits) を用いたものから、特定用途向け命令セット・プロセッサ (ASIP: Application-Specific Instruction Processors)

を用いたもの、あるいは、既存の標準的なプロセッサを用いたものまで、様々なレベルのものがある。多くの場合、各種のシステムの制御部分として利用される組込みシステムに適用されている [23, 24]。

しかしながら、ハードウェアとソフトウェアからなるシステム全体の仕様を統一的に表現するツール (記述) は、現在のところまだ実用的なものは存在しないと言われている [9]。その根本的な原因は、いずれもノイマン型処理方式を基礎としたプロセッサ・アーキテクチャを前提として、既存の CPU コアを流用するアプローチ、あるいはプロセッサ自身を設計するアプローチを採っていることにある。すなわち、ノイマン型プロセッサ上で制御駆動型に実行される逐次代入型プログラム構造の設計と、階層的な動作記述・構造記述を用いた比較的系統的なハードウェア設計技法とでは、両者の動作原理が根本的に異なるため、統一的に扱うことは不可能である。たとえば、**図 2.3** は、逐次代入型のプログラム記述とハードウェア水準の機能ブロック図を対比的に示している。両者の根本的な相違はその解釈実行原理にある。この図からも、機能間のデータフローを素直に表現するブロック図が構造記述として系統的に用いられてきた理由が容易に理解できる。

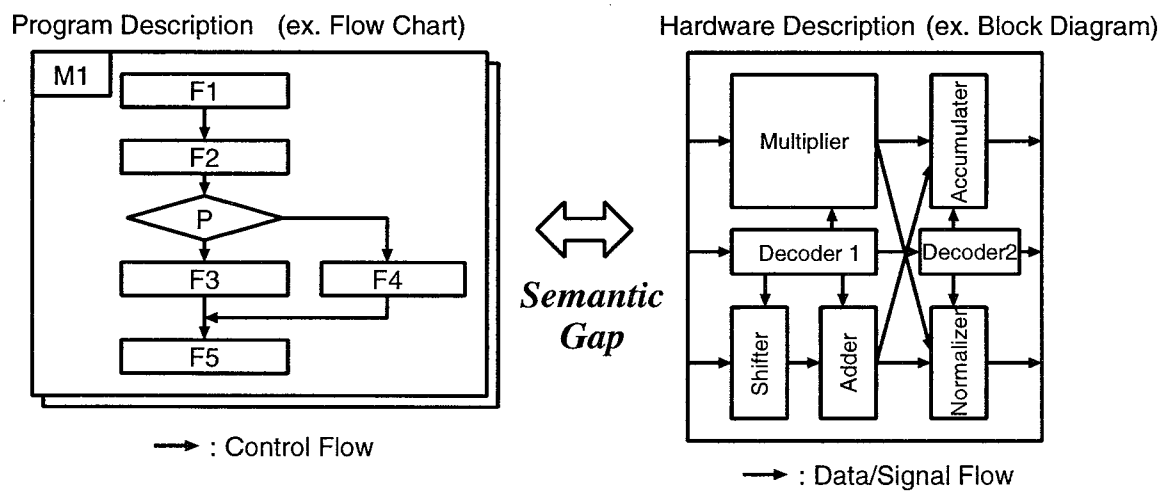


図 2.3 逐次代入型プログラム記述とハードウェア記述

Woo らはオブジェクト指向型の機能記述言語をシステム全体の統一的な仕様記述言語として用いる手法を提案している [4]。この手法では、ハードウェアで実現される部分はハードウェア記述言語に、ソフトウェアで実現される部分はプログラミング言語に変換されてそれぞれ実現される。しかしながら、現状のオブジェクト指向言語には、前述のよう

な根本的な問題点があるため、これを解決しない限り、実用化は困難である。

このプログラム処理とハードウェア論理の記述との間にある乖離を解消するためには、既に確立されている系統的なハードウェア記述と継ぎ目なく相互に変換可能なプログラム記述を導入し、要求されるシステム機能やコスト対性能に応じて、柔軟にソフト・ハード実現部分を変更可能な統一的システム仕様記述体系を確立することが重要である。このためには、プログラム記述やプロセッサ・アーキテクチャの基礎となる抽象マシンモデルを、システム機能の構造・振舞いを自然に表現可能で、かつ、ハードウェアの動作原理と継目なくインタフェース可能なものに置き換える必要がある。

2.3 データ駆動パラダイムの必要性とその特徴

統合的システム記述から系統的にシステムを実現し保守するためには、少なくとも、要求仕様定義水準の記述にはその実現法に依存しない自然な表現形式を導入する必要がある。そして、これらの記述を実現法の水準まで一貫的に保存することによって、要求仕様定義水準でもシステムの継承・保守を可能にすることが重要である。このためには、前節で明らかにしたように、システムの実現法として、システム記述上に表現される機能・振舞いをそのまま解釈実行でき、かつ、ソフトウェア・ハードウェアを継目なく実現可能な処理モデルを導入し、これによって一貫したシステム開発を可能とすることが要求される。

2.3.1 データ駆動原理

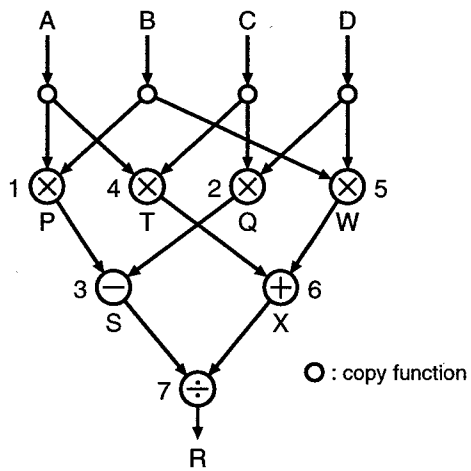
本研究は、このような要件を満たし、かつ、安全性に優れたデータ駆動原理に着目することによって、統合的システム記述体系の確立を目指したものである。このデータ駆動原理は、次に定義されるように、非常に自然で簡明な処理実行の原理である。

定義 2.1 (データ駆動原理) 処理機能を表すノード (節点) 群とその間のデータ依存関係を表すアーク (有向枝) からなる計算グラフを対象として、『あるノードに入力する全てのアーク上に (データの存在を表す) トークンが揃った時そのノードの実行が可能になる』制御原理をデータ駆動原理とする [25]。 □

したがって、データ駆動原理は、(i) 処理機能とその間のデータ依存関係による明示的なデータの授受、ならびに、(ii) 実行に必要なデータの可用性に基づく処理駆動原理 (発

火原理)、とによって、並列処理を素直に表現でき、かつ、副作用のない並列実行を規定する。特に、動的データ駆動型処理方式は入力アーク上に複数のデータの組〈トークン〉の滞留を許し、ハードウェア構成が許す限り、任意の個数のデータの組が単一のプログラムを共有し、さらにデータの組相互間の追い越しが許されるために、アーク上に単一のトークンの組しか許さない静的データ駆動型処理方式 [25] に比べて、はるかに柔軟な同時並行・パイプライン型並列処理能力を有している [26]。

データ駆動型プログラム記述は了解性・再利用性にも非常に優れているため、プログラムさらにはシステム記述の生産性・保守性を向上するための基盤としても捉えることが可能である [27]。すなわち、データ駆動原理では、入・出力が明確であり、モジュールの接続によって、構成要素自体の機能が変化することはないため、データ駆動原理に基づいて構成された部品は、階層的な拡張によって、任意の複雑さ (粒度) を持った部品として機能する。たとえば、高機能なソフトウェアモジュールから、ゲート水準の AND/OR 機能に至るまで任意の水準の部品を想定できる。また同時に、ノードの配置とは無関係に、ノード相互間の位相幾何学的な関係が不変であるため、プログラマの恣意が入る余地が全く無く、本質的なデータ依存性を陽に表現できる了解性に優れた記述の作成が可能である。これらの性質は特に、再利用が最も功を奏する保守の領域で大きな利点となる。



(a) 計算グラフ記法

1 P＝A×B

2 Q＝C×D

3 S＝P−Q

4 T＝A×C

5 W＝B×D

6 X＝T＋W

7 R＝S÷X

(b) 逐次代入型記述

図 2.4 データ駆動型記述と逐次代入型記述

以上の特徴は、図 **2.4**に示すごく簡単な例からでも容易に理解できる。たとえば、図 **2.4**(a) のデータ駆動型記述ではノードの空間的位置を変更しても、アークにより表現されるノード間のデータ依存関係は不変である。さらに、各ノードは必要なデータが揃った時のみ起動されるため、この記述の一部を実行したとしても何ら副作用を生じない。一方、同一の計算を逐次代入型記述により表現した図 **2.4**(b) では、文番号 1 と 2、4 と 5、(1,2,3) と (4,5,6) の交換が可能であり、同一のプログラムであっても多様な全順序型プログラムが生成されるという欠陥がある。また、この記述は一般にノイマン型処理方式によって実行され、変数の値を格納する記憶セル (レジスタあるいはメモリ) 内に所望のデータが存在するかどうかとは無関係に、プログラムカウンタが示す文番号によってのみ実行制御されるため、誤ったデータを対象に文を実行する危険性を原理的に有している。

さらに、アーク上に複数のトークンの組を許す動的データ駆動方式の場合には、図 **2.5** に示すように、黒色のトークン D の未着のために、黒色の組の実行が進行不可能な状況でも、後から到着した灰色のトークンの組の実行が進行できる。このように動的データ駆動方式は、メモリアクセスやプロセッサ間通信などに伴う種々の遅延が生じても、実行可能なデータの組から前進的に処理を進行できるという、遅延耐性もまた生来的に有している。

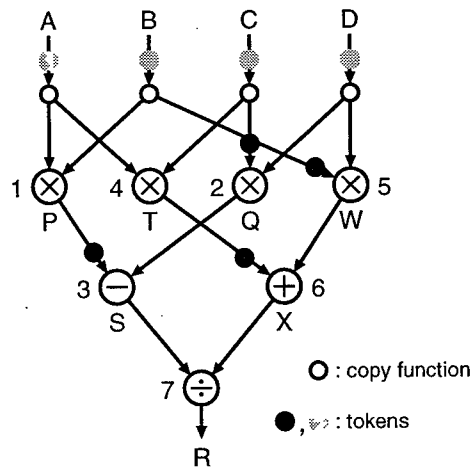


図 2.5 動的データ駆動型処理方式

2.3.2 データ駆動パラダイム

上述の性質を利用して、これまでにも、データ駆動原理による次のような試みが個別に行なわれている。たとえば、要求仕様水準ないしはソフトウェア仕様水準では、データフロー的なパラダイムを用いて、図的に仕様を記述する試みが多くなされている [19, 21, 1, 3]。古くは構造化分析手法から、最近では、OMT 法などのオブジェクト指向分析手法などにおいて、**図 2.2**の左図に示すデータフロー図 DFD(Data Flow Diagram) が用いられ、商用 CASE ツールにも実装されている。また、ソフトウェアライフサイクルの過程で意味のある図的表現を活用することを目的とした視覚的プログラミング環境の領域でも、データフロー図を用いたシステムが多く見受けられる [28, 29, 30]。これらは、データ駆動原理が機能の構成を表す図的表現と非常に馴染みやすいことを示唆している。

システムの詳細設計水準では、厳密なデータ駆動原理に従うプログラムは、機能の入出力端子がデータ依存関係により明確に定義され、かつ、その入力に必要なデータの組が与えられない限り、そのプログラム・モジュール (部品) は起動されることがないという特質を用いて、徹底した部品化によるプログラム生産・保守が試みられている [31, 32]。**図 2.6**(文献 [32] から引用) は、予め定められた部品しか使用させないように徹底し、ディジタル交換機の制御プログラムをデータ駆動論理図 DDL(Data-Driven Logic) と呼ばれる言語により記述し、実際の構内交換機 PBX に実装した例である。この事例では、DDL 記述がノイマン型プロセッサ上でエミュレートされているにも関わらず、SDL などを用いた従来法に比べて、設計、コーディング、テストの時間がそれぞれ 4 分の 1 に削減できたと報告されている。この事例は、最近注目を浴びているクリーンルーム手法 [33] におけるボックス構造化手法が、データ駆動型処理モデルによって原理的に実現されていることを示唆しているとも言える。

実行機械の水準でも、トークンの追い越しをも許す動的データ駆動型処理方式の処理装置が各所で試作されている [34, 35, 36, 37, 38]。我々の研究室でもこれまでに、柔軟なパイプライン型並列処理が可能でありかつ遅延耐性のある、VLSI 向き動的データ駆動型プロセッサ Qv-x を共同で試作してその性能評価を進めている [39, 40, 17, 18]。たとえば、信号処理向きデータ駆動型 VLSI チップ Qv-s (Data-Driven Processor Q; VLSI chip developed by Sharp Corp.)[17] は、**図 2.7**(a) の OCP(Operation and Control Processor) のブロック

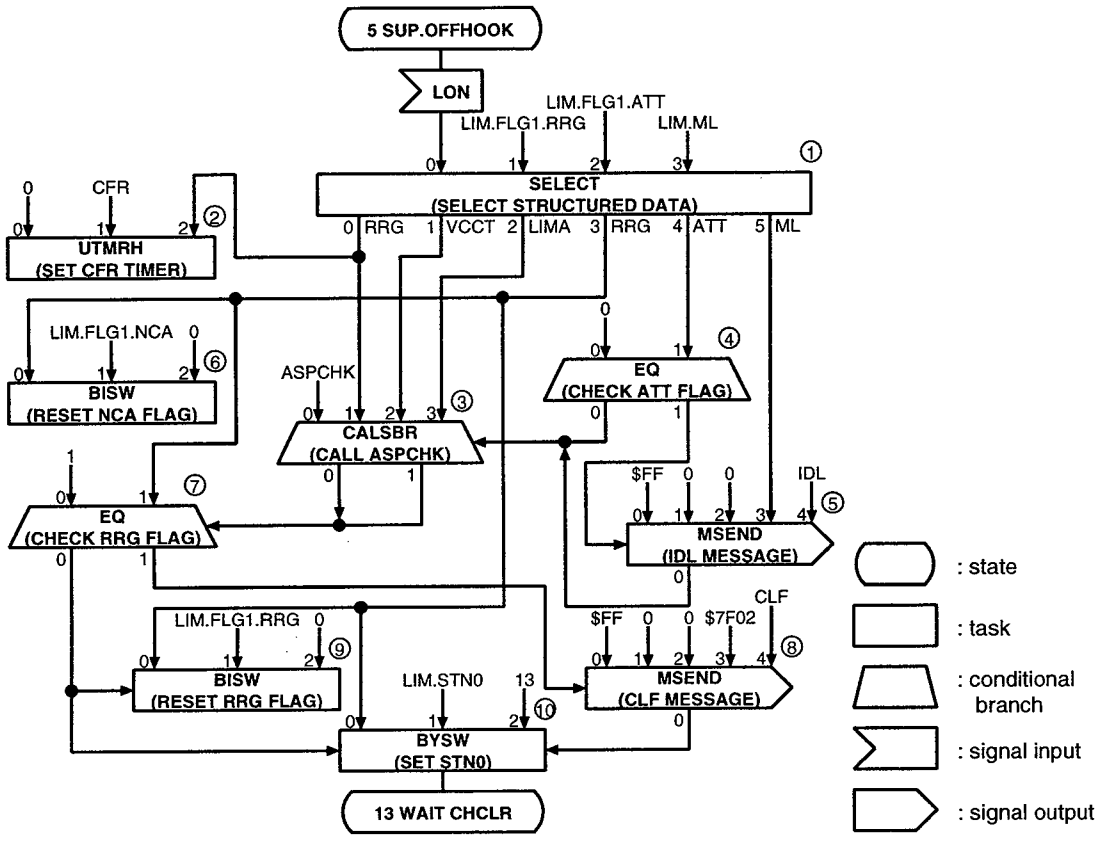


図 2.6 図的データ駆動言語 DDL による徹底した部品化例

図に示すように、4個のナノプロセッサ (INT, SYNC, TBL, GNT) をパケットスイッチ機構 (IO) により接続した形態で単一チップ内に集積化されており、600～700MOPS(Mega Operations Per Second) の実効性能を達成する。これは、データ駆動型処理では、原理的にいったん発火した処理は決して中止されず、パイプラインの制御が非常に簡単になるので、非常に多段のパイプライン処理が可能であるためである。現に Qv-s の各ナノプロセッサは、数十段のパイプライン機能により実現されているため、高いスループットを達成できる。たとえば、**図 2.7**(c) に示す整数・論理演算用ナノプロセッサ INT は 21 段の環状パイプライン機構により構成されている。Qv-x は一連の共同研究で開発されたプロセッサ **Q** の総称であり、プロセッサ全体が上記のようにパイプライン処理機構から構成され、一種の待ち行列 (Queue) 的な動作をする VLSI チップという意味から命名されいてる。

さらに、ハードウェア論理水準では、データ駆動型に自律的に動作する形態をとる、自

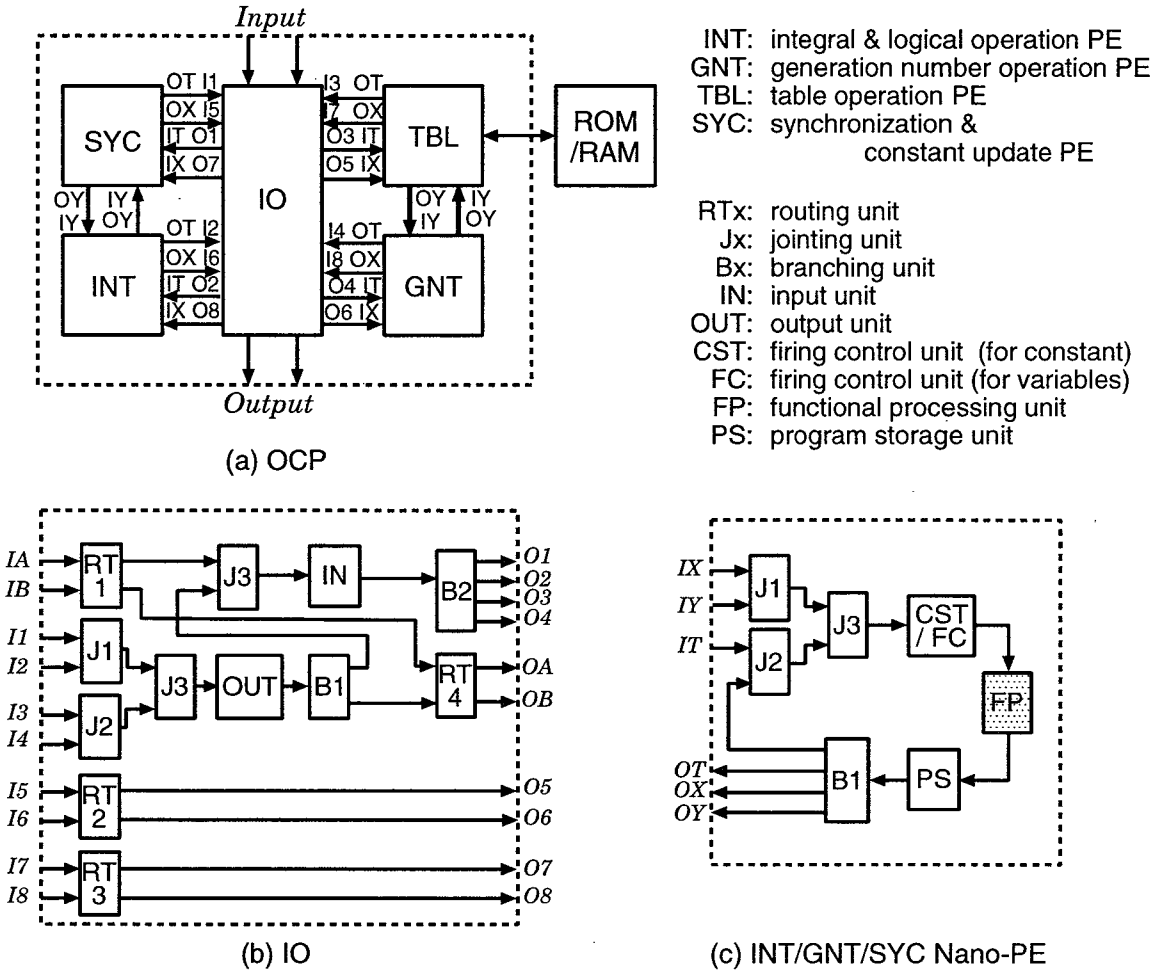


図 2.7 データ駆動型 VLSI プロセッサ Qv-s の構成

己タイミング型パイプライン機構が提案されている [41, 42]。このハードウェア機構は、**図 2.8**に示すいわゆるシェイクハンド形データ転送機構により、ラッチを多段に接続した構成をとる。この構成では、大域的なクロック同期が必要なくクロックスキューの問題が回避でき回路の高速化が可能になる。同時に、この回路は速度独立論理の基本素子の一つである C 素子 (Coincidence element) から構成されるため、ハードウェア機能記述から論理回路を自動的に設計する論理合成の分野の一つの問題であるタイミング検証の問題も回避できる。さらに、この回路を駆動する電流はパイプライン段間に局所化されるので小電力化が可能である、と同時に、データの処理中にのみ電力を消費するだけであるため、非常に低消費電力なシステムを構成可能である。これらの特徴から、上述のデータ駆動型 VLSI プロセッサのハードウェア実現法に採用され、平均 3W で動作する VLSI チップが

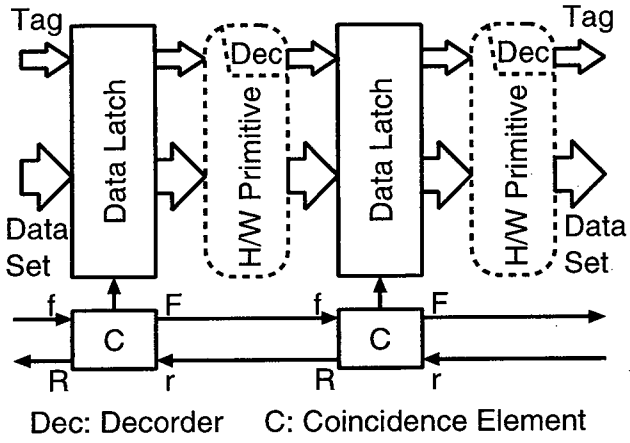


図 2.8 自己タイミング型パイプライン機構

実現されている [17]。また、この自己タイミング型パイプライン処理機構は、パイプライン間の相互作用によってより柔軟な集積化機能ブロックとしても機能する可能性も有している。このため、VLSI アルゴリズムやシストリック [43]・アルゴリズムを包含する、ハードウェア・アルゴリズムをそのままの形態でシリコン・ウェハ上に展開するすることが可能である。

以上のように、データ駆動原理はその特徴を活かして、システム仕様記述の水準からハードウェア実現法の水準に至るまで様々な水準で活用されている。以下本論文では、このようにデータ駆動原理を発展させた、記述法、処理モデル、ならびにアーキテクチャの背景となる包括的な考え方を**データ駆動パラダイム**と呼ぶ。

このような環境を統合的に活用すれば、仕様記述体系へのデータ駆動パラダイムの一貫した適用によって、要求定義水準の図的仕様記述がほぼ同型のまま実現法に写像可能なため、仕様記述水準でデータ駆動プログラムの持つ検証性・部品化能力を利用でき、かつ、プロトタイプを高度並列に実行できる体系の実現が可能である。この試みの一つが次章以降に述べる統合的なシステム仕様記述体系 AESOP (Advanced Environment for System Oriented Production) であり、システム設計者が、要求仕様記述水準でシステムの統合的な定義・検証を行なえ、かつ、可搬性のあるシステム記述の継承・保守が可能になる体系の確立を目的としている [10, 11, 12, 44, 45]。

2.4 結言

本章では、ソフトウェアとハードウェアからなるシステム全体の統合的な開発・保守の方法論として、従来の文章型記述に代わる、データ駆動パラダイムを基礎とする図的な統合的システム記述の継承・保守を重要視したシステム開発手法について述べた。まず、これまでのシステム開発の統合化のアプローチにおける問題点が、アーキテクチャ上の欠点を強く反映したプログラム記述の困難さにあることを示すと共に、統合的なシステム記述の体系に求められる要件を明らかにした。さらに、データ駆動原理に基づくパラダイムがシステム設計のあらゆる水準で有効に機能することを明らかにし、統合的システム記述によるシステム設計環境の理論的基礎になりうることを示した。

今後の課題として、このデータ駆動原理の特徴を機能デバイス水準のハードウェア設計にまで一貫して適用できる可能性を追究することが考えられる。すなわち、純粋なデータ駆動原理では、機能を表すノード間で授受されるトークンは、データの存在を表す抽象的な概念であり、データの構造や型に関しては言及されない。したがって、行列などの構造体データから多値あるいはアナログデータに至るまで、多様なデータが混在したシステムを想定して、これらを相互に変換する機能を用意すれば、矛盾無く解釈できる。また、データ駆動型プログラムは、それがハードウェア・ブロック図あるいは論理回路図としても解釈できる。従って、データ駆動型プロセッサは、ハードウェア機能図のインタプリタの機能を有していると言える。即ち、データ駆動型処理方式を導入すれば、多値・アナログ混在システムを前提としても、ソフトウェアとハードウェアとが相互にシームレスに変換可能なシステムを実現できる可能性を有している。これに関しては、より一般的にデータ駆動パラダイムを捉えた研究の進展が期待される。

第 3 章

図的システム仕様記述からのデータ駆動型プログラム記述の生成手法

3.1 緒言

前章では、了解性・安全性に優れたデータ駆動パラダイムを基礎とすれば、ことさらソフトウェア/ハードウェアを区別することなく、あらゆる機能水準で統合的なシステム記述を継承・保守できるシステム開発を原理的に採用できる可能性があることを述べた。

しかしながら、一般に複数人で協調してシステムの開発・保守が行われる状況が多いことを踏まえれば、システムの発注者からシステム開発の専門家に至るまでの広範囲のユーザが容易に理解できるシステム記述の導入が不可欠である。さらに、このシステム記述から実行可能なプログラムを直接生成して、迅速なプロトタイピングを可能にすれば、ユーザは、特定の形式的仕様記述形式やプログラミング言語に煩わされることなく、所望の機能を実現するアルゴリズムの探求に専念することができる。

第一の目的に対しては、広範囲のユーザにとって自然な半形式的な複数の図的表現を許す多面的な仕様記述法の導入を試みた。これは、要求仕様定義を、システムの発注者側にも容易に理解可能な図的表現、を用いて視覚化すれば、システムがどのように実現されるかについての知識を有しない人々にも、極めて有効な手段となることは既に多くの例によって認められているためである [28, 46, 47]。たとえば、(i) 機能ブロック図のように、システムの当面の機能とその継承・展開を想定した図的表現、(ii) このシステムに対する入出力データの構造への要求を示すユーザ・インタフェースの図的表現、あるいは、(iii) システムに対する入・出力相互間の時間的関係を示すシーケンス図、などが広く用いられている。

第二の目的に対しては、純粋なデータ駆動原理では規定できない履歴依存処理を含むよう拡張した抽象データ駆動型処理モデルを導入し、この水準で多面的かつ図的なシステム記述から獲得した情報の安全性を保証した上でプロトタイプを実行する方法を採った。すなわち、図的システム記述から中間的な抽象データ駆動型プログラム記述へ直接変換し、これをソフトウェアないしはハードウェアとして実現する方法を採った。

以下本章では、図的援助を用いて半形式的に定義されたシステムの要求仕様に、形式的な図的設計仕様記述を漸次付加して詳細化し、最終的にソフトウェア/ハードウェアとして直接実現可能な抽象データ駆動プログラム記述を生成する体系について述べる。まず、本システム記述の体系についてその概要を述べる。次に、半形式的な図的表現によるシステム仕様記述手法、ならびに、履歴依存処理のうち状態遷移処理を副作用なく規定できる抽象データ駆動型処理モデルを提案し、最後に、多面的な図的システム仕様記述から抽象データ駆動型プログラムを加法的(インクリメンタル)に生成する手法を示す。

3.2 統合的システム記述体系の概要

統合的システム記述体系 AESOP(Advanced Environment for System Oriented Production) では、**図 3.1**の概要図に示すように、要求仕様水準の記述に用いられた半形式的な図的記述の情報を実現法の水準にまで、効果的に継承するために、複数の半形式的図的表現によって多面的かつ階層的な要求仕様記述を許している。システムの設計仕様の水準では、これらの記述をそのまま継承し、これに実現手法上必要となる、データ構造とその処理アルゴリズムを加えて、漸次階層的に詳細化を繰り返して、抽象的なデータ駆動型プロセッサ上で実行可能なプログラムを生成する。この中間的なデータ駆動型プログラム ADP(Abstract Data-Driven Program) の水準は、要求仕様から継承した設計仕様に含まれるアルゴリズムとデータ構造を表現し、この水準で論理的検証や部品の蓄積を行なう機能を実現している。

さらに、本体系では、コスト性能比、さらには、特定の実現法上の機能的制約や資源制約を考慮して、実行可能なソフトウェア形式 (Executable Software Form) やハードウェア記述 (Hardware Description) を ADP から自動生成する方式を採っている。この実現法を想定したプロトタイプ実行の状況や結果は、その逆順に変換され、図的仕様記述上に多面的に視覚化される。すなわち、次章に述べるシミュレーション・エミュレーション技法を

用いて、性能評価支援が行われる。

AESOP ではこのように、ADP の水準で可搬性を保持しているため、原理的に任意の実行機械を想定可能であるが、多様な粒度の並列処理を自然に実行できるデータ駆動型プロセッサ Qv-x(Qv-1, Qv-m, Qv-s)[40, 39, 17] を対象とすれば、システム記述の実現法の水準まで一貫した仕様記述処理体系の構築が可能である。

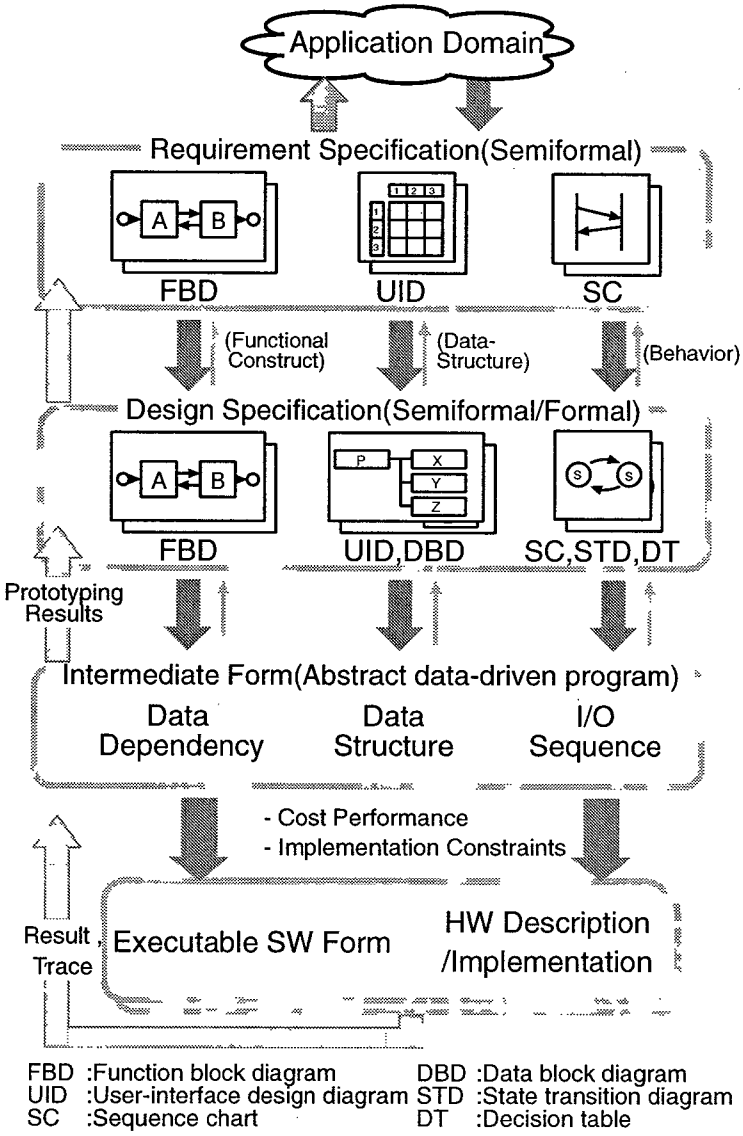


図 3.1 統合的システム記述体系 AESOP の概要

詳細は後述するが、**図 3.1**の上部に示すように、要求仕様水準でも利用可能な半形式的図的表現として、機能構成と機能相互間のデータ、イベントの依存関係を表す機能ブロック

図 FBD(Function Block Diagram)、ユーザインタフェースの図的表現 UID(User Interface Design Diagram)、および、システムの振舞いを表現するシーケンス図 SC(Sequence Chart) が提供される。さらに、主としてシステムの詳細仕様化時に半形式的な要求仕様記述を補完する形式的図的表現として、データ構造を表すデータブロック図 DBD(Data Block Diagram)、状態遷移論理を表わす状態遷移図 STD (State Transition Diagram)、および、選択的処理を表す決定表 DT(Decision Table) が提供される。

3.3 多面的な図的システム仕様記述と拡張データ駆動型処理モデル

3.3.1 多面的な図的システム仕様記述手法

要求定義水準では、当初から、形式的に仕様が記述されることは稀であり、これが設計水準の仕様記述に継承された段階でも、思考錯誤的に次第に仕様が洗練化されるのが一般的である。したがって、AESOP では、要求定義水準の記述に半形式的な図的記述を導入し、この記述からも抽象データ駆動型プログラムの生成に必要な情報を積極的に抽出して、システムの発注者ないしは設計者に、早期にフィードバックをかける立場を採用している。しかしながら、最終的な詳細設計水準の仕様記述の段階では、データ構造とその処理アルゴリズムを厳密に定義しなければならない状況も生じる。このため AESOP では、(i) 一般に用いられる半形式的な図的記述の拡張、(ii) 理論的基礎を有する形式的な図的表現の導入、および、(iii) 複数の半形式的な図的記述間の相互関係の明示手法の提供、によって形式性を付与し、これらを有機的に用いた仕様記述が可能な環境を提供している。

このような図的仕様記述手段を提供するには、仕様記述の了解性と実行可能プログラム生成の両者の観点から、その定義・解釈を決定することが重要である。特に、データ駆動パラダイムを基礎にした AESOP では、システム仕様記述に内在する並列処理性がそのまま保存される基本的特性があるため、この特性を積極的に活用した図的仕様記述手段の定義とその解釈とに基づいた多面的な仕様記述手法が採られている。

(a) 機能ブロック図を中心とした図的表現の解釈方針

多くの応用領域で一般的に用いられている機能ブロック図は、機能モジュールをノード（矩形あるいは円）で表現し、その間の接続関係をリンク（矢線）で図 **3.2**のように表現する。この図には、再利用の単位となる機能モジュールに関する情報、ならびにモジュー

ル間のデータ依存関係に関する情報といった、処理実行に本質的な情報が含まれている。このため、多面的なシステム仕様記述体系では、機能ブロック図を中心的な表現形式として導入し、これに種々の制約を付与する表現形式を補完的に用いて多面的に仕様記述を行なえる環境を提供することを意図している。

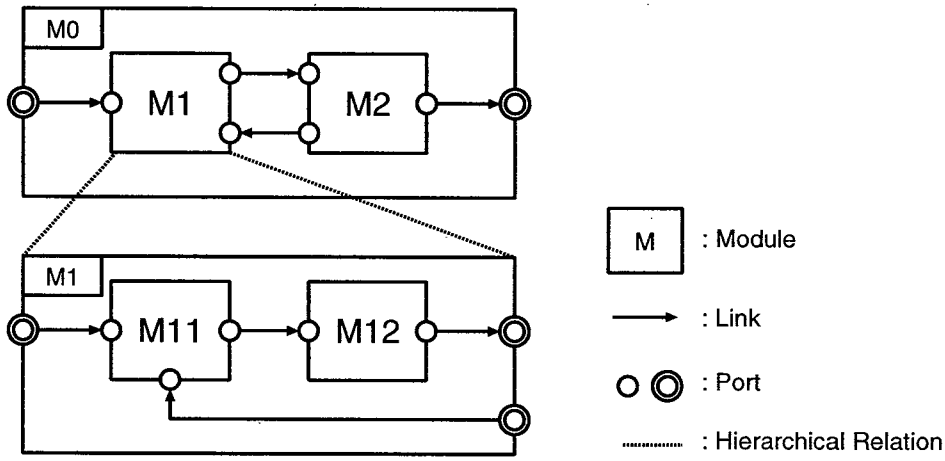


図 3.2 機能ブロック図の構成要素

機能ブロック図は設計仕様水準では最終的に抽象データ駆動プログラムに帰着できることが期待される。一方、要求分析段階の支援が可能な仕様記述環境を構築する観点からは、仕様記述の当初からデータ駆動実行が可能なモジュールの記述を強要しないよう配慮する必要がある。このため、機能ブロック図は、解釈に自由度を付与した半形式的な表現形式として捉えられている。

モジュール モジュールの本来の意味は『塊』であり、情報処理の分野では、独自の機能を持つ交換可能な構成要素と捉えられるのが一般的である。仕様記述環境におけるモジュールは、トップダウン的に仕様化する場合には、機能分割の対象となる概念であり、ある意味的にまとまった機能単位として考えられる。一方、ボトムアップ的に既存の部品を組み合わせて仕様化する場合には、部品はある程度洗練化されたモジュールであり、部品を組み合わせたものもまた、独自の機能を持つモジュールとなる。このように、どの階層でも、あるモジュールは、いくつかの別のモジュールから構成される。

このように用いられるモジュールは、ブラックボックスとして取り扱えるよう、そ

の内部の実現法に依存せずに機能的に規定されるべきである。従って、その機能が外界から受ける影響の原因と外界にもたらす影響の原因が観測可能なよう、外界とのインタフェースが明示的になっていなければならない。このため、多面的なシステム仕様記述体系の機能ブロック図では、モジュールの入出力端子が明示される。(ただし、後述するように、モジュールが独自の状態を持つ場合には、端子を明示するだけでは不十分である。)

また、他の表現形式の使用を示唆する情報や変換に本質的な情報を積極的に獲得する観点から、モジュールの属性を明示できる。詳細は後述するが、例えば、仕様化する対象と外界との関係を明示するために、外部モジュールという属性を設けている。当然のことながら、モジュールの属性の明示は了解性の向上に寄与する。

端子 上述のモジュールに付随する端子は、外界との接点（インタフェース）となるため、極めて重要な構成要素である。この端子に対して、多様な入出力データ（セット）を対応づけられることは言うまでもない。これがハードウェア・モジュールの端子と最も異なる点である。ハードウェア端子には常に真理値（0または1）の列が入出力されるために部品化やその再利用が容易になっていると考えられる。したがって、ソフトウェア端子においても、その入出力データ（セット）が明確に定義されていれば、部品化や部品の再利用が容易になると推測できる。このようにソフトウェア端子を捉えれば、ハードウェアを含めて、統合的にシステムの機能仕様が定義可能になる。

この端子は、データ（セット）の入出力点であるため、当然のことながら、階層的に定義できなければならない。たとえば、論理回路図において、あるモジュールのすべての入出力端子を最上位のモジュール定義中に記述することは原理的に不可能であるからである。

リンク モジュール間を接続するリンクは、そのリンクを介して何らかのデータ（セット）が授受されることを表すものである。すなわち、リンクはモジュール間のデータ依存関係を抽出するための手がかりになる情報である。

ソフトウェア・モジュール間のリンクは、端子の項でも述べたように、一般には、階層的に定義できなければならない。また、情報の流れを抽象化したものであるので、記述の容易さや了解性の向上のためには、リンクの分枝や合枝も記述できるべきである。

階層 人間が直観的に判断可能なマジカル数 [48] は高々 7 ± 2 であることを考慮すれば、記述の階層化は必須である。機能ブロック図による階層的記述には、モジュールの階層とそれに伴う端子やリンクの階層が含まれる。モジュールは、意味的にまとまりを持つモジュール群を一つの上位階層モジュールとして捉えることによって、階層化される。端子についても、それに対応したデータ (セット) の意味的なまとまりを基準に階層化される。

仕様記述の対象となる実システムの一般的なモデルは、イベント（あるいは、データ）駆動型に動作する（状態遷移）モジュール群とこれらのモジュールから参照・更新される構造体データを処理するモジュール群からなると捉えるのが自然である。このようなシステムモデリング手法に照らして、仕様記述対象モジュールがどのような役割を持つかを明確にするため、AESOP の機能ブロック図では、モジュールおよびリンクの属性 (役割) を明示的に表現できるアイコンを次のように定義している。

(b) 半形式的な図的記述の拡張

前述の機能ブロック図に関する解釈に準じて、AESOP では、機能と機能間の接続関係を表す一般的な機能ブロック図 FBD や、機能の入出力の時間的関係を表す一般的なシーケンス図 SC に対して、機能モジュールやデータフローの属性を明示的に表現するために、**表 3.1**および**表 3.2**に示す解釈を付与した図的記述要素 (アイコン) が新たに導入されている。これによって、初期の要求分析時には、**表 3.1**の第一行目に示す、一般的なアイコンを用いて定義されたモジュールが、設計が進行するに従って、第二行目以降のアイコン、すなわち、外界、選択分岐、状態、あるいは、ファイルといった属性を持つモジュールとして確定可能なようになっている。リンクについても同様であり、一般的な構造体データ、制御情報、および、ファイルアクセス情報のそれぞれの情報の授受を表現するアイコンが提供される。一方、第二列目は同一の機能であるが異なる状態値やファイルを有している複数のモジュールが存在することを表現するためのアイコンである。これは、後述する状態遷移プロセスが複数動作するシステムの記述などに適用される。以上の半形式的な図的表現の拡張によって、要求定義水準から詳細仕様水準に至るまで同一の図的表現が一貫して利用可能になる。

さらに、ユーザにとって最も身近にあるユーザインタフェースには快適な操作性・了解

表3.1 FBD・SCにおけるモジュールの図的表現と解釈

モジュールの種類	多重実行	解釈と 対応表現形式
一般形		一般的な処理 FBD
外界		仕様化対象外処理
選択分岐	(△)	選択処理 DT/FBD
状態	—	状態を伴う処理 STD
ファイル	—	データ抽象 DBD

表3.2 FBD・SCにおけるリンクの図的表現と解釈

リンクの種類	解釈
→	一般的な構造体データ
⇢	制御情報
⇨	ファイルアクセス

性が求められ、システム開発での役割も大きいことに着目して、AESOPではユーザインタフェースの図的表現UID(User Interface Design Diagram)が導入されている。ユーザインタフェースの図的記述は、システムの内部構成に携わらないユーザでもユーザインタフェースへの要求を素直に定義できると同時に、プロトタイピング結果の確認に有効である[49]。このようなユーザインタフェースへの要求定義には、実現法の生成に必要な本質的な情報が多く含まれている[50]。すなわち、主として、システムへの外部入出力に関す

る情報、および、システム内部で記憶されるべきデータの情報に関する情報が含まれている。**図 3.3**は、AESOP に導入されている UID 記述の例であり、リフト制御問題 (第 5 章参照) のユーザインタフェースを記述した例である。この記述では、リフトの台数とビルの階が二次元の表形式で表現され、各リフトの属性や各階の属性がポップアップ・ウィンドウにより表現されている。また、特定のリフトの特定階に関する情報も個別に表現されている。すなわち、UID 記述においては、マトリクス形式を基本とした表現から、システム内部で記憶されるデータ情報として、構成要素の属性定義から、構成要素がもつデータの型・サイズ・次元・定義域、およびインスタンスの情報が抽出される。さらに、外部入出力データの情報として、構成要素に起こり得るイベントの定義から、システムへの外部入出力データの構造の情報や機能構造情報中の外部入出力ポートとの関係が抽出される [51]。

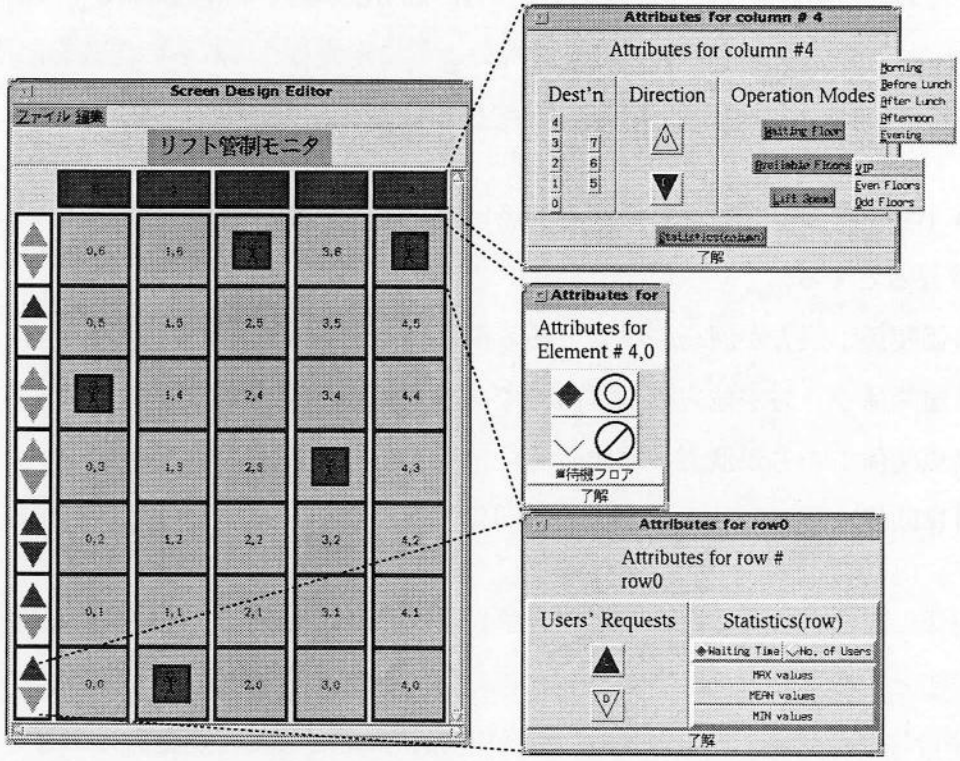


図 3.3 ユーザインタフェースの図的記述例

(c). 形式的な図的表現の補完的な導入

システムの詳細仕様水準では、半形式的な図的記述の拡張のみでは完全な詳細定義は困難である。それゆえ、データ構造を表すデータ構造図 DBD、状態遷移論理を表す状態遷移図 STD、および、選択的処理を表す決定表 DT の形式的な表現形式を導入し、これらを用いた記述を半形式的な仕様記述を補完する形態で適切に付加し、実行可能なプログラム仕様記述へと洗練する手法が採用されている (**表 3.1**の第三列参照)。また、これらの図的表現形式の理論的背景には、たとえば、関係代数、オートマトン理論、述語論理などが導入でき、次章に述べるトークンフロー水準の検証に比べて、仕様記述の意味により近い水準で、種々の形式的記述の検証手法 [52] を将来的に適用できる可能性もある。

(d) 相互関係の明示

多面的かつ階層的に定義される図的仕様記述には、複数の記述間で互いに重畳して定義される情報が存在するので、これらの重複情報の首尾一貫性を維持することが重要である。このため、利用者から捉えた場合の図的仕様記述要素間の相互関係として、等価関係、階層関係、従属関係、および対等関係を次のように定義し、これらの定義を記述者が陽に行なうか、あるいは、次章に述べる相互変換機能により生成する手法が採られている。

定義 3.1 (図的仕様記述要素間の相互関係) 図的仕様記述の要素間の相互関係は次のように定義されるとする。

等価関係：双方が同一の要素である関係
階層関係：一方が他方の上位階層である関係
従属関係：一方が他方に属する主従関係
対等関係：相互に独立な一対一の関係 □

等価関係にある場合には、一方が削除されれば、他方も削除される。従属関係にある場合には、主となる要素が削除されたときのみ、従となる他方の要素が削除される。階層関係および対等関係にある場合には、一方が削除されたとしても、他方は影響を受けない。たとえば、ユーザインタフェースの図的表現 UID では、仕様化対象システムの外部データに関する要求が主として定義され、そして、FBD 上のリンクとは従属関係、データ構造図 DBD とは等価関係として、相互に関連づけられる。

(e) 状態を伴う処理の多面的な図的システム仕様記述の例

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。状態遷移処理は、入力 (系列) と現状態に応じて副アルゴリズムを選択する処理であり、プロセス内に閉じた局所的な履歴依存性を持つ。このような状態機械の外部仕様の記述には、従来、シーケンス図や状態遷移図が主に用いられてきた。しかしながら、それらの表現形式はあくまでもドキュメントとして機能させるために、用いられてきたものであり、機械的に解釈可能なようには考えられていない。また、内部仕様水準の記述には、仕様の抜けを検出する意味から、網羅的に動作を定義できる遷移表あるいは決定表が主に用いられてきた。しかしながら、内部仕様の水準になると、逐次代入型処理方式の制約を受けた設計仕様として定義せざるを得ないため、大域変数を伴った不必要に複雑な仕様記述になってしまう傾向がある。

システムをモデル化する際には、特定の処理モデルに依存しない水準で、対象とする処理の形態を自然に捉えることが重要である。状態を伴う処理の場合、状態が必要な箇所に対して可能な限り独立なサブオートマトンに分解し、状態数の少ない機能の組み合わせによりシステムをモデル化すること、さらに、サブオートマトン間で授受する情報を可能な限り極少化して各サブオートマトンの独立性を高くすること、が必要である。これによって、モジュラリティが高く、見通しの良い、システムとしてモデル化でき、各サブオートマトンの再利用も容易になる。したがって、AESOP の仕様記述手法では、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重要視されている。そして、半形式的な図的要求仕様記述を補完する形態で、これらの記述要素を簡潔かつ明示的に表現できる記法が導入されている。

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としてデータブロック図 DBD を採用した。これによって、部分的に異なる副データ構造を有するイベント群を抽象化・統合化した、一つの複合イベントが表現可能になる。

状態分割のための表記 ：上述の状態モジュールの振舞いを状態遷移図 STD により定義し、さらに、これらの詳細な振舞いを階層的に 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。

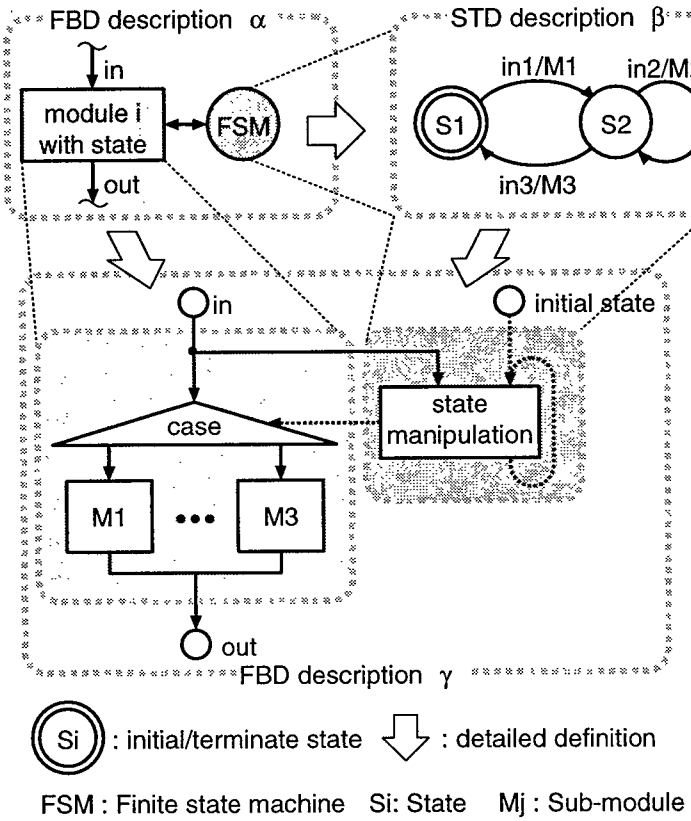


図 3.4 状態遷移処理の多面的図的仕様記述手法

これらの仕様記述の表記法を用いれば、**図 3.4**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。**図 3.4**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、**図 3.4**右上の 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、**図 3.4**下の 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、次項に述べる、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。

3.3.2 拡張した抽象データ駆動型処理モデル

AESOP では、部品化とプロトタイピングとによって、システムの生産性・保守性の向上を意図している。特に、部品化を容易にするには、入出力データ構造と発火規則に厳密なデータ駆動原理を用いた処理モデルの有効性が既に実用水準で確認されていることは前章にふれた通りである。しかし、純粋なデータ駆動原理だけでは、履歴依存性を含む、一般的な処理をすべて包含できない。これまでにも、この履歴依存性を許すデータ駆動言語として、図的データ駆動言語 D^3L[53, 54] やデータフローマシン用関数型言語 Valid[55] などが提案されているが、いずれも、仕様に含まれる多様な履歴依存性のセマンティクスを表現する水準まで規定されていない。

これに対して AESOP では、抽象機械上での実行可能プログラムを、アルゴリズム部分とデータ構造部分に明示的に分離し、従来のプログラミング言語では失われがちな、データ構造に関する情報をなるべく仕様記述水準から抽出して保存することが意図されている。すなわち、いわゆる履歴依存処理を、処理アルゴリズムの履歴依存性を示す状態遷移処理と、そのアルゴリズムの処理の対象となるデータ構造の履歴依存性とに区別してこの問題が解決されている。

この方法によれば、明確なデータ依存性の表現とデータ駆動型発火原理を持つモジュールの集合としてプログラムが表現されるので、部品化が容易なだけではなく、上述の意味で、拡張されたデータ駆動型プログラムであるために、トークンの生成・消費に関しては、厳密な検証性を持つ必要がある。

制御フロー型プログラムでは、文の間のデータの受渡しだけではなく、履歴依存性のある処理を扱うためにも、すべて変数を介した代入文が用いられている。しかしながら、この代入文が生じる深刻な副作用はこの形式のプログラムを部品化する上で大きな障害の一つとなっている。AESOP で採用されている処理モデルでは、単純なデータの受渡しの副作用は、トークンの授受によって、完全に回避される。さらに、このモデルでは、データ駆動原理の適用によって、副作用が生じる箇所が明示される利点がある。したがって、こ

の性質を用いて、アルゴリズムが入力の系列に影響されるという意味での履歴依存性を、状態概念としてまず分離する。同時に、本質的に副作用の回避が必要な処理としての、構造体データ処理の部分は (ファイル) プロセスとして検出し、その構造体データに固有な副作用回避 (一貫性保証) 手法を導入する方法が採られている [10]。

以下では、本研究で明らかになった、前者の状態遷移処理を中心に、その解釈実行規則を述べる。

(a) 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 3.2 (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 [54]。 □

定義 3.3 (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が属するストリーム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とする。 □

定義 3.3の発火規則を有する本処理モデルは、単一のストリームに属するトークンに関するパイプライン型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属するトークン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) 複数の独立なストリームを受理して多重に処理が進行するプログラムの動作検証を行なう際に、各々のストリームに関する静的データ駆動型実行に関しては、データ駆動型プログラムのグラフ構造を、また、ストリーム間の相互作用を規定するタグ付き動的データ駆動型実行に関しては、タグ処理の首尾一貫性を、それぞれ個別に検証すればよいので、検証のための探索空間の縮退が可能になる。

(b) 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。

定義 3.4 (well-formed) 関数的なオペレータを非巡回的に結合して記述されたデータ駆動型プログラム DDP(*Data-Driven Program*) は良形 (*well-formed*) である [25]。 □

性質 3.1 良形な DDP は、内部にトークンを含まない初期状態 M_0で、全ての入力に一つのトークンを与えた時、有限の実行系列により、その入力トークンに対して一意的なトークンを全ての出力に生成した後、再び M_0に戻る。 □

性質 3.2 良形な DDP では、定義 3.3 の発火規則を有する場合でも、性質 3.1 が保証される。 □

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 (以降、FSM 構造と呼ぶ) を以下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。

定義 3.5 (FSM 構造) FSM 構造は、入力イベント Iと現状態 S_iを鍵として、予め定数として与えられる IDにより指定された出力テーブルと次状態テーブルを参照して、出力イベント Oと次状態 S_{i+1}を求める (**図 3.5**参照)。なお、初期状態 S_0を示すトークンは図中点線部に実行前に入力されているとする。また、全てのデータが 2 進符号化され、FSM 構造は論理和とテーブル参照機能のみによって構成されるとする。 □

この FSM 構造を含むデータ駆動型プログラムは、次の性質を満たす限り、安全な実行が保証される。

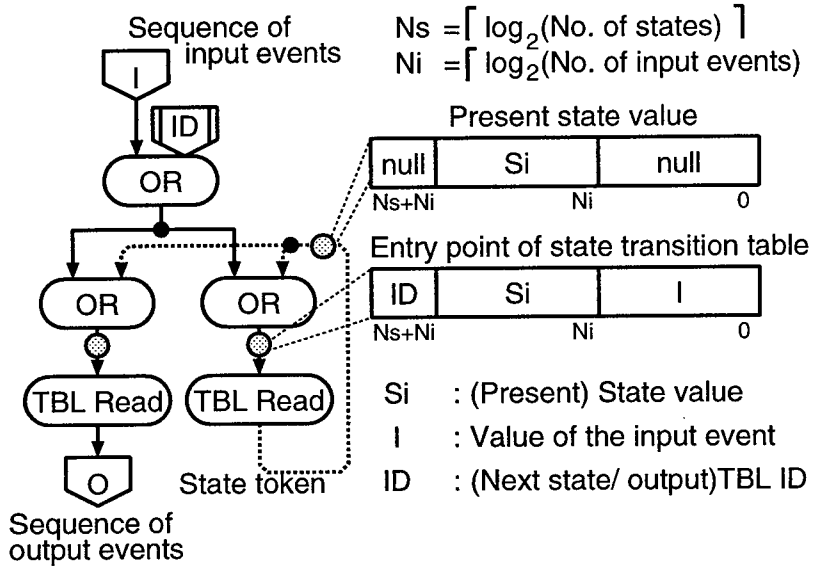


図 3.5 FSM の抽象データ駆動型プログラム構造

性質 3.3 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、FSM 構造を除くプログラム構造が非巡回グラフであれば、初期状態 M_0'で、全ての入力に一つのトークンを与えた時、有限の実行系列により、その入力トークンに対して一意的なトークンを全ての出力に生成した後、再び M_0'に戻る。 □

図 3.5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 (出力テーブルと次状態テーブル) 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) が局所化・極小化されるため、部品化、さらには、単位時間あたりの処理率の極大化も可能になる。

さらに、**図 3.6**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。逐次代入型プログラムの場合には、パイプライン型処理が原理的に不可能なことに加えて、割り込み処理やそれに伴う文脈の退避・復帰を行うためのマルチタスキング処理が付加的に必要になる問題があるが、本処理モデルではこの問題が原理的に解決される。

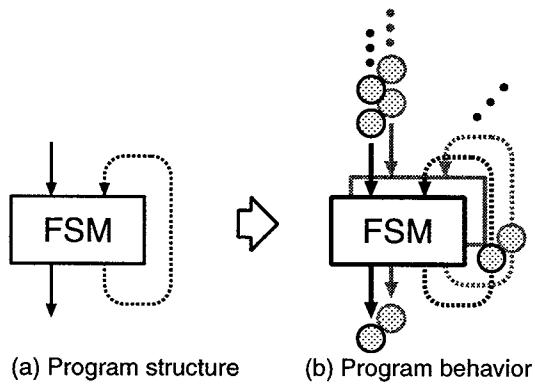


図 3.6 FSM 構造の動的データ駆動型多重実行

3.4 抽象データ駆動型プログラムの直接生成手法

本節では、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 VLSI プロセッサ Qv-x[40, 39, 17] 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。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 3.7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 3.6 (Abstract Data-Driven Program) 抽象データ駆動型プログラム ADP は、5 つ組 ADP $= (N, P, A, ST, TBL)$ で表現されるとする。

- ノード n の集合：$N = \{n_i\}$; $n_i = (F_i)$
- ポート p の集合：$P = \{p_i\}$; $p_i = (D_i, \pi_i, n_i^p)$

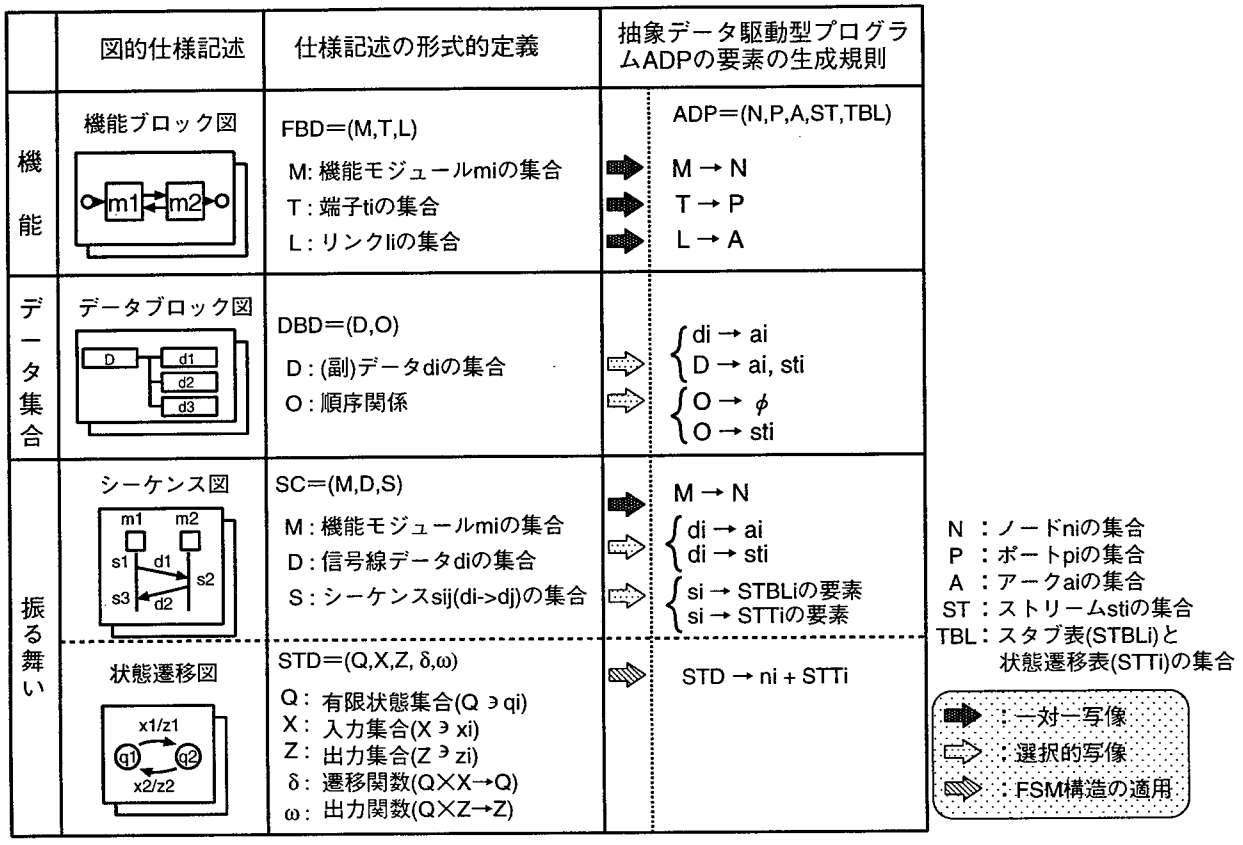


図 3.7 図的仕様記述からの抽象データ駆動型プログラム要素の生成規則

- アーク a の集合：$A = \{a_i\}$; $a_i = (p_i^s, p_i^d)$
- ストリーム st の集合：$ST = \{st_i\}$; $st_i = (IN_i, T_i, p_i^r)$
- スタブ表 $STBL$ と状態遷移表 STT の集合：$TBL = \{STBL_i\} \cup \{STT_i\}$

但し、F_i, D_i, IN_i, T_iは実現法の制約を抽象化した要素であり、それぞれ、機能、データ構造、ストリーム要素の実現値、タグを表す。また、π_i, n_i^p, p_i^s/p_i^d, p_i^rはそれぞれ、ポートの番号とそのポートが属するノード、アークの始点/終点ポート、関連するポートを表すとする。 □

システムの仕様記述に含まれる情報は大別して、機能に関わる情報、振舞いに関わる情報、データに関わる情報に分類される [56]。本変換手法では、この枠組で、以下のような変換手法を採用している。

[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 に一対一に変換可能である。ただし、機能ブロック図記述中に略記法が用いられている場合には、その解釈を行なった後に、一対一に変換される。

[変換手法 b] **データ集合の変換** ：データ集合は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 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、データ集合 Dは、永続的記憶のためのデータ構造ストリームへ直接的に変換される。

[変換手法 c] **振舞い情報の変換** ：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 ($STBL$: Stub Table) 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 3.5**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。

機能ブロック図FBDやシーケンス図SCを用いた図的仕様記述には、既に述べたように解釈に自由度を付加している。このため、不完全な情報の積み重ねにより形式的なADPを最終的に導出する過程を伴うので、対話的支援手法との融合が必要不可欠である。これに関しては、次章に詳述する。

以下では変換手法を具体的に説明するため、機能ブロック図のサブセットであり、解釈が一意的な信号流れ図SFG(Signal Flow Graph)[57] を中心とした仕様記述から加法的にADPを生成する手法について述べる。

【信号流れ図の変換手法の構成例】 信号流れ図SFGは、節点 (加算、複製) と枝 (乗算枝、遅延枝) からなる有向グラフであり、各節点の処理機能は、全ての入力点にデータが到着した時に発火する。すなわち、SFGはデータ駆動原理に基づいて解釈実行される。このため、要求仕様に含まれる同時並行性とパイプライン並列性がSFG上で自然に表現される。さらに、複数の入力信号系列に対しても、各信号系列の識別さえできれば、パイプライン型多重処理として、矛盾無くデータ駆動解釈可能である。

図 3.7に従えば、SFGの構成要素とADPの構成要素との対応関係は**図 3.8**のように規定される。すなわち、SFGの構成要素である入出力、加算/複製器、乗算/遅延枝、およびモジュールをそれぞれ外部ポート (EP)、機能ポート (FP)、機能アーク

(FA)、およびノード (N)・内部ポート (IP) として捉えることによって、図中の下線付きの ADP のキー要素に一対一に対応させる。ここで遅延枝はデータ駆動型処理モデルに特有の delay 命令、すなわち、タグ (T) にある値 n を加算する命令に置換される。以上の対応関係を用いると、たとえば、ユーザが SFG 上で FA を削除した場合、対応する ADP の**N**が削除される。このとき構文規則から P,A も同時に削除される。これと同様に SFG の構成要素の追加、値の更新についても、N,P,A の構文検査により、一対一に ADP を変更できる。同様に、ADP のデータ構造情報 (D, ST) を生成するために、**表 3.3**のような写像関係を用いる。

SFGの構成要素		ADPの構成要素	
入出力 ◎	EP	◎ 外部ポート	**P**
加算／複製器	FP	① ③ + ② ④ ノード	P*,**N**,P*
乗算枝 n	FA	n 定数 ② ポート ① × ③	A,P,**N**,P,A
遅延枝 z^{-1}		0 初期トークン ① delay ②	
モジュール ① F ②	N	① F ②	**N**
	IP	対応ブロック	**P**

※ P*は、NAまたはFAがFPに接続するのに応じて追加

図 3.8　SFG の図的構成要素と対応する ADP 要素

このように加法的に生成された ADP を、たとえば、データ駆動型プロセッサ Qv-x 上でソフトウェア実現する場合には、入力ストリーム (ST) と予め用意する部品 (F) に関する情報をプロセッサの制約を考慮して付加する。これによって、$ADP = (N, P, A, ST)$ の 4 つ組に対応する情報を置換でき、以下のように、Qv-x プログラムが生成できる。

まず、ADP 上で抽象化された部品 (F) と、トークンの値 (IN)・識別子 (T) を、データ型 (D) を元にして、実機械の機能的制約を加味して変換する。この時、予め

表 3.3 SFG 構成要素と ADP のデータ情報の対応

構成要素	データ情報	ADP
EP	データ型	D
EP	サンプリング周期	D
FA	定数	IN,D

最適化した部品を用意しておけば、ADP から Qv-x プログラムにほぼ一対一に変換できる。たとえば、信号処理向きデータ駆動プロセッサ Qv-s は、12bit 固定小数点数のパケット形式に対する加算・乗算に加えて、画像信号処理専用の複合的なプリミティブ命令 (F) の実行が可能である [17]。このため、12bit 内の精度のデータとその処理機能は、一対一に変換できる。一方、これ以外のデータ構造 (D) に対しては、これに適合した部品を用意し、トークン (IN,T) をこの部品に合わせて、再編成する。

さらに、実際には、入力信号の数、各信号の標本化レート、プログラムの同時並行処理性が資源制約を越えないように最適化を施し、プロセッサの処理能力を最大限に発揮させる必要がある。この最適化手法については別途検討を進めている [16]。

以上の変換手法を簡単なフィルタに適用し、Qv-s 上で実行可能なプログラムを生成した例を**図 3.9**に示す。このフィルタは、FM 変調時に S/N 比を平均化する目的で利用されるディエンファシスフィルタである。この図から容易にわかるように、仕様記述の構造が最終的なオブジェクトプログラムの水準まで保存されていると同時に、多重処理に関する付加的なライブラリが一切追加されていない。それにも関わらず、本 Qv-s 用プログラムによって、複数の信号系列を同時に受理して多重にフィルタ処理を実行することが可能になる。

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、加法的 (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、それ以前のプログラム生成系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

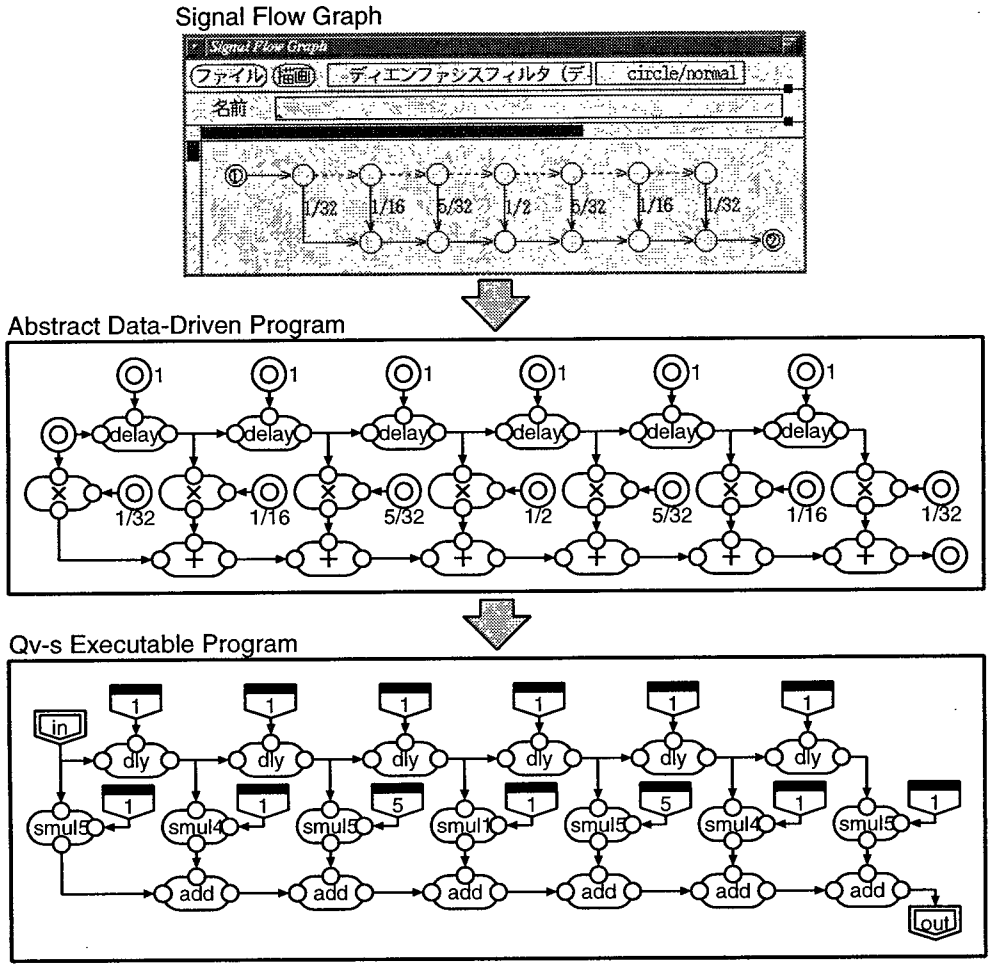


図 3.9　ディエンファシスフィルタの SFG 記述からのデータ駆動型プログラムの生成例

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される。

3.5　結言

本章では、図的援助を用いて半形式的に定義されたシステムの要求仕様に、形式的な図的設計仕様記述を漸次付加して詳細化し、最終的にソフトウェア/ハードウェアとして直接実現可能な抽象データ駆動プログラム記述を生成する体系について述べた。まず、本システム記述の体系についてその概要を述べ、次に、半形式的な図的表現によるシステム仕様記述手法、ならびに、履歴依存処理のうち状態遷移処理を副作用なく規定できる抽象データ駆動型処理モデルを提案した。最後に、多面的な図的システム仕様記述から抽象

データ駆動型プログラムを加法的 (インクリメンタル) に生成する手法を示した。これらの手法の提案によって、実現法に依存しない水準でのシステム記述の継承・保守を可能とする基盤が基本的に与えられた。

今後の課題としては、よりユーザに親しみやすいシステム記述とするための図的表現法に関する検討、ならびに、構造体データ処理を含む一般的な抽象データ駆動型処理モデルの定式化が残されている。前者に関しては、一般に、仕様化の対象となるシステムの初期仕様や機能拡張・変更の仕様を決定する要求分析の際には、その時点に要求されるシステム仕様のみならず、そのシステムが将来に渡り安定して稼働・進化し続けられるように、将来の拡張を予測・考慮した後に、最終仕様が決定されることを考慮する必要がある。この時、考えられる仕様を列挙する、発散的な思考が要求されると同時に、その時点の仕様を確定するための収束的な思考も要求される。したがって、親和図などの図形思考法 [47] の概念を取り入れて、たとえば、必要な機能の列挙が可能な機能ブロック図、ならびに、必要なデータ (集合) の列挙が可能なデータ構造図をブレーン・ストーミングの経過を表現可能なように拡張すること、および、システム外部から見た可観測な振舞いの列挙が可能なようにシーケンス図を拡張することが考えられる。

後者の抽象データ駆動型処理モデルにおける構造体データ処理構造に関しても、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。その実現法に関しては、これまでにも、多様なアーキテクチャが提案されている [58]。現在プロセッサアーキテクチャを含めて検討中である。具体的には、機能的なメモリを有するストリーム型データ駆動プロセッサ・アーキテクチャを提案し、大量のメモリを必要とする計算物理や画像処理の領域では有効であることを確認している [59, 60]。

第4章

システム仕様記述過程の対話的な支援手法

4.1 緒言

データ駆動パラダイムは、データ駆動型プログラムの検証性によって、システム記述の検証、部品化、ならびに、部分的プロトタイピングが原理的に安全に実現される特徴を有している。さらに、前章に述べた加法的 (インクリメンタル) な変換手法によって、仕様記述が少しでも修正・変更されれば、即座に、その影響を確認する対話的な環境を提供することができる。したがって、AESOP では、これらの特徴を活用して、システム記述の機能に関する有効性確認 (validation)・検証 (verification)[61] を対話的に支援する手法、ならびに、ソフトウェア/ハードウェア実現法の選択のために必須となる対話的な性能評価支援手法を導入している。

第一の目的に対しては、AESOP に導入した数種の半形式的な図的表現を活用する。すなわち、これらの記述手段は、対象とする問題をそれぞれ異なる側面から表現し、個々の表現形式に固有な情報と複数の表現形式に共通な情報を含んでいる。したがって、表現間に共通な情報を手がかりとして、表現の一貫性がある程度確認できるので、相互補間による仕様情報の獲得が促進される。同時に、論理的検証や、プロトタイピング実行の結果を複数の図的表現に視覚的に反映できるため、ユーザが頭の中に描いている仕様イメージと実際に記述された仕様との間の整合をとる、有効性確認もまた促進される。

第二の目的に対しては、システムの実現法を選択する際に、システムへの要求が満足できるかどうかは、当面の要求と将来の拡張性を含めてスループットの確保が第一義的に重要であると考える。ハードウェア論理水準にまで詳細化されたシステム仕様記述に関しては、我々の研究室で提案している自己タイミング型パイプライン [62] ならびにνMOS[63] などの高機能素子を用いれば、最大限のスループットを達成できる自律分散型 ULSI シス

テムが実現される。しかし一般には、コスト性能比の向上が要求されるため、汎用的なプログラム処理と、専用的なハードウェア論理とが有機的に協調し、システムに要求される処理能力を実現せねばならない。このため、システム仕様の中で、スループットへの影響がクリティカルにならない部分は、その機能の共通部分を抽出し、汎用的なプロセッサに置き換え、スループットを制約することなく、ハードウェアの高い稼働率を維持する設計を可能にするための支援環境の構築が必要である。

以下本章では、前章に述べた抽象データ駆動型プログラム (ADP) の生成手法を基礎にすれば、多面的に定義されたシステム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。さらに、システム記述のうち、ソフトウェア実現部分に関する性能評価支援手法として、データ駆動型プログラムの性能予測手法ならびに視覚的性能評価支援手法を提案する。

4.2 図的仕様記述相互間の変換による対話的支援手法

AESOP では、仕様記述から ADP の生成に必要な情報を積極的に獲得して、部分的にでもプロトタイピング実行できる環境を意図している。しかし、ブロック図表現は、前章に述べたように、多様な解釈が可能である。このため、階層的なブロック図表現に加えて、振舞いやデータ構造などの補助的な情報の定義を促すように問い合わせるなどの、対話的な環境を提供する必要がある。本節では、データ駆動解釈に必要な振舞いとデータ構造の情報を明らかにし、その対話的な獲得手法を述べる。さらに、振舞いが確定した段階で、副作用のない ADP の生成を促すための検証手法について述べる。

4.2.1 相互変換手法

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対話的に誘導する支援機能がユーザフレンドリな環境の実現には必須である。

前章に述べた ADP 要素の生成規則の採用によって、このような対話的支援機能も容易に実現できる。すなわち、ADP 要素の生成規則の逆変換規則を用意すれば、多面的な図的仕様記述の相互間の変換が統一的に実現でき、対話的な仕様記述環境の提供が可能になる。たとえば、図的仕様記述 α と β が同一の対象を異なる側面から相互補完的に定義している場合、$\alpha \rightarrow$ ADP $\rightarrow \beta$、あるいは、$\beta \rightarrow$ ADP $\rightarrow \alpha$ へと ADP を介して相互に変

換可能なため、重複情報の矛盾の指摘や、補完情報の定義の誘導が可能になる。

一般に機能ブロック図中のモジュールは、イベント駆動とデータ駆動、要求駆動の3通りに解釈できる。すなわち、複数の入出力ポートを持つモジュールは、入出力間に**図 4.1** のように様々な因果関係があると解釈される。これを一意に解釈するには、シーケンス図などによりモジュールの振舞いが直接定義されるか、あるいは、下位階層の機能構造の探索によって部品間のデータ依存関係から入出力の因果関係を導出する必要がある。

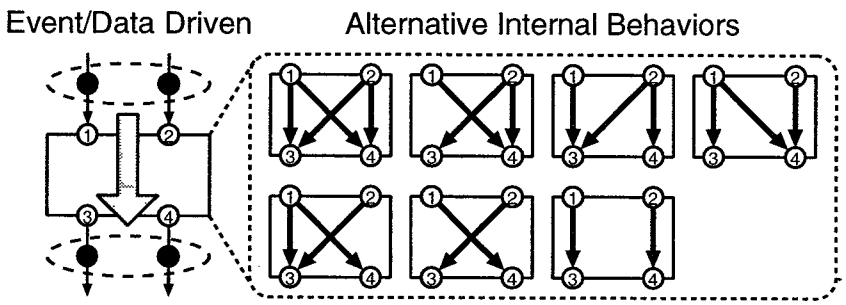


図 4.1 モジュールの振舞いの多様性

このような状況を考慮して、AESOP 環境では、適切なアドバイスあるいは問合せを利用者に提示する観点から、

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ている。これによって、多面的な記述相互間の矛盾の指摘や見落としの防止などの間接的な援助も可能になる。

この仕様記述の解釈実行の基盤として、各種の図的表現から抽出する仕様情報と ADP の両者を包含する中間形式 IF(Intermediate Form) と呼ぶ、情報を用意する。

定義 4.1 (中間形式) 中間形式 (IF) は、仕様階層毎に ADP の5つ組に因果関係を加えた6つ組 (N, P, A, ST, TBL, IOE) からなるとする。

因果関係 IO Event の集合：$IOE = \{ioe_i\}$; $ioe_i = (p_i^d, p_i^s)$

但し、Nは階層的に定義される。 □

図 4.2はこの中間形式の構成要素間の相互関係を図示したものである。仕様記述の各階層毎に、この相互関係にしたがって、仕様記述に含まれる本質的な情報を統合する。図中の矢線は、各要素間の一対一関係ないしは一対多関係を表しており、前章に述べた図的仕様記述要素間の相互関係に相当する。図中の網掛け部分はぞれぞれ、前章に述べた変換手法 a,b, および c により取り扱われる機能、データ、振舞いに関する情報の領域を示している。これらの領域の重複部分に含まれる要素が図的表現形式間に共通に含まれる情報となり、この情報の追加・削除・変更が相互変換の契機となる [64]。

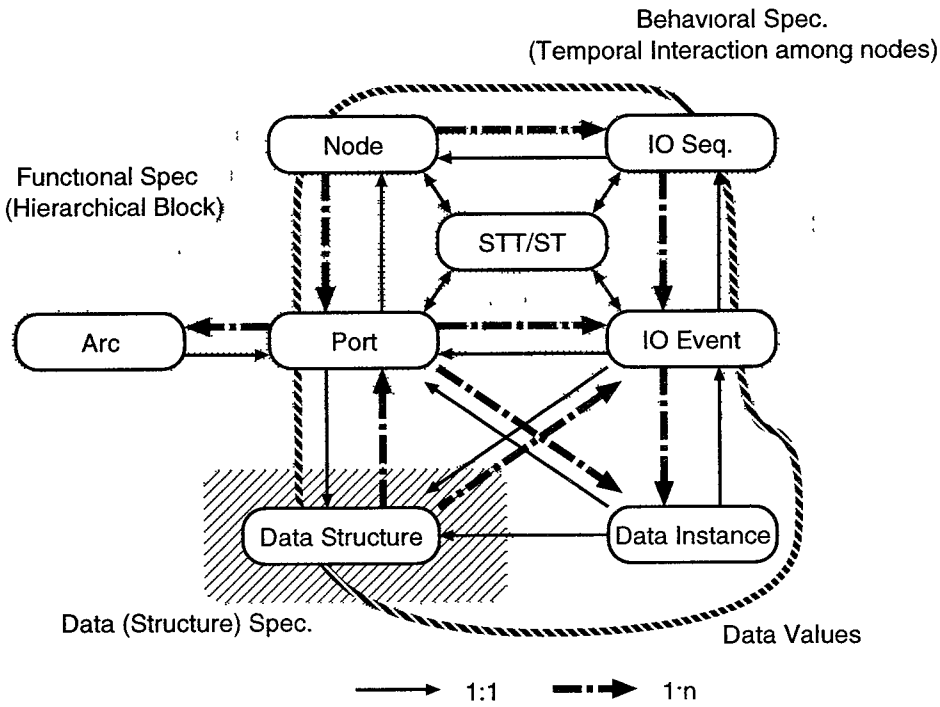


図 4.2 中間形式の各要素間の対応関係

この中間形式を対象として、本変換系では、さらに積極的なプロトタイピングを可能にするため、まだ定義されていない補助的な情報に妥当な暗黙値を用意して、仮に解釈実行を試みる。その結果、ユーザの同意が得られれば、その時点で仕様を確定できる。この暗黙値としては、たとえば、信号処理分野を対象とする場合には、振舞いに関しては、**図 4.1**左上のデータ駆動型の因果関係を、データ構造に関しては、固定小数点 (整数) データを採用すれば効果的であることが判っている [13]。

具体的には、**図 4.2**に示す中間形式情報として階層的に統合することによって、以下に

示す情報を検出して、追加・修正を施された箇所に対して**図 3.7**の生成規則を逆に適用する。

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)。

データ ： データと機能に関する情報との対応関係の未定義・矛盾箇所。

振舞い ： 断片的な振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。

その結果、異なる表現形式の相互間での変換が実現され、利用者の意図が早期に仕様記述上に反映されるように積極的に利用者に働きかける環境が提供可能になる。

4.2.2 中間形式の論理検証手法

構造的に副作用のないQv-xプログラムの生成のために、ストリームに関する *well-behaved* 性を規定できる ADP の性質を活用して、上述の中間形式の各階層毎に、分割統治型に、トークンの可達性の検査を行う [65]。本検証では、ストリームの要素が増減する選択的処理などを隠蔽した *well-behaved*[25] なモジュールを前提として、その因果関係に従った依存グラフの探索により、トークンの可達性を検査する。

(a) 中間形式 IF の検証用グラフ

本手法では、トークンの可達性を調べる操作を単純化するため、依存グラフ構造と因果関係を統一的に扱う検証用グラフを導入している。定義 3 より ADP のノードにおいては、入力ポートに到着した一つのトークンが、複製や抹消されることなく、因果関係のある出力ポートに送出される。この因果関係はアーク集合と等価である。よって、IF のグラフを**図 4.3**のように (P, A) の 2 つ組で構成される、トークンフロー検証用グラフ TVG (Token-flow Verification Graph) に変換する。またこの TVG は、IF を介して、**図 4.3**の点線の枠内に示すように、仕様記述の編集操作に対して加法的に生成できる。このため、逆変換により検証結果を仕様記述上へ直接反映することが可能になる。

(b)TVG の検証アルゴリズム

TVG 上でのトークンの可達性検証のために、トークンの到達可能部分を以下のように定義する。

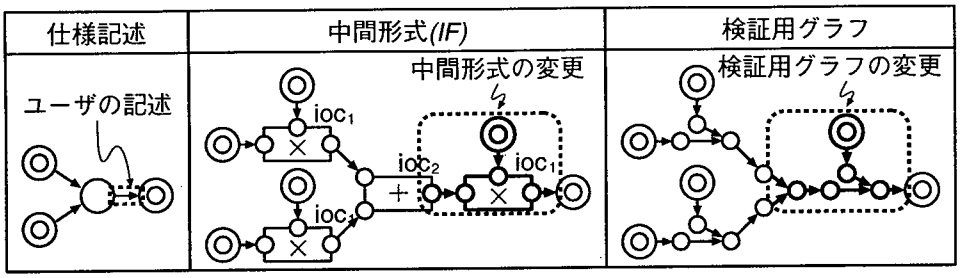


図 4.3 トークンフロー検証用グラフ

定義 4.2 (トークンの到達可能部分) 外部入力以外で入力アークが未定義であるポートと、初期トークンを除去したTVGの全ての閉路には、トークンが到達できない。これらの2種類の箇所を始点とする部分グラフには、トークンの到達が不可能である。よって、これ以外がトークンの到達可能部分である。 □

このTVGの探索を局所化するために、加法的に変更されるIFの情報を利用する。そして、変更部分の影響する範囲のみを探索する。具体的には、有向グラフの強連結成分の抽出手法を応用し、各ポートに検証結果を付加してこれを伝播する手法を採る。

本検証に導入したTVGは、IFの入力ポートが一本のアークのみに接続するため、自己閉路と多重枝を持たない単純グラフ (Simple Graph) である。単純グラフの探索には一般に、節点数と枝数の和に比例する計算量が必要であるが、本検証手法では、探索範囲を一階層内に限定できる。従って、仕様記述の了解性の指標となるマジカル数 (7±2 quanta) を考慮すれば、本手法は、十分実用的な時間で応答でき、対話的に必要な情報を獲得することが可能である。

また、この検証結果は、前述の相互変換機能によって、複数の異なる図的記述上に反映されるため、より積極的な情報獲得が可能になる。

4.3 ソフトウェア実現のための性能評価支援手法

本章の緒言にも述べたように、統合的システム記述による方法論においては、システムの実現法を選択する際に、コスト性能比を考慮した上で、仕様記述から生成された抽象データ駆動型プログラムADP中の個々のモジュールについて、ハードウェアアルゴリズムとして実現すべきか、それとも、ソフトウェアアルゴリズムとして実現すべきかを最終的には決定する必要がある。このため本研究では、システム仕様記述の中で、スループッ

トがクリティカルにならない部分は、その機能の共通部分を抽出し、汎用的なプロセッサに置き換え、スループットを制約することなく、ハードウェアの高い稼働率を維持する設計を可能にするための支援環境の提供を目指している。

本節では、その第一歩として、システム記述のうち、ソフトウェア実現部分に関する性能評価支援手法について考察している。

AESOP により抽象データ駆動型プログラム ADP をソフトウェア実現する、すなわち、数種のデータ駆動型プロセッサ Qv-x 上で高度並列に実行するには、ADP から特定の Qv-x システムの物理的構成に最適化した実行形式プログラムに変換する手法の確立が必要になる。すなわち、各 Qv-x の機能的制約 (命令セット、演算精度) や、物理的資源制約 (プロセッサ数、各種記憶容量など) を加味した最適化が要求される。また、仕様記述の記述水準、すなわち、ADP の記述水準を向上させるには、特定のプロセッサや専用ハードウェア構成に最適化した高機能かつ高性能なプログラム部品を多数蓄積し、ADP ではこれらをブラックボックス化して扱えることも重要である。このような最適化手法の検討ならびに部品の蓄積を効率良く行なうために、本研究では、現存あるいは将来に開発すべき数種の Qv-x を前提として、そのプログラムならびにアーキテクチャの開発、機能検証、ならびに性能評価を視覚化によって支援する手法を採っている。

(i) **Qv-x プログラムの機能検証**

データ駆動型プログラムでは、データ依存関係によってのみ処理実行が進行するため、ブレークポイントで、サスペンド・レジュームしたとしても、プログラムの実行結果には影響を与えないという特徴がある。すなわち、従来の逐次プロセスの並列プログラムのデバッグ手法の重大な問題である、プローブ効果 [66] が発生することは少ない。唯一非決定的な動作をするのは、データ依存関係が合流 (merge) する箇所のみである。これに関しては、静的に予め容易に検出可能であるため、この箇所にブレークポイントを設定して観察すれば、プログラムの正しさが容易に確認できる。

(ii) **Qv-x プログラムの性能評価**

AESOP における ADP の実行性能の評価に際しては、実行機械に依存しない処理アルゴリズムの実行に本質的に必要な演算のみを対象に比較検討を行なうべきである。Qv-x プログラムには、データ駆動に固有の `COPY` 命令や `SYNC` 命令、動的データ駆動に固有のタグ操作命令が含まれている。このため、実効的な命令のみを対象にした

性能 (単位時間あたりの処理率) の平均値や時間的変遷を抜粋できる必要がある。

(iii) **プロセッサの性能評価**

動的データ駆動方式を採用したQv-xは、メモリアクセスやプロセッサ間通信の遅延に対する耐性を原理的に有している。しかし、これは、プログラム構造に依存する特性でもあり、プログラム最適化の鍵の一つにもなる。このため、メモリアクセス遅延時間、機能分散形態などの種々の支配要因に対する遅延耐性を比較検討できる必要がある。また、Qv-xでは、プログラムの並列性やプロセッサ間通信量がパイプラインの容量を越えても、FIFO型緩衝記憶QB(Queue Buffer)により一定量のパケットを緩衝でき、QBがオーバフローしない限りパイプライン内のパケット流量が極大に維持される。しかし、QB内のパケット量に応じて、パケットがQB通過時間が変化するため、これがプログラム全体の応答時間性能を律速する。たとえば、遅延ループのある信号処理プログラムなどの場合には、ループ箇所の処理時間の短縮がプログラム全体のスループット向上の鍵となる。したがって、Qv-x上での効果的なプログラムの実行には、プロセッサ内のパイプライン流量をできる限り最大に維持し、かつ、QB内の平均パケット量を極小化できるように、プログラムを最適化することが重要となる。

(d) **マルチプロセッサの性能評価**

Qv-xアーキテクチャでは、自己タイミング型の緩衝記憶QB付きルータチップにより、多数のQv-xプロセッサを相互接続すれば、容易にマルチプロセッサ・システムへの拡張が可能である。このようなマルチプロセッサの性能評価のためには、各プロセッサの稼働状況やプロセッサ間通信の状況を的確に観測でき、その結果に基づきパケット流量のボトルネックを解消するための種々の最適化手法を実験的に検討できる必要がある。

以下では、筆者が属する研究室で共同開発された動的データ駆動型VLSIプロセッサQv-m(Q VLSI chip developed by Mitsubishi: 通称 RAPID)[39]を想定して、議論を進めている。まず、簡単にプロセッサの動作を以下に説明しておく。

動的データ駆動型処理方式における処理実行は、たとえば図 **4.4**に示す環状パイプライン形式で容易に実現される。入力部 (IF) に到着したオペランドパケットは、単純な連想記憶である待合せ記憶部MM(Matching Memory)内に入り、タグの内容に従って対とな

るべきパケットを待合せる。対を成したオペランドパケットは演算パケットとして出力され、演算処理部 FALU (Floating-point Arithmetic & Logic Unit) に送られる。FALU では、タグの一部に含まれている演算コードにしたがって演算処理が行われ、その結果が、オペランドパケットと同じ形式の結果パケットとして送出される。同時並行に、プログラム記憶部 PM(Program Memory) 内では、次の行き先と演算コードが読み出され、FALU からの結果パケットに付加され、ふたたびオペランドパケットとして出力される。この際、複数の行き先を持つ結果パケットは複製 (copy) され、各々の行き先と演算コードが付加され、前述の動作が繰り返される。ただし、これらのオペランドパケットの行き先がプロセッサ外部であれば、出力部 (IF) で選択的に外部へ送出される。なお、Qv-m の場合、FALU での演算の代わりに、データ記憶部 DM(Data Memory) 内のデータの参照あるいは更新を行う演算コードも用意されている。

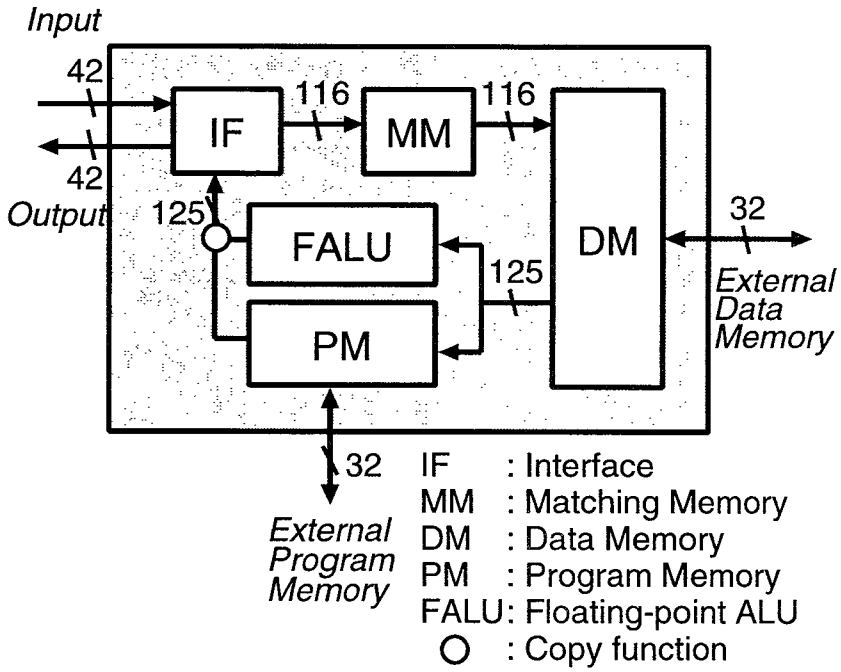


図 4.4 動的データ駆動型 VLSI プロセッサ Qv-m の基本構成

4.3.1 性能予測手法

Qv-x のパイプライン並列処理能力を限界まで引き出すには環状パイプライン内のパケット流量の変動を可能な限り平滑化する必要がある。以下では、パケット流量の変動要因に

ついて述べ、定常的に処理可能なパケット流量に関する理論的評価を行う。この理論的に見積もられた性能評価結果は、ソフトウェア/ハードウェア実現法を選択するための一つの指標として、ユーザに提供される。

(a) 同時並行処理ノード数変動とコピー発生量

データ駆動型プログラムに内在する同時並行処理性は、プログラムの同一ランクに属するノードの数 (同時並行処理ノード数) によって定まる。ここで、ノードが属するランクは、そのノードが実行される時刻の予測値として捉えられ、そのノードに到達できる全てのソースノードからそのノードへ到達する最長経路の距離と定義する。一般に、同時並行処理ノード数はランク毎に異なり、結果として、単一のプログラムを1組の入力パケット群によって実行する際のパケット流量は、この同時並行処理ノード数に応じて変動する。一方、同一のプログラムが複数の入力パケットストリームに共有される場合は、単一パケット群に関する変動が時間的にずれた形で重なり合うことになる。

プログラムの同時並行処理性の変動をプロセッサ内部のパケット流量変動という観点から見ると、待合せ記憶部 MM でオペランド対を生成するために格納されることによる流量の減少と、プログラム記憶部 PM における結果パケットの複製 (コピー) による流量の増加に起因することがわかる。すなわち2入力命令の結果パケットが単一の行き先を持つ場合に流量は減少し、n $(n=1,2)$ 入力命令の結果パケットが $n+1$ 以上の行き先を持つ場合は増加する。以下では、**図 4.4**で示された環状パイプライン構造をもつプロセッサにおける、プログラムの同時並行処理性を表す特徴パラメタならびに、このパラメタによって定まる ALU における最大パケット流量を導く。

【プログラムの特徴パラメタ】

プログラムの同時並行処理性を少数の特徴パラメタを用いて簡易に表現する。まず初めに、プログラム内に条件分岐が存在する場合には、分岐先の接続アーク数が多い側に分岐すると仮定する。また、閉路が存在する場合には、これを展開して非巡回グラフとする。このように、対象とするプログラムをプロセッサにかかる負荷がより重くなるように変形した後、以下のような特徴パラメタを定義する。

入力ソース数 N_{Input} 出力シンク数 N_{Output}

単項演算命令数	N_{1in}	二項演算命令数	N_{2in}
1行き先命令数	N_{1out}	N行き先命令数	N_{Nout}
総入力アーク数	N_{InArc}	総出力アーク数	N_{OutArc}
命令数	N_{op}	コピーの総数	$N_{SumCopy}$

これらのパラメタ間では、式 (4.1)、(4.2)、(4.3) が成り立つ。

$$N_{op} = N_{1in} + N_{2in} = \sum_{i=1}^{\infty} N_{iout} \tag{4.1}$$

$$N_{InArc} = N_{1in} + 2 * N_{2in} \tag{4.2}$$

$$N_{OutArc} = \sum_{i=1}^{\infty} N_{iout} \tag{4.3}$$

【ALU における最大パケット流量】

上記で与えられた特徴パラメタを元に ALU における最大パケット流量を導出する。まず R_{Input} を外部から入力部 IN への入力パケット流量とし、R_{Output} を出力部 OUT から外部への出力パケット流量とする。さらに、プロセッサ内部の機能部 x と yとの間のパケット流量を R_{x-y}で表す。ただし、$x, y \in \{IN, MM, ALU, PM, OUT\}$ である。たとえば R_{IN-MM}は入力部 IN と待合わせ記憶部 MM の間のパケット流量を表す。

いま、パイプラインを構成するハードウェアによって定まる最大パケット流量を R_{max} とすると、この制約から次式を得る。

$$R_{IN-MM} \leq R_{max}, \qquad R_{PM-OUT} \leq R_{max} \tag{4.4}$$

また、ALU ではパケットの生成や消滅は起こらないため、ALU における入力パケット流量と出力パケット流量は等しく、

$$R_{MM-ALU} = R_{ALU-PM} \tag{4.5}$$

となる。一方、MM では二項演算命令に対するオペランド対パケットの生成のために一部のパケットが格納され、MM からの出力パケット流量が低下する。MM へ向かうパケットが MM に格納される割合は全入力アーク数に対する二項演算命令数の比 (N_{2in}/N_{InArc}) に等しい。よって、MM からの出力パケット流量 R_{MM-ALU} は、

$$\begin{aligned} R_{MM-ALU} &= R_{IN-MM} * (1 - (N_{2in}/N_{InArc})) \\ &= R_{IN-MM} * N_{op}/N_{InArc} \end{aligned} \tag{4.6}$$

を満たす。さらに、PMでは結果パケットの連続複製によりPMからの出力パケット流量は増加する。出力パケット流量の増加の割合は全命令数に対する全出力アーク数の比に等しい。よって、

$$R_{PM-OUT} = R_{ALU-PM} * N_{OutArc}/N_{op} \tag{4.7}$$

が成立する。

式(4.4)、(4.6)ならびに式(4.4)、(4.7)より、ALUにおける入力、出力パケット流量はそれぞれ

$$\begin{aligned} R_{MM-ALU} &\leq R_{max} * N_{op}/N_{InArc} \\ R_{ALU-PM} &\leq R_{max} * N_{op}/N_{OutArc} \end{aligned} \tag{4.8}$$

を満たす。よって、式(4.5)より次式を得る。

$$R_{MM-ALU} = R_{ALU-PM} \leq R_{ALU_{max}} \tag{4.9}$$

ただし、R_{ALU_max} はALUにおける平均パケット流量の最大値であり

$$R_{ALU\_max} = R_{max} * \frac{N_{op}}{\max(N_{InArc}, N_{OutArc})} \tag{4.10}$$

で与えられる。すなわち、ALUにおける平均パケット流量の最大値は、パイプラインの物理的最大パケット流量 R_{max}、プログラムの入力アークの総数 N_{InArc}、出力アークの総数 N_{OutArc} ならびに総命令数 N_{op} によって決定される。また、式(4.8)より、$N_{InArc} \geq N_{OutArc}$ の時には、INのパイプライン合流部からMMのパケット格納部までのパイプラインがボトルネックになり、逆に、$N_{InArc} \leq N_{OutArc}$ の時には、PMのコピー部からOUTの分岐部までのパイプラインがボトルネックになることがわかる。

(b) 入・出力パケット流量の変動

Qv-xは入力パケットによって起動される受動的プロセッサである。入力パケットの投入間隔は処理内容あるいは処理要求によって定まる外的要因であり、これを制御するためには、プロセッサの外部において制御機構を導入する必要がある。

一方、結果パケットの行き先がプロセッサ外部である場合は出力部へ送られる。プロセッサ外部に何らかの資源競合がある場合、結果パケットが直ちに外部へ出力されず、プロセッサ内に留まる可能性がある。これはパイプラインのパケット流量の平滑化の阻害要因となる。

以下では、特定の単一プログラムに対するストリーム型処理を行う場合の、プログラムに依存した入力データの理論的な最大入力可能パケット流量を導く。

【理論的最大入力可能パケット流量】

プログラムを一度だけ実行するために必要な入力パケット組を入力単位としたときの、入力パケット組の流量を $R_{SetInput}$ で表すと、定義より

$$R_{SetInput} = R_{Input}/N_{Input} \tag{4.11}$$

である。1つの入力パケット組がプロセッサに投入されると、ALU にはプログラムの全命令（N_{op}個）だけの処理を要求する。一方、ALU における最大パケット流量 R_{ALU_max} は式 (4.10) で与えられるため、$R_{SetInput}$ は次式を満たさなければならない。

$$R_{SetInput} * N_{op} \leq R_{ALU\_max} \tag{4.12}$$

よって、式 (4.10)、(4.11)、(4.12) より、R_{Input} は

$$R_{Input} \leq R_{Input\_max} \tag{4.13}$$

を満たす。ただし R_{Input_max} は理論的最大入力可能パケット流量であり

$$R_{Input\_max} = R_{max} * \frac{N_{Input}}{\max(N_{InArc}, N_{OutArc})} \tag{4.14}$$

で与えられる。すなわち、理論的最大入力可能パケット流量はパイプラインの最大パケット流量 R_{max} 、プログラムの入力アーク数の総数 N_{InArc}、出力アーク数の総数 N_{OutArc} ならびに入力数 N_{Input} によって決定される。

【性能見積り手法の適用例】

図 4.5は3章の変換手法の適用例に取り上げたディエンファシス・フィルタの Qv-m プログラムである。このプログラムの特徴パラメタを次に示す。

入力ソース数	N_{Input}	： 1
総入力アーク数	N_{InArc}	： 26
総出力アーク数	N_{OutArc}	： 26
命令数	N_{op}	： 20

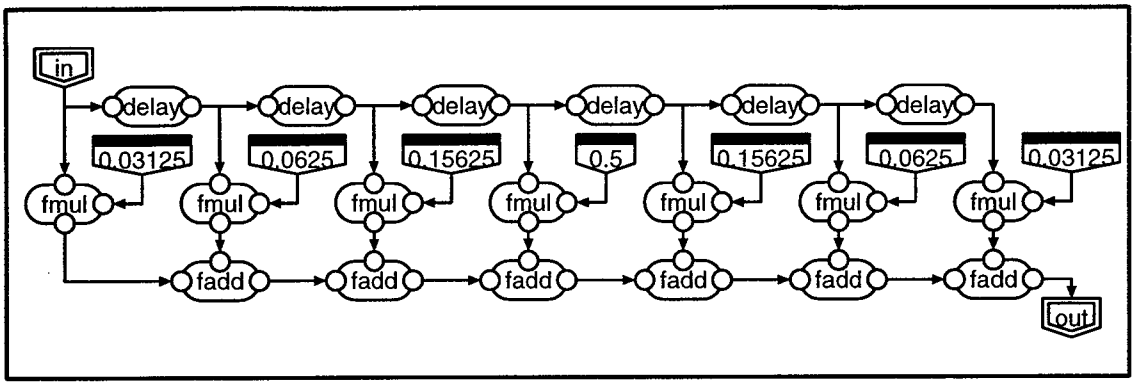


図 4.5 ディエンファシスフィルタの Qv-m プログラム

これらのパラメタから、ALU における最大パケット流と、理論的最大入力可能パケット流量は式 (4.10)、および式 (4.14) より次のように算出される。ただし、Qv-m では、R_{max} は 50M packets/sec である。

$$R_{ALU\_max} = 50 * \frac{20}{\max(26,26)} \simeq 38.5(Mpackets/sec) \tag{4.15}$$

$$R_{Input\_max} = 50 * \frac{1}{\max(26,26)} \simeq 1.92(Mpackets/sec) \tag{4.16}$$

すなわち、このプログラムを単独で Qv-m 上で実行する場合、最大 1.92MHz のサンプリング周期の入力信号を受理して処理できると見積もれる。この値は、実測値とも一致している。

本項に述べた性能見積もり手法は、パケット流量の変動を吸収する緩衝バッファQB の容量が十分に用意されている場合に成立する。システム仕様記述のソフトウェア実現法を最適化する観点からは、与えられた QB の容量の下でのプログラムの最適化に活用できる。また、ハードウェア実現を最適化する観点からは、与えられたプログラムに対して必要な QB の容量を設計するために活用可能である。

4.3.2 視覚的プロトタイピング環境

AESOP の視覚的プロトタイピング環境では、前述の各種の機能・性能評価の効果的な支援のために、次の視覚化方針を採用している。

i) Qv-x プログラム上のトークンや Qv-x プロセッサ上のパケットの多次元的振舞いを、相互に関連づけた視覚化。

ii) マジカル数 7 ± 2 を考慮した、部分的振舞いや統計情報の視覚化。

以下では、これらの視覚化方針にしたがって、特に、統合的システム記述のソフトウェア実現法であるプログラムとそのアーキテクチャの開発のための、機能検証・性能評価の支援手法を述べる。

(a) 階層的図的プログラムの作成・機能デバッグ支援

動的データ駆動型プログラムは、ノード (節点) とアーク (有向枝) から成る階層的な有向グラフにより**図 4.6**のように表現でき、また、その動作は有向グラフ上のデータ依存関係に従った多数のトークンの振舞いとして現れる。すなわち、ある瞬間には、(i) 同時並行性により同一のタグを持つトークンが複数のアーク上に存在し、(ii) パイプライン並列性により異なるタグを持つトークンが同一アーク上に存在し、次の瞬間には、それらの一部が消費され、別のアーク上に新たなトークンが生成される、といった形態で多次元的にトークンが振舞う。

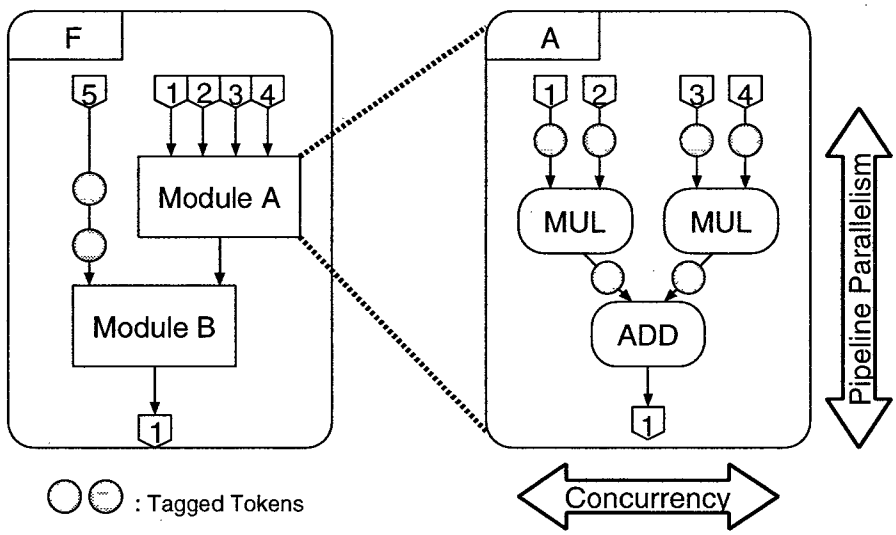


図 4.6 データ駆動型プログラムとその動作

したがって、プログラムの機能デバッグの際には、このように多次元的に振舞う多数のトークンを追跡し、バグの原因を特定し解消しなければならない。これらの作業を効果的に支援するには、多数のトークンを適切な側面から抜粋して提示することが必要である。

以上のことから、本支援手法では、プログラム上でのトークンの振舞いが直観的に把握

できるよう、図的データ駆動型プログラムとその上でのトークンフローのアニメーションを行なう機能を提供する。さらに、多次元的なトークンの振舞いを種々の側面で観測するために、以下のような切口で、部分的な視覚化機能を提供する。

機能階層 機能的にまとまった単位の階層的プログラムの編集と表示、機能階層のツリー表示

タグ タグ付きトークンのタグ値によるフィルタリング機能

処理構造 特定の処理構造 (選択構造、遅延ループ、共有関数など) の明示的表示

さらに、これらの機能階層、タグ値、および、処理構造毎に部分的にアニメーションを行なうウィンドウ上で、以下の支援機能を提供する。

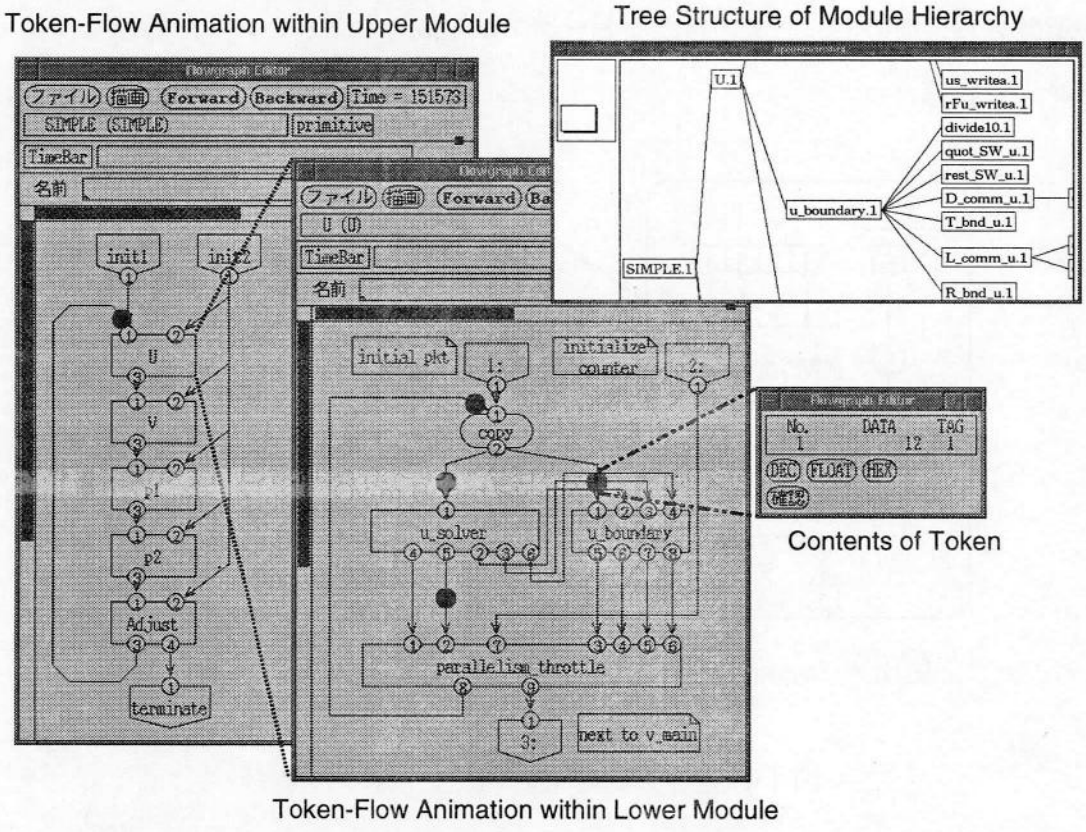


図 4.7 Qv-x の視覚的プログラミング支援環境の使用例

a) トークンの行き先ポート、データ値、および、それらに対するマスクの設定を許すブレーク機能

b) 非決定的実行の可能性がある箇所の明示機能

c) 実行履歴に基づくバックワード・アニメーション機能

d) 履歴依存データ記憶部 (DM) のデータ値の遷移状況の視覚的表示機能

図 4.7は、本視覚的プログラム支援環境を利用して、数値流体 CFD(Computational Fluid Dynamics) のシミュレーション・プログラムの作成・デバッグを行なっている様子を示している。右上の図がプログラムの機能階層のツリーの表示であり、この中の節点を選択すれば、対応する階層の実行アニメーションが行なわれる。すなわち、トークンを表す円状のアイコンが流れる様子が表示される。ここでは、プログラムのボトルネックになる箇所を直観的に把握可能にするために、アーク上に滞留するトークンの数を円の大きさに対応させて表示している。また、アニメーションを一時停止してトークンを選択すれば、そのパケットの内容が右図のようにポップアップされる。

(b) 単一プロセッサの性能評価支援

Qv-x プロセッサ内での並列性制御や待ち合わせ記憶の有効利用法などの最適な実現法の検討のためには、プロセッサ内の物理的構成がプログラムの実行性能に及ぼす影響を解析する必要がある。本評価環境が前提としているデータ駆動型プロセッサ Qv-x は基本的には、前述したように、パケットの待ち合わせ記憶部 (MM)、関数的処理部 (FP)、プログラム記憶部 (PM) からなる環状パイプライン構造からなる。さらに、パケット流量の揺らぎを緩衝するために、FIFO 型のキューバッファ (QB) が付加されている。

このパケット流量は、(i) プロセッサ外部からの入力パケットの流量によるパイプライン並列性の変動、(ii) プログラムの持つ同時並行型並列性の変動、(iii) 待ち合わせ記憶部 MM におけるハッシュ衝突の回数、に伴って、増減する。

したがって、Qv-x プログラムを最適化する際には、これらが適切に制御されているかどうかを観測する必要がある。このため本支援手法では、(i)MM の占有状況や競合中のハッシュ・アドレスに対応するトークンの探索、(ii) 入力データの投入間隔やプログラム構造を修正した時の性能 (入出力パケット流量、応答時間、QB 内のパケット量、FP の実効的な稼働率, DM アクセス率) の比較、が可能でなければならない。

一方、プロセッサ上でのパケット内の行き先ノードのみでは、プログラムの論理的構造との対応を直観的に把握することが困難である。このため、プロセッサ上でのパケットの

振舞いを階層的なデータ駆動型プログラム上での振舞いに容易に対応づけられる機能を提供する。

(c) マルチプロセッサの性能評価支援

Qv-x マルチプロセッサは各プロセッサ毎にローカルなメモリを有する分散メモリ構成となっている。本アーキテクチャ上で最適化を施すには、特定のプロセッサ間通信路ならびに特定のプロセッサがボトルネックにならないように、

- プロセッサ間の相互接続トポロジとルーティング手法
- プロセッサ間の相互接続トポロジ上へのプログラム・データの負荷・機能分散

を適切に選択しなければならない。

したがって、本支援手法では、プロセッサ間の相互接続トポロジとその上へのプログラム・データの分割・配置方法を任意に指定可能とするインタフェースを提供する。さらに、その上での実行状況を的確に把握できるようにするため、

- 各プロセッサの ALU 稼働率、QB 長、ハッシュ衝突、DM アクセス率の時間的推移
- 各プロセッサ間通信路のパケット流量 (直接通信、Global DM アクセスの比率、通過パケット)

を視覚的にアニメーション表示する機能を提供する。

図 4.8は、図 4.7に示した CFD プログラムを、トーラス状に相互接続された 4 台の Qv-m 上で実行した時の、性能評価の様子を示している。図中左上は、本環境の主操作パネルであり、中央上に、4PE のトポロジを表示し、この上で、各プロセッサの ALU 稼働率とプロセッサ間通信量をアニメーション表示している。右上には、各プロセッサの ALU 稼働率の時間的推移をグラフ表示している。また、特定のプロセッサの動作の詳細な観測が必要な場合には、図中左下に示すプロセッサ構成上で種々の性能評価が可能である。図中右下は、MM のハッシュメモリの占有状況を視覚的に表示し、特定のハッシュアドレスの内容を右下のようにポップアップウインドウ上で確認する機能が提供されている。これによって、ハッシュ衝突の原因の解析が可能になる。図中にはないが、プロセッサの各構成要素間に流れるパケット流量の時間的推移をグラフ表示することも可能である。

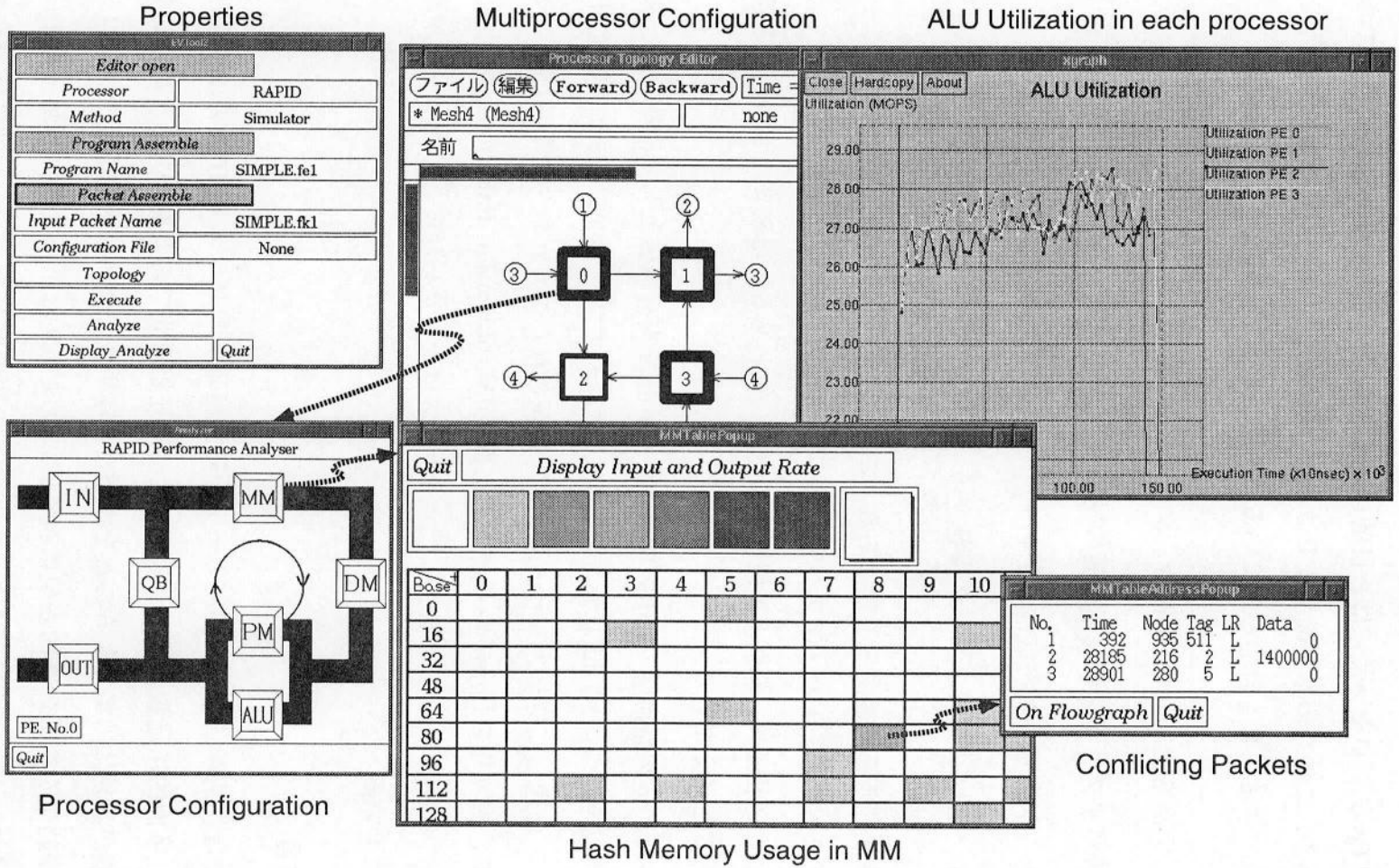


図 4.8 Qv-x マルチプロセッサの視覚的評価支援環境の使用例

(d) 新規アーキテクチャの性能評価支援

現状のQv-xプロセッサには、目的に応じて最適なアーキテクチャに発展できる余地が充分にある。このため本支援手法では、これらの最適なアーキテクチャの探索を実験的に検討可能にすることも目的としている。たとえば、

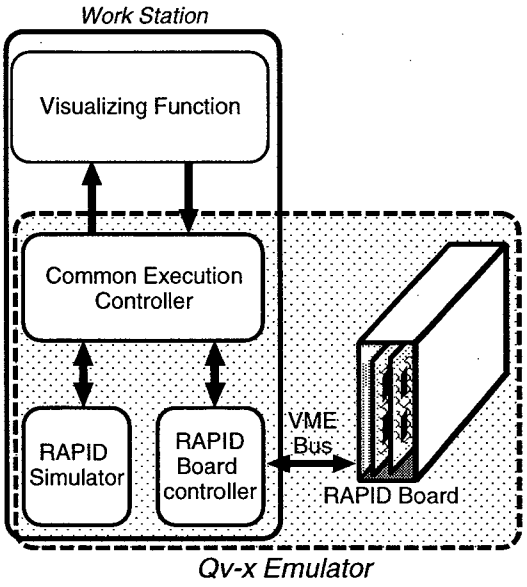
- 新しい命令セット (演算命令、タグ処理命令を含む)
- 巡回パイプラインの要素機能の構成
- パケット流量の制御機構 (活性パケットの退避、復帰機構)
- PM／DMのスワッピング機構

などの性能評価を支援する予定である。これらの支援機能の実現法としては、データ駆動型プログラムがハードウェアアルゴリズムを自然に表現できる特徴を活用して、現行のプロセッサ上で上記ハードウェア機構をエミュレートあるいはシミュレートする手法が当面有望であると考えている。

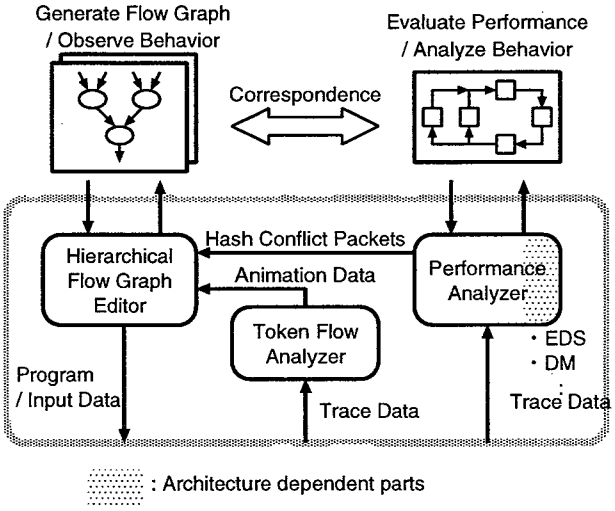
図 4.9にデータ駆動型VLSIプロセッサQv-mを実行機械として構築した、視覚的支援環境の構成を示す。図 (a) に示すように、マルチプロセッサシステムの評価用ボードがVMEバスを通してワークステーションに接続されたエミュレータならびにシミュレータを介して、視覚化を行っている。視覚化機能 (Visualization Function) は、図 (b) に示すように、フローグラフを対象としたツールとプロセッサを対象としたツールが統合されて構成されている。

4.4 結言

本章では、前章に述べた抽象データ駆動型処理プログラム (ADP) の生成手法を基礎にすれば、多面的に定義されたシステム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した。本手法では、論理的検証や、プロトタイピング実行の結果を複数の図的表現に視覚的に反映できるため、ユーザが頭の中に描いている仕様イメージと実際に記述された仕様との間の整合をとる、有効性確認もまた促進される。次に、システム記述のうち、ソフトウェア実現部分に関する性能評価支援手法として、データ駆動型プログラムの理論的な性能予測手法ならびに視覚的性能評価支援手法を



(a) 全体構成



(b)視覚化機能部の機能構成

図 4.9 視覚的評価支援環境の実現例

提案した。これらの数学的手法と実験的手法は目的・用途に応じて使い分けることによって、より柔軟な性能評価が可能になる。

今後は、複数人でのシステム開発をより効果的に支援するために、提案した相互変換手法を拡張できる可能性を追究することが考えられる。すなわち、

(a) 仕様記述相互間の変換手法の拡張：必ずしも完全には記述されない要求仕様記述に対

して、不足箇所の誘導ならびに記述相互間の一貫性保証のために、異なる協調者により記述された異なる表現形式間での相互変換を施すことは極めて有効な協調支援機能となる。したがって、処理実行に本質的なデータ依存関係とデータ構造を自然に表現できる、データ駆動型処理モデルを介して、このような相互変換機能を実現するアルゴリズムの定式化が望まれる。

(b) 協調作業と個人作業との調停戦略の定式化：協調作業は、有効な個々人の作業が有機的に統合して始めて、意義のあるものになる。これは、要求分析段階ならびにシステム設計段階においても、同様である。したがって、個人作業と協調作業を調和させて円滑に作業が進行できる調停戦略の定式化、さらには、これに基づく支援手法の確立が望まれる。

また、本章ではソフトウェア実現法に関する性能評価支援手法のみを提案したが、ハードウェア・ソフトウェアを含めた統合的なシステム実現法に関するコスト性能比を定量的に評価するための数学的モデルやプロトタイピング手法に関して、これらを確立することが残されている。

第 5 章

図的仕様記述支援システムの一実現法

5.1 緒言

本システム記述体系の実装に際しては、将来的により一般的な図的仕様記述を対象とする支援システムへ徐々に拡張するためのカーネル機能として利用可能なようジェネリック(自己適用的) に設計した。すなわち、本プロトタイプ自身を図的なシステム仕様記述表現を用いて定義し、さらに、この仕様記述から自身の機能を実現するデータ駆動型プログラムを半自動的に生成し、これを模擬する形態で既設の UNIX ワークステーション上に実装している。

以下本章では、自己適用的なプロトタイプの構築手法について述べ、これに基づいて実現したプロトタイプシステムとして、信号流れ図からデータ駆動型プロセッサ上で実行可能なプログラムを直接生成するシステムの構成について述べる。さらに、本システムを実時間制御システムならびに画像信号処理システムの仕様に適用した結果として、多面的な図的記述の例とその変換例を示し、本体系によりシステム記述からデータ駆動型 VLSI プロセッサ Qv-x 上で実行可能なオブジェクトプログラムが直接生成されることを実証的に明らかにする。

5.2 自己適用的なプロトタイプの構築手法

AESOP における変換手法ならびに記述支援手法をより実用的な水準で確立するには、実用的な問題への適用経験を蓄積して徐々に精錬化することが重要になる。すなわち、AESOP の多面的な図的仕様記述手法を実用的なシステム開発へ適用してドキュメント性の高い手法に精錬化し、さらに、AESOP により生成されるデータ駆動型プログラムの実行性能をデータ駆動型マルチプロセッサシステム Qv-x 上で定量的に評価・検討した結果

に基づいて、仕様記述から最適な実行オブジェクトへ変換するアルゴリズムを精錬化していく必要がある。

したがって本研究では、AESOPの手法の検証・精錬化とプロトタイプの構築による実証的検討とを同時並行に行なうために、AESOPの多面的な図的システム仕様記述手法をAESOP自身のプロトタイプの実現過程に自己適用しながら、AESOPプロトタイプをジェネリックに徐々に発展させる方針を採用している。具体的には、

Step1. 当面の再利用を重視した効率の良いデータ駆動型ソフトウェア部品を蓄積して高水準の仕様記述ができる環境を構築するために、まず、既存のワークステーション(WS)・ネットワーク上に、ノードおよびアークがプリミティブであるデータ駆動型グラフを変換・実行できるAESOPを実現する。具体的には、信号処理分野で用いられる信号流れ図からの変換系を実現する。

Step2. しかる後に、データ駆動アルゴリズム部分の実行形式をAESOP自身の環境で作成して、これをデータ駆動型システム上に実現すると同時に、GUI(Graphical User Interface)処理およびファイル処理をWS上に残したまま実現する形態に徐々に移行する。

Step3. さらに将来的には、GUIのみをWS上に残して、構造体データ処理をもデータ駆動型システム上に移行する構想である。

この自己適用・自己検証過程における各段階においては、AESOPプロトタイプの仕様を多面的かつ図的に記述・変更し、そして、これらの仕様記述に基づく機能をWS上に実現することになる。また、実装の各段階においては、前述の将来構想を念頭に置いて、核型プロトタイピングにより徐々に機能を拡張する必要がある。この際には、データ駆動型システム上への移行箇所とそのインタフェースが明確にでき、かつ、継続的に起こるであろう将来的な仕様の追加・変更に応じてWS上のプログラムを容易に保守できなければならない。これらの要件を考慮して、オブジェクト間のメッセージ授受によって、擬似的にデータ駆動型システム記述が模擬でき、かつ、再利用性が比較的高い言語であるC++を用いている。以下では、AESOPの基本的な機能構成とその振舞い、ならびに、データ構造に関する基本設計を図的システム仕様記述手法を用いて説明し、さらに、プロトタイプ第一版として構築した、信号流れ図を対象としたシステムの構成を示す。

5.2.1 AESOP の機能構成

プロトタイプの主要な機能構成を、本プロトタイプ自身を用いて多面的かつ図的に記述した**図 5.1**の左上の機能ブロック図に示す。この図に示すように、AESOP システムは、以下のサブシステムから構成される。すなわち、各種の図的表現の編集系 (Editor)、ADP を含む中間形式の生成・検証を行う変換系 CV(Converter)、Qv-x プロセッサ上でのプロトタイピング実行を行う実行系 EX(Executer)、ならびに、図的表現の描画情報と CV や EX で扱う本質的な中間形式情報とを仲介する入出力系 IO(IO subsystem) から構成される。さらに、データ中心型システム [67] として構成するために、図中右上に示す各種の情報の一貫性をアクティブに維持して管理する統合ファイル管理系 UF(Unified File System) が導入されている。また、仕様記述の部品を管理する部品管理系 CP(Component Manager) とその部品群をブラウズするためのブラウザ (Browser) が用意されている。

図 5.1の右下のシーケンス図は、本システムの振舞いの一例を表している。まず、編集系から、記述操作情報が IO に渡されると、その記述に関連する既に定義された中間形式情報を UF から直接受け取った CV は、ADP への変換を試みると同時に、論理的検証を行う。その結果が相互変換の結果として他の表現形式の編集系上に随時反映される。

編集系 ：編集系は、各図的表現の編集、ならびに、システムからの応答の表示を行う。AESOP が提供する図的表現は、ノードとアークからなるグラフ型表現と表形式表現に大別される。このため、編集系の実現においては、2 種の核機能を実現し、パラメトリックにアイコンなどを設定するだけで、AESOP が提供する任意の図的表現の編集・表示が可能なサブシステムを構築する。

入出力系 IO ：入出力系は、多様な仕様記述形式に依存しない柔軟な処理系を構成するための緩衝機能の役割を果たす。すなわち、編集系において定義された図的な仕様記述の内、変更された箇所に関する描画情報を対象として、中間形式に反映される本質的な情報のみを抽出する機能を有している。したがって、編集系における新しい表現形式の追加や既存の表現形式の変更に対して、AESOP システムの機能変更の影響を入出力系内で吸収できる。

変換系 CV ：変換系は、入出力系により抽出された、機能、振舞い、あるいは、データ構造に関する部分的な中間形式情報を、既に定義済の中間形式情報に統合する。す

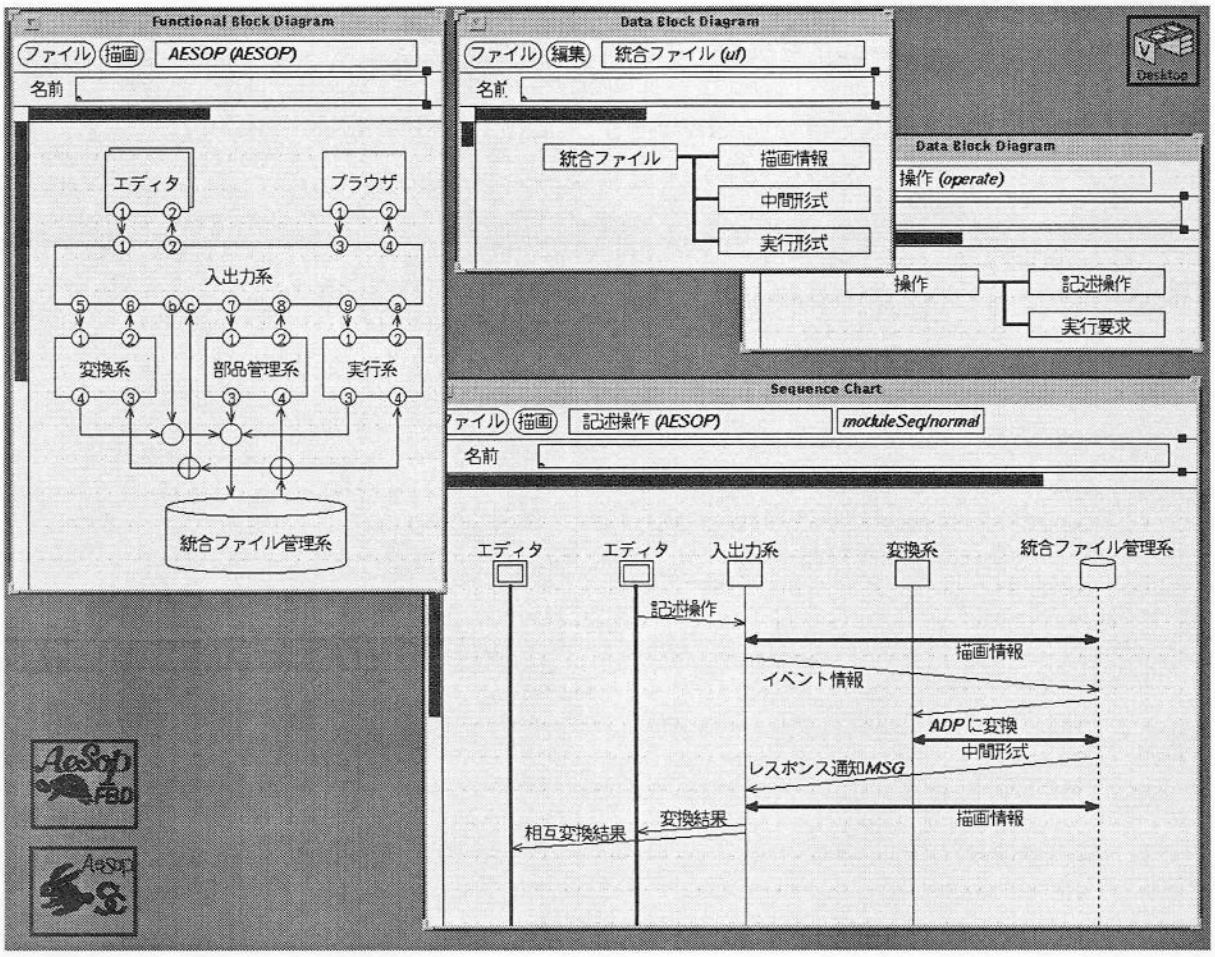


図 5.1 AESOP の機能構成

なわち、ADP へ変換する。また、そのために必要な検証を行い、構造的に不備があれば、その旨を入出力系 IO に通知する。同時に、部分的にでも抽象データ駆動型プログラムが生成できれば、そのプロトタイプ実行を実行系に指示する。

実行系 EX ：実行系は、統合的システム記述のソフトウェア実現のためのサブシステムである。すなわち、ADP に実行機械の制約を加えて、データ駆動型プロセッサ Qv-x 上で実行可能なオブジェクトプログラムと必要な入力パケットデータに変換して、プロトタイプとして実行する。

部品管理系 CP ：部品管理系は、仕様記述水準の部品の効果的な分類と管理を行う独立したサブシステムである。同時に、部品の登録時や検索・適用時には、入出力系や

変換系と連係して、部品再利用のための必要な情報収集・提供を行う。

統合ファイル管理系 UF ：統合ファイル管理系は、上記の各サブシステムに論理的なデータ構造の操作インタフェースを提供する。さらに、**図 5.1**中のシーケンス図にも示しているように、受動的なデータベース処理機能ではなくアクティブデータベース[68]として、トランザクションの結果をアクティブに他のサブプロセスに送出する形態で動作する。このため、サブプロセスの機能をより簡潔に構成することが可能となっている。

このような対話的かつ加法的なプログラム生成を実現するには、ユーザの記述操作毎に、これに対応する情報を各サブシステム間でデータ駆動的に授受する必要がある。これらのデータ授受は、自己適用的なシステム構築手法により、データ駆動的なアルゴリズムとして、表現される。よって、このアルゴリズムに従ったデータ授受は、UNIX プロセス間のメッセージ通信により実現されている。この実現法では、メッセージの順序関係さえ保存すれば、複数のサブシステムからのメッセージが混在しても動的データ駆動的に実行可能である。

5.2.2 AESOP におけるデータ構造

上述の統合ファイル管理系内で主に管理される情報は、**図 5.1**の右上に示す、仕様記述の描画情報、ADP を含む中間形式情報、ならびに、ソフトウェア実現法である実行形式プログラム情報である。以下では、編集系以外のすべてのサブシステムの操作対象となる中間形式のデータ構造とそのアクセスの実現法を示す。

中間形式は信号流れ図 SFG を中心とするブロック図の依存グラフを表す機能構造情報と、仕様の検証や実行形式の最適化に必要となるデータ構造情報から構成される。各情報の定義には、その要素間の存在制約と継承関係を明示できるデータ構造図が用いられている。

図 5.2に示す機能構造情報は、ADP の情報の保持のために、データ駆動型プログラムグラフの構造を表すノード、ポート、ならびに、ポート間の接続関係を示すアークに加えて、これらの間に成立する存在制約関係により構造化される。さらに、仕様記述の階層やホワイト／ブラックボックス部品を表現するブロック、ならびに、ブロックの入出力点を

表す外部ポートが付加され、各階層毎に局所的に構造化される。同時に、ブロックの入力ポートと出力ポート間に定義される入出力の因果関係が、ポート情報に重畳される。これによって、任意の階層での局所的な分析・検証が可能となる。

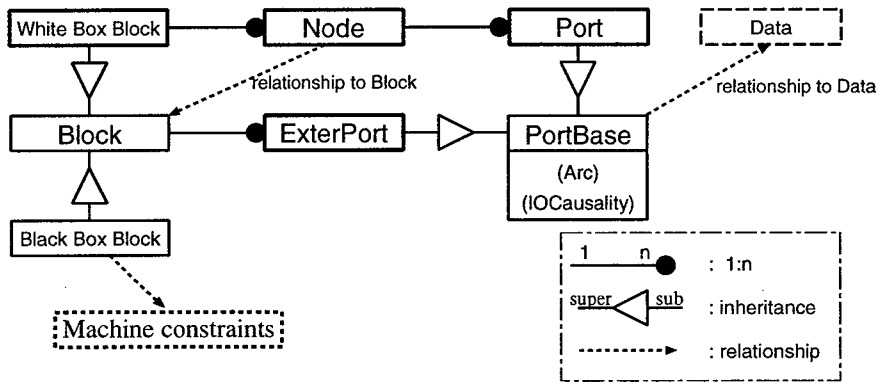


図 5.2 中間形式の機能構造情報

データ構造情報は、機能情報を補完して、多重ストリームに対する同時並行・パイプライン型実行、ならびに、演算語長の見積りを可能とする情報でなければならない。このため、ADP に対応するストリーム集合ならびにストリームとその要素は、抽象データ型を用いて構造化される。またこれに加えて、初期値の定義や実行経過の観測に必要なインスタンスが統合される。これらにより、仕様の論理的検証や実行形式でのタグ処理構造の生成を可能にしている。以上からデータ構造情報を**図 5.3**のようにモデル化する。

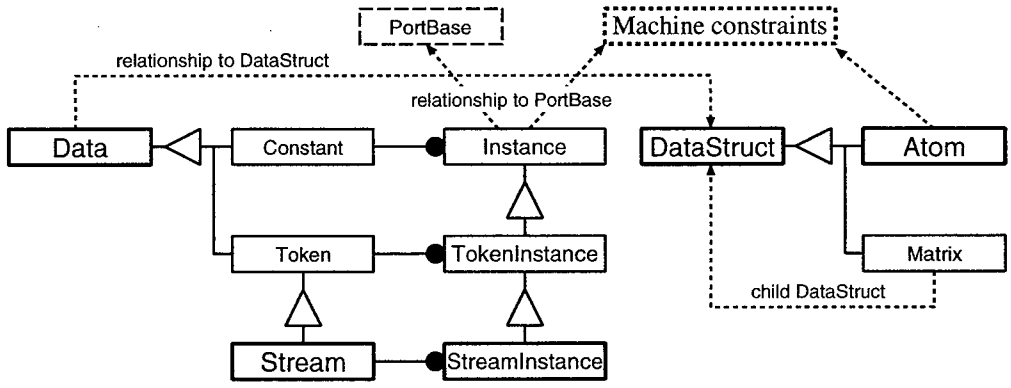


図 5.3 中間形式のデータ構造情報

この中間形式情報は、関係データベース上の第三正規形に写像され、データの冗長性を

排除して一貫性を維持される。さらに、この情報にアクセスする各サブシステムには、上述のデータ構造図に示す論理的なデータ構造に対するトランザクションが処理可能なように統合ファイル管理系 UF へのアクセスインタフェースが提供される。

一方、各サブシステム間で授受されるデータに関しては、統一的な形式を導入して、UNIX プロセス間のメッセージ授受として実現される。このメッセージ形式は、**図 5.4**に示すように、(送信元、送信先、操作、操作対象の識別子、引数) の 5 つ組から構成される。たとえば、編集系から入出力系へのデータ送信の場合、操作として、追加/削除/属性の設定、操作対象の識別子として、ブロック ID/記述要素 ID、引数として、操作と記述要素に応じた値を保持したメッセージが用いられる。

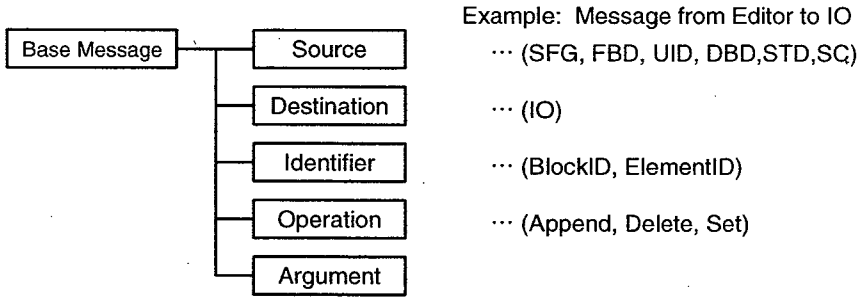


図 5.4 サブシステム間のメッセージ形式

以上のデータ中心設計とメッセージベースの実装により、各サブシステムを独立に設計・実現することが可能になる。実際に、次項に述べる試作システムでは、メッセージ授受機能を 8 クラス、統合ファイル管理系を 23 クラスで実現することによって、各サブシステムのクラス数を高々数個に抑えて簡潔に実装している。したがって、本実現法は自己発展的にシステムの機能拡張を実施する上での基盤となり得ると考えられる。

5.2.3 試作システム

以下は、実装した本プロトタイプシステムの機能構成である。なお、各サブシステムは UNIX 上のプロセスとして実現され、TCP(Transmission Control Protocol) によりデータ授受を行うため、計算機ネットワーク上で複数のユーザがシステム記述を行うことも可能な構成となっている。

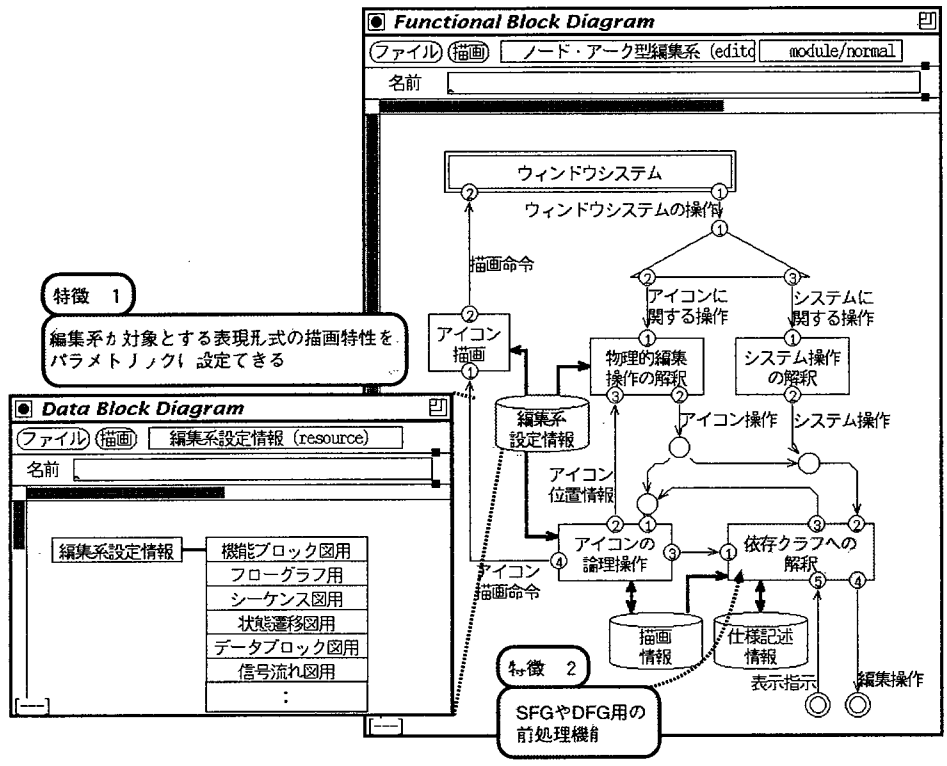


図 5.5 ノード・アーク系図的表現の編集系の構成

編集系 A 機能ブロック図、信号流れ図、シーケンス図、状態遷移図、およびデータブロック図などのノード・アーク系の図的表現を編集・表示する機能を提供する。**図 5.5** にその機能構成を示す。X-Window システムの Athena Widget を用いて実現され、リソースを変更するだけで、各表現形式の編集系として機能する構成となっている。また、アイコンのみが異なり、解釈が類似している表現形式の解釈ための前処理機能が付加されている。これによって、以下に述べる AESOP のカーネル機能のインタフェースが汎用化できる。

編集系 B ユーザインタフェースの図的表現、決定表、および入出力データ表などの表形式表現を編集・表示する機能を提供する。Tcl-Tk を用いて実現されている。

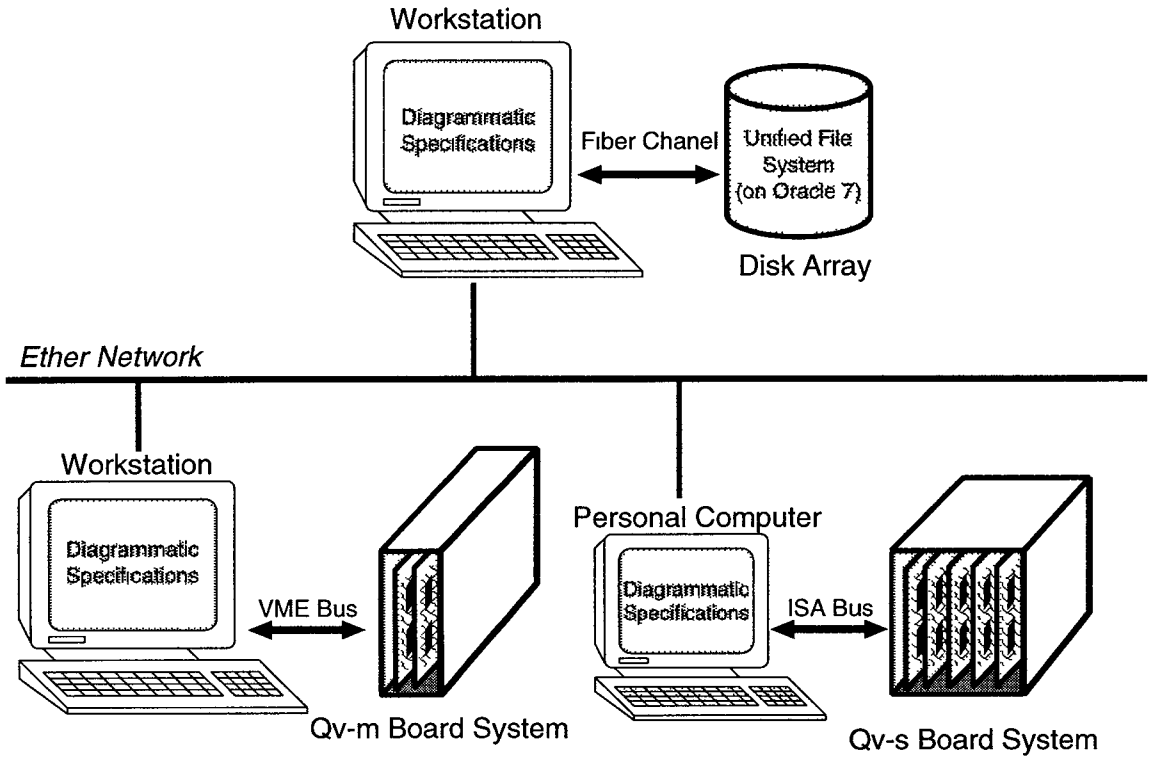
入出力系・変換系 機能ブロック図、信号流れ図、および入出力データ表から、抽象データ駆動型プログラム記述を包含する中間形式情報をインクリメンタルに生成し、対話的に検証する。

実行系 抽象データ駆動型プログラム記述から2種類のデータ駆動型VLSIプロセッサQv-mとQv-sに対応した、オブジェクトコードを生成する。さらに、実機上でプロトタイプ実行を行い、その実行結果を返す。

統合ファイル管理系 関係データベース管理システムOracle7を基礎として、前述の論理的なデータ構造操作インタフェースを提供する、と同時に、アクティブなデータベースとしての機能も提供する。

図 5.6は上記の機能を実現した試作システムの構成図である。図中上部は、AESOPのカーネル機能である変換系、入出力系、および統合ファイル管理系の機能を実現するワークステーションとディスクアレイからなるサブシステムである。統合ファイル管理系には、AESOPで取り扱うあらゆる情報が蓄積されるため、高信頼であり高バンド幅を持つディスクアレイが採用されている。図中中央部は、ADPからQv-mオブジェクトプログラムへの変換機能を有するワークステーション、ならびに、Qv-mボードシステムからなる、実行系サブシステムである。Qv-mボードシステムには、8プロセッサが搭載され、最大400MFLOPS(Mega Floating-Point Operations Per Second)の性能を発揮する。図中下部は、信号処理専用のQv-sプロセッサ用のサブシステムである。Qv-s用言語のアセンブラ機能を有するパーソナルコンピュータ、ならびに、Qv-sボードシステムからなる。Qv-sボードシステムには、62個のナノプロセッサが搭載され、最大15GOPS(Giga Operations Per Second)を超える処理性能を発揮する。この試作システム上のすべてのサブシステムは、Ethernetを介して接続され、いずれのサブシステムにおいても図的システム記述の編集やプロトタイピング結果の確認が可能なように構成されている。

Qv-s用サブシステムに実装されたQv-sボードのブロック図と写真をそれぞれ**図 5.7**(a)(b)に示す。写真中央から左下の領域にマウントされているのが、シャープ(株)によりVLSI実現された4つのデータ駆動型VLSIマルチプロセッサQv-sである。OCP(Operation and Control Processor)は、2.3節にも述べたように、整数(固定小数点数)演算、論理演算、タグ処理、ならびにテーブルメモリ参照機能を持ち、4つのナノプロセッサから構成されている。本ボード上では信号フィルタのための前処理のような処理を受け持つと同時に、ボード上のルータ(パケットスイッチ)としても働く。VMPはパケット内アキュムレータ機能とデータ/キャッシュメモリとを用いてデータ処理を行い、2つのナノプロセッサとメ



(a) システム構成

	Qv-m Board System	Qv-s Board System
No. of VLSI Processors	8 processors	62 processors
No. of chips	8 chips	25 chips
Maximum Performance	400 M FLOPS/system (50 M FLOPS/chip)	15 GOPS/system (600 MOPS/chip)
Data Format	32bit floating-point	12bit fixed-point

※ GOPS: Giga Operations Per Second
MOPS: Mega Operations Per Second

(b) Qv-xシステムの諸元

図 5.6 AESOP の試作システムの構成

モリから構成されている。本ボード上では、信号フィルタ処理の中心的処理あるいは後処理を受け持つよう構成されている。本ボード以外にも、9チップが搭載されたボードも用いられている。

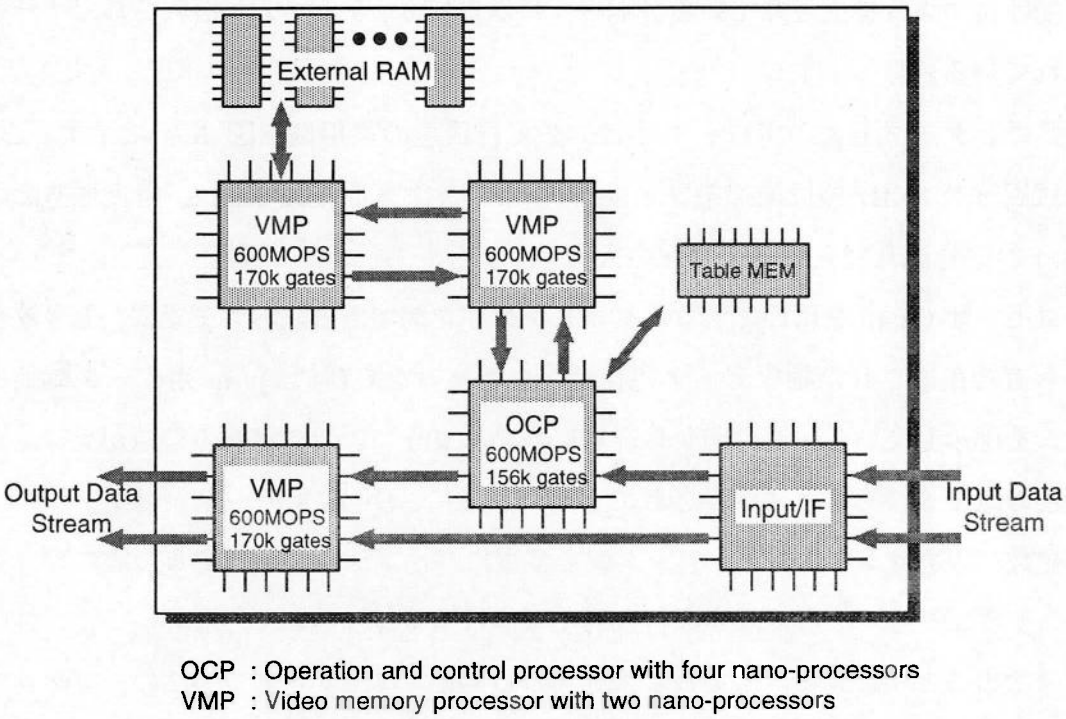
本プロトタイプ上でのプロトタイピング実行環境の適用例を**図 5.8** に示す。これは、通過域周波数1KHzの低域通過フィルタ(サンプリング周波数8KHz、阻止域遮断周波数3KHz)の信号流れ図表現に、複数の入力サンプリング・データ列を与えて、データ駆動型プロセッサQv-mを用いてプロトタイピングした例である。参考までに、信号流れ図表現から直接生成された抽象データ駆動型プログラムならびにQv-m用データ駆動型プログラムを図示している。この図から容易にわかるように、仕様記述の構造がソフトウェア実現法の水準までそのまま保存されている。さらに、Qv-m用データ駆動型プログラムには、複数の入力サンプリング・データ列を多重にパイプライン処理するための割り込み処理やスケジューリング機能などの付加的な機能は全く付加されていない。また、性能の側面では次のことが言える。すなわち、単一のQv-mプロセッサ上でのこのプログラムの最大入力可能パケット流量は1.5M Packets/secであるため、本プログラムを用いれば180回線強の信号系列を多重に処理可能である。また、同一のプログラム構造においてフィルタ係数を変更しさえすれば、48M bit/sec (1.5MHz × 32bits)の入力信号1回線をパイプライン型に処理可能でもある。

5.3 実時間制御・信号処理分野への適用

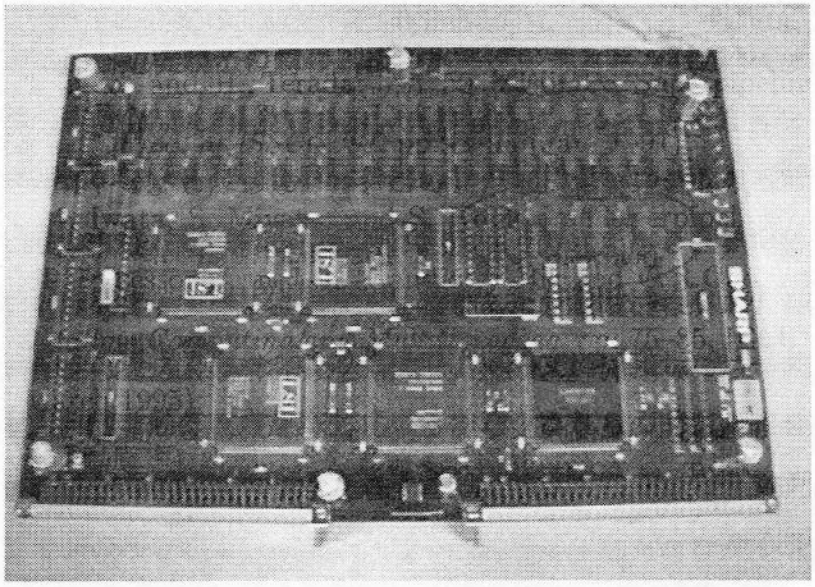
5.3.1 リフト制御問題への適用

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として提案されているリフト問題[69]を取り上げた。ここでは、3章に述べた状態遷移プロセスの多重実行が陽に反映するように、オリジナルのリフト問題を次のように改変している。

【リフト問題】:「10台のリフトが30階立てのビルにある。各階には、上昇/下降の要求を示すボタンがある。各リフトには、行き先の階を表すボタンがある。各リフトは、これらのボタンにより指定された要求階に移動し、停止する。要求がない場合、最後に停止した階で待機する。群管理機能は、各リフトに割り当てられた要求を総合して、フロアボタン要求を特定のリフトに割り当てる。また、リフトの中には奇数階にしか停止しないも



(a) ブロック図



(b) ボード写真

図 5.7 Qv-s マルチプロセッサ・ボードの構成

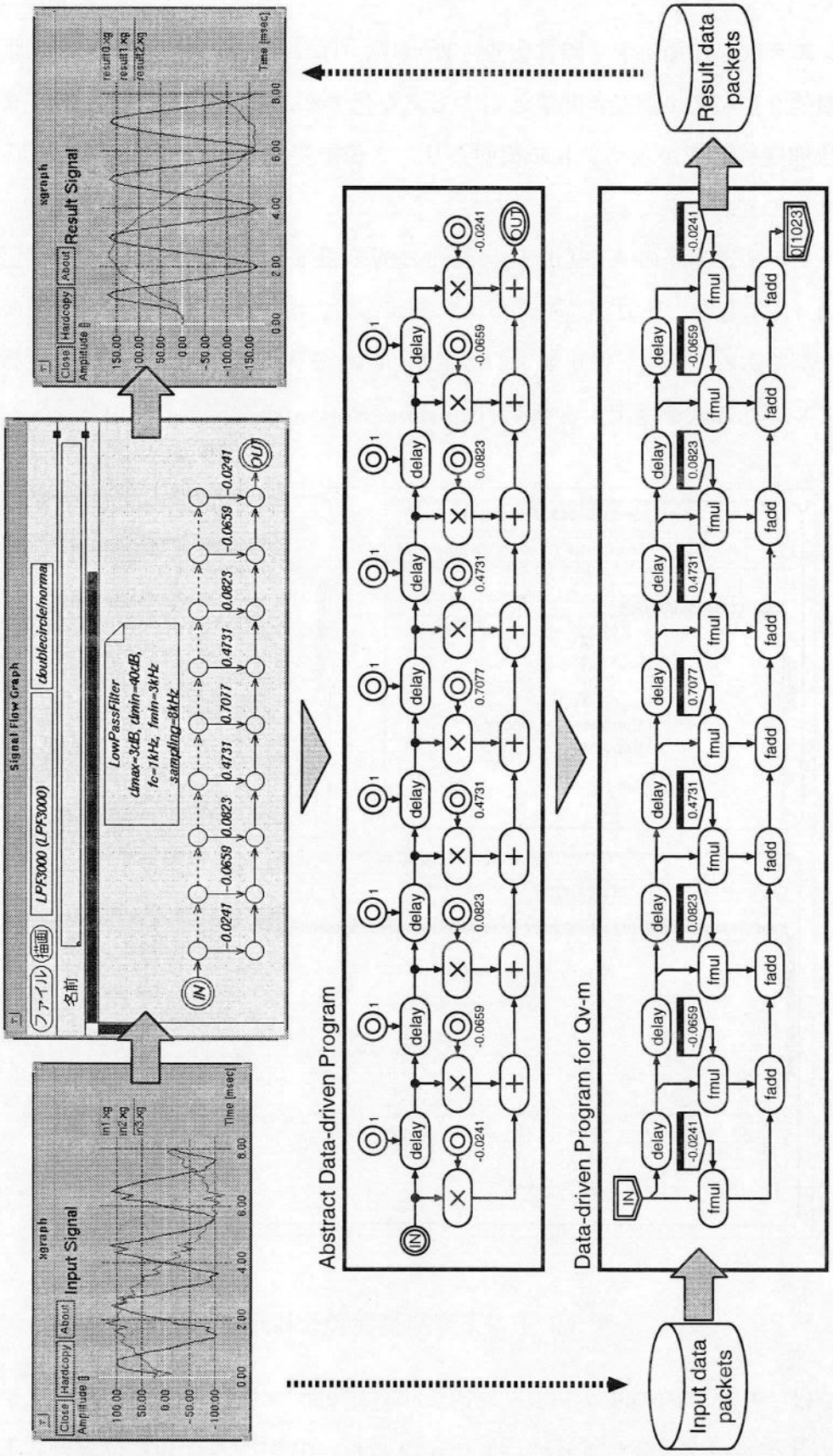


図 5.8 プロトタイピング環境の適用例

のがある。」

このシステムを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 **5.9**に示す。この仕様記述例は、3.2節の図 **3.4**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。

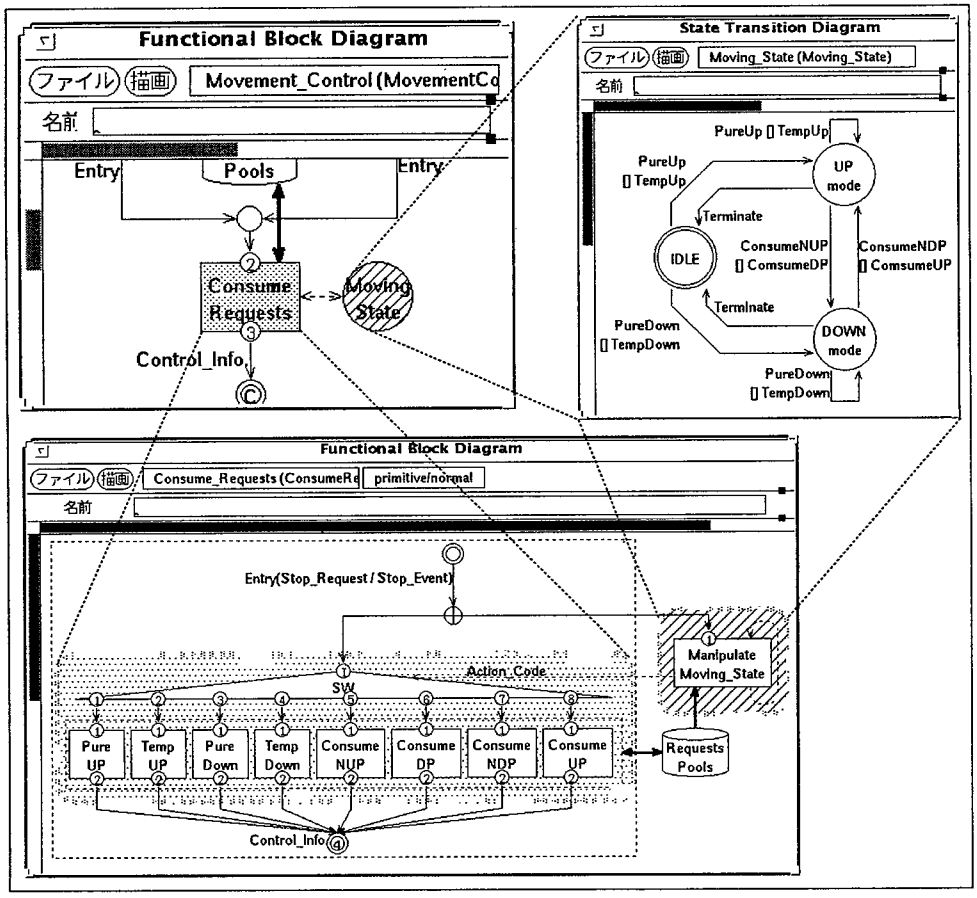


図 5.9 リフト制御問題の仕様記述例

ここでは、リフト内のリフトボタンおよび各階のフロアボタンからの要求が抽象化され、次の 4 グループに分類されている。すなわち、(i) リフトの現在位置から上昇する際

に消費できる上昇要求 CUP(Current Up request Pool)、(ii) リフトよりも下層階からの上昇要求 NUP(Next Up request Pool)、(iii) リフトの現在位置から下降する際に消費できる下降要求 CDP(Current Down request Pool)、(iv) リフトよりも上層階からの下降要求 NDP(Next Down request Pool)、である。

各リフトの状態は STD 記述 (Moving_State) 上で、待機状態 (IDLE)、上昇中 (UP mode)、および下降中 (DOWN mode) の 3 状態として定義されている。待機状態から上昇中への遷移では、リフトの現在位置で発生した上昇要求の消費 (PureUP)、あるいは、現在の階よりも上層階で発生した要求の消費のための一時的な上昇 (TempUP) のいずれかを実行する。待機状態から下降中への遷移も同様である。上昇中から下降中への遷移では、DP(CDP と NDP の和) の消費 (ConsumeDP) あるいは NUP の消費 (ConsumeNUP) のための下降が行われる。逆の遷移も同様である。この仕様記述例では、STD 記述にしたがって、各遷移の動作、すなわち、FBD 記述中の副アルゴリズムが選択される構成となっている。

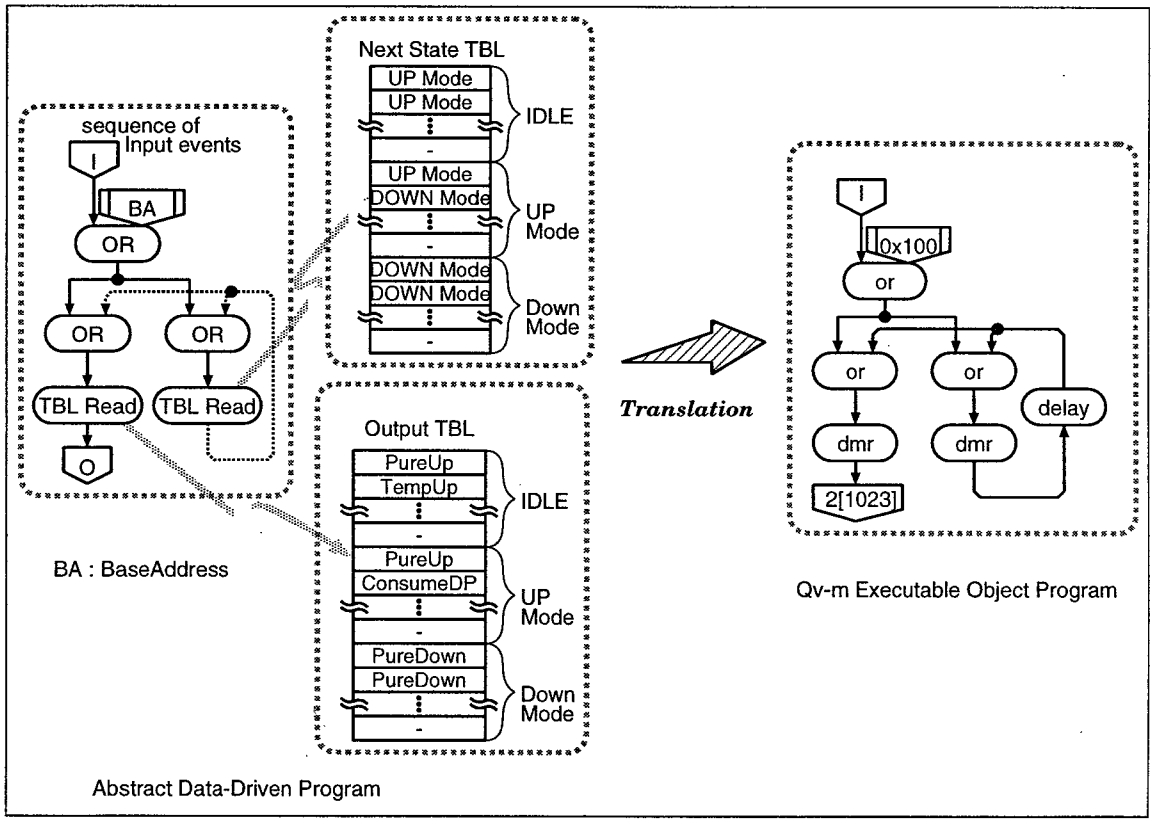


図 5.10 リフト制御問題における状態遷移動作の変換例

図 5.10の左図は、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であり、複数のリフトの状態遷移論理を多重に解釈実行することが可能である。ここでは、**図 3.7**の生成規則を適用して、**図 3.5**に示したFSM構造を生成した例を示している。また、入力イベントと現状態を鍵として参照される、次状態テーブルならびに出力テーブルの変換結果を示している。

このプログラムは**図 5.10**の右図に示すように、データ駆動型VLSIプロセッサQv-m[39]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、多重に実行可能なオブジェクトコードに容易に変換される。ここでは、ストリーム中の要素の識別子をインクリメントするdelay命令を状態更新ループ中に挿入し、テーブル参照機能としてデータメモリの参照命令dmrを用いている。

以上のリフト問題への適用例から、状態を伴う処理に関しても、仕様記述からのデータ駆動型プログラムの直接生成が可能であることが確認された。

5.3.2 MUSE信号デコーダへの適用

実装したプロトタイプシステムの有効性を確認するため、ハイビジョン衛星放送用MUSE (Multiple Sub-Nyquist Sampling Encoding) 信号[70]のデコーダにおける動領域部分の仕様記述を取り上げた。本デコーダは、**図 5.11**の上部の二つの機能ブロック図で表現しているように、輝度信号Yと色信号Cのそれぞれに対して、非線形符号を線形化する逆ガンマ処理 (Nonlinear) とFM変調時のS/N比の平均化のためのディエンファシスフィルタ (Deemphasis Filter) を施した後、16.2MHzでサブサンプリングされた信号をフィールド内内挿 (Y/C-interpolation) により補間し、最後に、Y信号とC信号からRGB信号に変換する処理である。

図 5.11は、デコーダの一部であるYC-RGB信号変換フィルタの仕様を、SFGを用いて記述する過程に本手法を適用した例である。**図 5.11**の右下図は、YC-RGB信号変換フィルタの機能構造とストリームデータの仕様から加法的に生成された、IF(ADP)である。このIFから導出されたTVGの検証により、入力Y,C_r,C_bから出力R,B,Gまでのトークンの可達性、ならびに(Y,C_r)とR、(Y,C_b)とB、(Y,C_r,C_b)とGの間の因果関係が検出される。これにより、TVGの検証結果がSFG上の構成要素単位で反映される。その結果、この仕様記述がストリームを対象とするADPへ変換される。**図 5.11**の左下図は、この

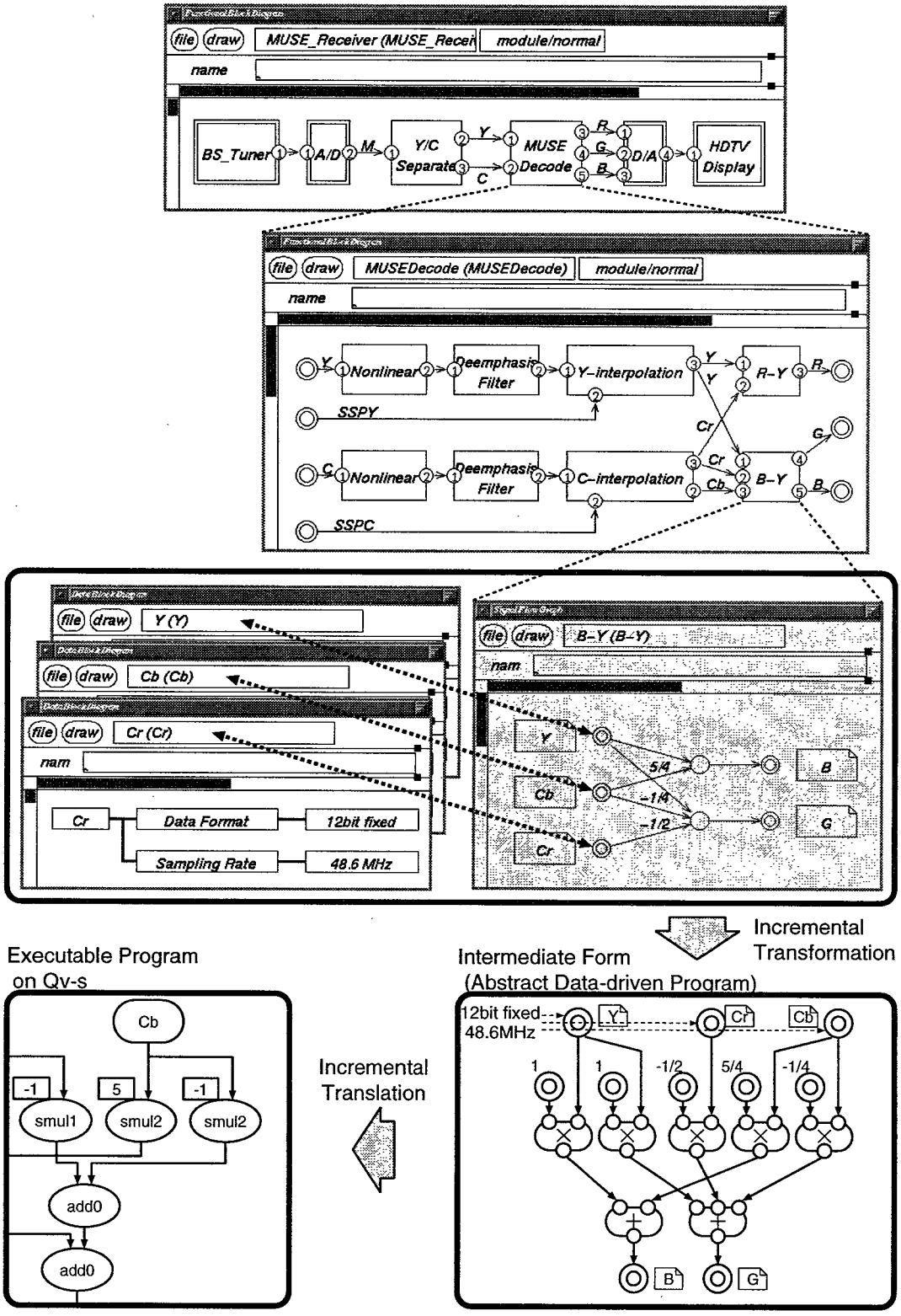


図 5.11 MUSE 信号デコーダの仕様記述例とそのデータ駆動型プログラムの生成例

ADPから生成される、データ駆動プロセッサQv-s[17]用のプログラムの一部である。以上のことから、本AESOPプロトタイプ・システムの環境において、SFGから、ADPを介して、Qv-sプログラムが同形のまま加法的に生成されることが確認された。

このように本手法を用いれば、基本的な信号処理システムを実現するためのパラメタやアルゴリズムの変更を、直接ソフトウェア実現法としてのオブジェクト・プログラムへ反映でき、了解性の高い図的仕様記述の水準でシステムの生産ならびに保守を行える。さらには、システム内のすべての機能がプログラム可能なビデオデコーダを導入することによって、多様化している動画圧縮方式に柔軟に適応可能な動画像処理システムの実現が可能になる。たとえば、現在MUSE信号デコーダを実現しているQv-sシステムの構成に少数のデータ駆動型プロセッサを付加するだけで、EDTV IあるいはIIのデコードが可能になることが確認されている。現在開発中の次世代チップを用いれば、1チップあたり約4GOPS(Giga Operations Per Second)の性能を発揮できるため、MPEG2/4のデコード機能もシステム内に含めることが期待される。

5.4 結言

本章では、統合的システム記述体系の一実現法として、今後の拡張性を考慮した自己適用的なプロトタイプの構築手法とこれに基づくAESOPプロトタイプシステムの概要とその適用事例について述べた。本構築手法では、データ中心アプローチによる構造体データの統合的管理とサブシステム間のメッセージの共通化を徹底したため、非常に簡潔な構成でプロトタイプシステムの実装が可能となった。したがって、本プロトタイプシステムは、今後の漸増的な機能拡張のための核システムとして機能する機能することが期待される。また、2種類の動的データ駆動型VLSIマルチプロセッサシステムを実行系のエミュレーション機能として組み入れたため、非常に高速なプロトタイピング環境が実現された。

また、本プロトタイプをリフト制御問題に適用した結果、データブロック図DBDおよび状態遷移図STDにより情報を補完すれば、仕様記述中の機能ブロック図から、ほぼ一対一に対応した実行可能プログラムが直接生成可能なことが確認された。さらに、ハイビジョン衛星放送用MUSE信号デコーダに適用し、1チップあたり600MOPS(mega operation per second)の性能を達成するデータ駆動型プロセッサQv-s上で実行可能なプログラム

が、信号流れ図から加法的に直接生成可能なことを確認した。これによって、本体系が、将来多様化するであろうマルチメディア信号処理の領域においても、柔軟なシステムを構築するための支援環境になりうる可能性が見出せた。

今後は、本プロトタイプをデータ駆動型プロセッサシステム上へ移植する検討を通して、本システム記述体系自身を適用例とした実証的な定量的評価・検討を行うことが望まれる。また、実用規模の応用問題への適用を行うためには、部品を徐々に蓄積していくと同時に、部品の効果的な分類手法 [71] に基づく知的な検索機能の検討も必要である。

第6章

結　論

本論文では、ソフトウェア/ハードウェアを区別しない機能的な水準で統合的にシステム記述を継承・保守する体系の模索のために、自然な処理原理であると考えられるデータ駆動パラダイムを導入した統合的システム記述体系について考察した。すなわち、システムの初期開発に留まらず保守・運用の段階に至るまで、データ駆動パラダイムを一貫して導入することによって、(1) システムの要求仕様記述と詳細設計仕様記述との間の乖離、ならびに、詳細設計仕様記述とハードウェア仕様記述との間の乖離が、同時に原理的に解消されること、(2) 図的なシステム仕様記述から、その実現法に直接変換可能な抽象データ駆動型プログラムが生成可能になるため、システム記述の水準での開発・保守が可能になること、(3) システム記述とその実現法との間の中間的な水準で、記述の安全性の保証、部品化、ならびに、部分的プロトタイピングの基盤が提供されるため、システム記述の水準でより効果的な開発・保守が可能になること、さらに、(4) これを基礎とした対話的なシステム記述体系の構築が可能になること、などについて考察した。以下に本研究で得られた諸成果をまとめる。

第2章においては、統合的なシステム開発・保守の方法論として、従来の逐次代入型の文章記述に代わり、データ駆動パラダイムを基礎とする図的な統合的システム記述の継承・保守を重要視したシステム開発手法について述べた。まず、これまでの統合的システム開発方法論の一つのアプローチである、統合化 CASE (Computer-Aided Software Engineering) ならびにハードウェア・ソフトウェア協調設計手法における問題点が、アーキテクチャ上の欠点を強く反映したプログラム記述の困難さにあることを示すと共に、統合的なシステム記述の体系に求められる要件を明らかにした。さらに、データ駆動原理に基づくパラダイムがシステム設計のあらゆる水準で有効に機能することを明らかにし、

統合的システム記述によるシステム設計環境の理論的基礎になりうることを示した。

第3章においては、図的表現を中心とした統合的なシステム記述体系 AESOP(Advanced Environment for System Oriented Production) の概要について述べた。さらに、統合的システム記述のための多面的かつ図的な仕様記述手法、ならびに、その変換のために拡張した抽象データ駆動型処理モデルを提案した。本処理モデルは、純粋なデータ駆動原理には規定できない履歴依存性の内、複数の入力ストリームを副作用なく多重に処理可能な状態遷移処理を規定するため、実時間制御における状態推移から、ハードウェア水準のフリップ・フロップに至るまでの多様な処理を包含できる。最後に、本体系の根幹となる技術として、多面的な図的システム仕様記述から抽象データ駆動型処理プログラムを加法的 (インクリメンタル) に直接生成する手法として、集合間の要素の写像関係に基づくプログラム生成手法を定式化し、実現法に依存しない水準でのシステム記述の継承・保守を可能とする基盤を与えた。

第4章においては、AESOP システムを広範囲のユーザにとってより親しみやすい仕様記述環境として提供するために、システムとの対話を通してインクリメンタルに仕様を記述可能とする、仕様記述支援手法について述べた。すなわち、多面的に定義されたシステム仕様記述相互間での変換を介した対話的な有効性確認・検証支援、ならびに、ソフトウェア/ハードウェア実現法の選択のために必須となる性能評価支援の一手法を提案した。また、残された問題として、複数人でシステムの開発・保守を実施する際の協調作業支援手法、ならびに、ハードウェアのコスト性能比の評価支援手法などがあることにふれた。

第5章においては、AESOP システムの実現法として、今後の拡張性を考慮した自己適用的な実現法を提案し、これに基づくプロトタイプシステムの構成について述べた。本構築手法では、データ中心アプローチによる構造体データの統合的管理とサブシステム間のメッセージの共通化を徹底したため、非常に簡潔な構成でのプロトタイプシステムの実装が可能となった。また、2種類の動的データ駆動型 VLSI マルチプロセッサシステムを実行系のエミュレーション機能として組み入れたため、非常に高速なプロトタイピング環境が実現された。したがって、本プロトタイプシステムは、今後の漸増的な機能拡張のための核システムとして機能することが期待される。次に、本システムを実時間制御システムならびに画像信号処理システムの仕様に適用した結果として、多面的な図的記述の例とその変換例を示し、本体系によりシステム記述からデータ駆動型 VLSI プロセッサ Qv-x 上

で実行可能なオブジェクトプログラムが直接生成されることを実証的に明らかにした。

本研究の最大の成果は、従来の逐次型処理方式に基づく文章型記述の部品化手法の単なる拡張ではなく、新しい動的データ駆動型処理モデルに基づいた、統合的システム記述環境ならびに保守環境の基本的な像を提示したことが挙げられる。しかし、本論文ではこのような統合的システム開発・保守パラダイムの基本的な問題を示したに止まっており、実際的な規模でのシステム開発にこのような手法を活用するには、各章の結言にも述べたように、まだ多くの課題が残されており、生産性・保守性などに関する定量的な評価検討も進める必要がある。ただし、第2章に引用したDDLの事例では、プログラミング水準でのみデータ駆動型プログラムの部品化を徹底しただけでも、生産性と相関があるドキュメント量が従来手法に比べ1/3に削減されている。これを考慮すれば、本論文に述べた一貫した統合的システム記述体系の下では、10倍以上の生産性向上も夢ではないと考えられる。

本研究では、図的表現とこれに陽に表現されている並列処理性をそのまま実現法の水準にまで一貫して継承することによって、実現法に煩わされずに純粋にシステムの構想やアルゴリズムを探求する環境の提供を重要な目的としている。人間の自然な思考は一般的には並列あるいは図的であり、かつ曖昧に始まりしだいに精密化される。この意味で、本研究のアプローチは自然な方向である。しかしながら、図的表現は、現状のワークステーション水準での編集機能では、文章型記述に比べて操作性・閲覧性に乏しいため、特に、構造化エディタなどに慣れたシステムエンジニアなどには、受け入れられ難い点もある。これに関しては、発散的な思考・収束的な思考の道具を提供する観点からのユーザインタフェース、あるいは、個々の利用者固有の思考過程に適応可能なユーザインタフェースに関する研究が必要になる。

本統合的システム記述体系におけるハードウェア実現法は、生産性・保守性の向上を可能にするのみならず、小電力・高性能である特徴も有している。すなわち、自己タイミング型パイプライン処理機構によるハードウェア構成は、同時並行・パイプライン型並列処理が自然に実現できるだけでなく、単位消費電力あたりの処理能力PPW(Performance Per Watt)にも優れている。現在完成しているデータ駆動型VLSIプロセッサは、800MIPS/W水準を達成しており、次世代集積技術によれば、さらに小電力消費化が可能になる見通しがある。したがって、当面は、小電力化が必須である携帯用情報機器や、高性能な入出力

処理が要求されるマルチメディア機器など、現状では専用 ASIC でしか対応できないシステムに対して、本プロセッサが適用されることが期待される。特に昨今、これらのシステムに対する要求は加速して高度化しているため、本統合的システム記述体系の下で、漸増的にシステムの開発・発展を効果的に実施できるであろう。これに関しては、現在、新しいメディアプロセッサ NMP(New Media Processor) システムの開発に適用を始めているところである。

情報通信システムは、本来、人間の知的活動を促進できるよう、人間の頭脳や感覚器を支援する道具として、機能することが望まれる。このためには、現状の多くの計算機が主として提供している、左脳型の論理的記号処理だけではなく、人間の知的活動の大部分を占めていると言われている、画像・音声の認識/合成などの、右脳型の直観的連想 (認識) 処理を伴う、機能的記憶処理を提供することもまた重要である。論理的な記号処理に関しては、人間が効率良く処理できない領域であるため、非常に重要な分野でもある。このため、基本的な処理方式やアーキテクチャがほとんど進歩しないにも関わらず、関連技術が発展し、計算機のダウンサイジングやネットワーク化が進められてきた。その結果、現状の情報システムは、個々の要求に応じたシステム構成をとれる、個性化された計算機を多数集合した分散処理形態をとりつつある。また一方では、個性化・分散化に相反するメインフレームは、周辺機器との間の入出力データ・チャネル数が多いため、現状では、本来の数値計算機能とは別の用途である、大規模なデータベースのトランザクション管理などに利用されている。しかしながら、このような情報システムの機能の個性化・分散化、ならびに記憶の大容量化に伴って、基本的な処理方式であるノイマン型逐次処理方式に起因する問題が顕在化してきて、有機的に結合した分散型情報システムへの発展が困難になってきているのが現状である。この意味で、本論文に提案したデータ駆動パラダイムは生来的に自律分散的な動作原理を有しているため、これらの問題の一つの解決法になることが期待される。一方、直観的な連想処理については、古くから理論的な研究が行なわれていたが、当時はこれを効果的に実現するハードウェア技術がなかった。昨今の集積回路技術の進歩により、様々な実現法が研究されてはいるが、これらの連想処理モデルが持つ本来の自律分散的な動作モードとは本質的に異なる逐次型処理計算機上で、種々の実験的評価を行なわざるを得ないため、効果的に実証的研究や実現が行なえない状況にある。この観点からも、単純な処理実行原理を持つ機能要素を規則的に組み合わせて、望みの機

能を実現する自律分散的なシステム構成パラダイムである、データ駆動パラダイムが有望である。たとえば、このような動作様式は、直観的な連想処理の実現法として注目されている、人工神経回路網 [72] や遺伝的アルゴリズム [73] などのノンアルゴリズミックな処理にも見られ、動的データ駆動型処理方式によって超並列にシミュレート可能である [72, 73]。したがって、本パラダイムはより知的な情報処理システムの構成パラダイムとしても発展することが期待される。

従来手法では、ソフトウェア部品やハードウェア部品の積み重ねによる過去の多くの資産と経験とに後押しされ、不断の努力が続けられているため、一朝一夕には、革新的なパラダイムシフトは起こらないであろう。しかしながら、データ駆動パラダイムに基づく統合的システム記述体系に関して基本的な像を示した本研究の成果が、そのようなパラダイムシフトの契機になることを期待して、本論文の結びとしたい。

謝　辞

本研究の全過程を通じて、終始懇切なる御指導、御鞭撻を賜った大阪大学大学院工学研究科情報システム工学専攻寺田浩詔教授に衷心より感謝の意を表する。

本論文作成にあたり、懇篤なる御指導いただくと共に種々の御高配を賜った大阪大学大学院工学研究科情報システム工学専攻白川功教授、藤岡弘教授、村上孝三教授、ならびに電子情報エネルギー工学専攻岸野文郎教授に深謝の意を表する。

本研究の遂行にあたり、常に温かい御激励をくださった大阪大学工学部長鈴木胖教授、大阪大学大学院工学研究科情報システム工学専攻薦田憲久教授、ならびに西尾章治郎教授に深謝する。

筆者が大学院電子工学専攻の博士課程に在籍して以来、電子工学一般及び各専門分野に関し御指導、御教示を賜った大阪大学大学院工学研究科電子工学専攻児玉慎三名誉教授、西原浩教授、濱口智尋教授、尾浦憲治郎教授に厚く感謝する。

筆者が寺田研究室に在籍以来、種々の面でひとかたならぬ御指導、御助言ならびに御激励頂いた宝塚造形大学大村皓一教授、福井大学工学部浅田勝彦教授、筑波大学電子・情報工学系西川博昭助教授、大阪大学大学院工学研究科滝根哲哉助教授に厚くお礼申し上げる。

また本研究の遂行にあたり、データ駆動型VLSIプロセッサシステムを御提供頂くと共に種々の技術的な御討論、御助言を頂いた三菱電機株式会社小守伸史博士、田村俊之氏、坪田浩乃女史、シャープ株式会社宮田宗一博士、岡本俊弥氏、芳田真一氏に心からお礼申し上げる。

末筆ながら寺田研究室の許炎助手、江木康雄技官、大学院学生の唐沢圭氏、上方輝彦氏、新吉高氏、ならびに秘書の若林市子嬢には種々の面で御協力いただいた。ここに記して感謝する次第である。

参考文献

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

[2] P. Cord and E. Yourdon: "Object-Oriented Analysis," *IEEE Computer Society Press Tutorial*, pp.272 – 289 (1990).

[3] D. A. Monarchi and G. I. Puhr: "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM*, Vol.35, No.9, pp.35-47 (Sep. 1992).

[4] N. S. Woo, A. E. Dunlop, and W. Wolf: "Codesign and Cospecification," *IEEE Computer*, Vol.27, No.1, pp.42–47 (Jan. 1993).

[5] R. Balzer: "A 15 Year Perspective on Automatic Programming," *IEEE Trans. on Software Engineering*, Vol.SE-11, No.11, pp.1257–1268 (Nov. 1985).

[6] 磯田, 黒木: "統合化 CASE システム SoftDA の機能," コンピュータソフトウェア, Vol.10, No.2, pp.26-37 (1993-03).

[7] R. K. Gupta, C. N. Coelho, Jr., and G. DeMicheli: "Program Implementation Schemes for Hardware-Software Systems," *IEEE Computer*, Vol.27, No.1, pp.48–55 (Jan. 1993).

[8] D. E. Thomas, J. K. Adams, and H. Schmit: "A Model and Methodology for Hardware-Software Codesign," *IEEE Design and Test of Computers*, Vol.10, No.3, pp.6-15 (Sep. 1993).

[9] 安浦: "基本ソフトウェアとコデザイン," 情報処理, Vol.36, No.7, pp.620–626 (1995-07).

[10] 西川, 寺田, 芳田, 宮田, 日根, 野口, 西川, 原, 嶋, 鷲野: "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 (1990-08).

[11] M. Iwata and H. Terada: "Multilateral Diagrammatical Specification Environment Based on Data-Driven Paradigm," In G. Gao, J.-L. Gaudiot, and L. Bic, editors, *Advanced Topics in Dataflow Computing and Multithreading*, pp.103–112, IEEE Computer Society Press (March 1995).

[12] 岩田, 寺田: "図的仕様記述からのデータ駆動型プログラムの生成手法", 情報処理学会論文誌, Vol. 36, No. 5, pp.1203–1210 (1995-05).

[13] 唐沢, 新, 岩田, 寺田: "信号流れ図からのデータ駆動型プログラムの対話的生成手法," 電気学会論文誌C, Vol. 116-C, No.11, pp.1307–1312 (1996-11).

[14] 岩田, 坪田, 田村, 小守, 寺田: "動的データ駆動型処理システム Qv-x の視覚的評価支援環境," 電子情報通信学会コンピュータシステム研究会, Vol. CPSY94-79, pp. 57–64 (1994-12).

[15] 田村, 坪田, 小守, 久間, 岩田, 寺田: "データ駆動型プロセッサ RAPID のソフトウェア開発環境," 電子情報通信学会コンピュータシステム研究会, Vol. CPSY94-93, pp. 73–78 (1994-12).

[16] 上方, 岩田, 滝根, 寺田: "分散キューバッファを持つデータ駆動型プロセッサ Qv-x の性能評価," 電気学会論文誌C, Vol. 116-C, No.11, pp.1295–1300 (1996-11).

[17] 芳田, 紫竹, 松浦, 村松, 岡本, 宮田: "映像信号処理向きデータ駆動型プロセッサ," 信学技報, Vol.DSP95-109, pp.39-46 (1995-10).

[18] H. Terada, M. Iwata, and S. Miyata: "600MOPS Super-Pipelined Data-Driven Processors and Their Application to HDTV Signal Processing," *Proc. of Australasian Computer Architecture Workshop'96*, (1996 to be published).

[19] D. T. Ross: "Structured Analysis(SA): A Language for Communicating Ideas," *IEEE Trans. on Software Engineering*, Vol. SE-3, No.1, pp.16–34 (1977).

[20] P. T. Ward: "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing," *IEEE Trans. on Software Engineering*, Vol. SE-12, No.2, pp.198–210 (1986).

[21] M. A. Jackson: "*System Development*," p.418, Prentice/Hall (1983).

[22] B. B. Wyatt, K. Kavi, and S. Hufnagel: "Parallelism in Object-Oriented Languages: A Survey," *IEEE Software*, Vol.9, No.6, pp.56-66 (June 1992).

[23] E. A. Lee, W. H. Ho, E. E. Goei, J. C. Bier, and S. Bhattacharyya: "Gabriel: A Design Environment for DSP," *IEEE Trans. on Acoustic, Speech and Signal Processing*, Vol.37, No.11, pp.1751–1762 (Nov. 1989).

[24] A. Kalavade and E. A. Lee: "A Hardware-Software Codesign Methodology for DSP Applications," *IEEE Design and Test of Computers*, Vol.10, No.3, pp.16–28 (Sep. 1993).

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

[26] Arvind and R. Iannucci: "Two Fundamental Issues in Multiprocessing," *Proc. of 10th International Symposium on Computer Architecture*, pp.426–436 (June 1983).

[27] 寺田: "VLSI 向きデータ駆動型プロセッサ," 信学誌, Vol.72, No.7, pp.742-749 (1989-07).

[28] N. C. Shu: "Visual Programming," Van Nostrand Reinhold Company Inc., New York (1988).

[29] 西川, 寺田: "視覚的プログラミング環境," 情報処理, Vol.30, No.4, pp.354–362 (1989-04).

[30] M. Hirakawa, S. Iwata, I. Yoshimoto, M. Tanaka, and T. Ichikawa: "HI-VISUAL Iconic Programming," *Proc. of IEEE Workshop on Visual Languages*, pp.305–314 (1987).

[31] H. Shirasu, T. Suzuki, Y. Maejima, S. Tanabe, and A. Kusaba: "Innovative Approach to Switching Software Design Using Dataflow Concept," *Proc. of International Switching Symposium*, S-B 4.3 (Mar. 1987).

[32] 鈴木, 前島, 田辺, 白須, 大坪: "データ駆動論理方式交換ソフトウェア設計手法," 信学論 *(B-I)*, Vol.J72-B-I, No.4, pp.343–352 (1989-04).

[33] P. A. Hausler, R. C. Linger, and C. J. Trammell: "Adopting Cleanroom Software

Engineering with a Phased Approach," *IBM Systems Journal*, Vol.33, No.1, pp.89–109 (Jan. 1994).

[34] V. P. Srini: "An Architectural Comparison of Dataflow Systems," *IEEE Computer*, Vol.19, No.3, pp.68–88 (Mar. 1986).

[35] R. P. Hopkins, P. C. Treleaven, and D. R. Brownbridge: "Data-Driven and Demand-Driven Computer Architecture," *ACM Computing Surveys*, Vol.14, No.1, pp.68–88 (Jan. 1982).

[36] J. Gurd, C. Kirkham, and I. Watson: "The Manchester Prototype Computer," *Communications of the ACM*, Vol.28, No.1, pp.34–52 (Jan. 1985).

[37] T. Shimada, K. Hiraki, and S. Sekiguchi: "Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computation," *Proc. of 13th International Symposium on Computer Architecture*, pp.226–234 (1986).

[38] J. Hicks, D. Chiou, B. S. Ang, and Arvind: "Performance Studies of Id on the Monsoon Dataflow System," *MIT Technical Report*, No.345 (May 1993).

[39] S. Komori, T. Tamura, F. Asai, H. Tsubota, H. Sato, H. Takata, Y. Seguchi, T. Ohno, T. Tokuda, and H. Terada: "A 50 MFLOPS Superpipelined Data-Driven Microprocessor," *Proc. of ISSCC '91*, pp.92–93 (Jan. 1991).

[40] H. Terada, M. Iwata, S. Miyata, and S. Komori: "Superpipelined Dynamic Data-Driven VLSI Processors," In J.-L. Gaudiot L.Bic and G.R.Gao, editors, *Advanced Topics in Dataflow Computing and Multithreading*, pp. 75–85, IEEE Computer Society Press (March 1995).

[41] S. Komori, H. Takata, T. Tamura, F. Asai, T. Ohno, O. Tomisawa, T. Yamasaki, K. Shima, K. Asada, and H. Terada: "An Elastic Pipeline Mechanism by Self-Timed Circuits," *IEEE J. Solid-State Circuits*, Vol. SC-23, No. 1, pp. 111–117 (Feb. 1988).

[42] F. Asai, S. Komori, T. Tamura, H. Sato, H. Takata, Y. Seguchi, T. Tokuda, and H. Terada: "A Self-Timed Clocking Design for Data-Driven Microprocessor," *IEICE Japan, Trans. on Electronics*, Vol. E74-C, No. 11, pp. 3357–3765 (Nov. 1991).

[43] S. Y. Kung: "VLSI Array Processors," *IEEE Acoustic, Speech and Signal Processing*, Vol.37, No.7, pp.4–22 (July 1985).

[44] 岩田, 内田, 西, 寺田: 多面的図的仕様記述環境の通信ソフトウェアへの適用, 第一回「通信ソフトウェアのための新しい方法論」WS 予稿集, pp.B-2-1–B-2-8 (1992-06).

[45] 岩田, 寺田: "データ駆動パラダイムによる図的仕様記述体系 AESOP," 第 49 回 情報処理学会全国大会予稿集, 4M-8, pp. 5–171–5–172, (1994-09).

[46] A. M. Davis: "A Comparison of Techniques for the Specification of External System Behavior," *Communications of the ACM*, Vol.31, No.9, pp.1098–1115 (Sep. 1988).

[47] 二見良治: "TQC に役立つ図形思考法," 日科技連, p.220 (1985).

[48] G. A. Miller: "The Magical Number Seven, Plus or Minus Two: Some Limits on Your Capacity for Processing Information," *Psychological Review*, Vol.63, No.2, pp.81–96 (1956).

[49] A. I. Wasserman: "Toward a Discipline of Software Engineering," *IEEE Software*, Vol.13, No.6 (Nov. 1996).

[50] T. Sumner: "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *IEEE Computer*, Vol.28, No.3 (Mar. 1995).

[51] 笠原, 唐沢, 種田, 岩田, 寺田: "AESOP の多面的な図的仕様記述法," 第 49 回 情報処理学会全国大会予稿集, 4M-9, pp. 5–173–5–174, (1994-09).

[52] S. L. Gerhart: "Special Issue on Formal Methods," *IEEE Software*, Vol.7, No.5 (Sep. 1990).

[53] 西川, 寺田: "履歴依存性を許すデータ駆動図式," 信学論 *(D)*, Vol.J66-D, No.10, pp.1169–1176 (1983-10).

[54] 寺田, 西川, 岩田, 岡本, 宮田, 小守, 嶋: "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 (1988-10).

[55] 長谷川, 雨宮: "データフローマシン用関数型高級言語 Valid," 信学論 *(D)*, Vol.J71-D, No.8, pp.1532–1539 (1983-08).

[56] C. A. Vissers et al.: "Architecture and Specification Style in Formal Descriptions of

Distributed Systems," *Proc. of the 8th International Workshop on Protocol Specification, Testing and Verification*, pp.189–204 (1988).

[57] L. P. A. Robichaud, M. Boisvert, and J. Robert: "*Signal Flow Graphs and Applications*," Prentice-Hall, p.214 (1962).

[58] J. L. Gaudiot: "Structure Handling in Data-Flow Systems," *IEEE Trans. on Computer*, Vol. C-35, No.6, pp.489–501 (June 1986).

[59] 岩田, 寺田: "機能メモリ主導型データ駆動アーキテクチャQ-FM とその評価," 情報処理学会計算機アーキテクチャ研究会, Vol. 95-ARC-113-29, pp. 225–232 (1995-08).

[60] M. Iwata and H. Terada: "Stream-Oriented Data-Driven Architecture with Functional Data Memory," *Trans. of Information Processing Society of Japan*, (submitted for publication).

[61] B. W. Boehm: "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Vol.6, No.3, pp.10–17 (Jan. 1984).

[62] H. Terada, Y. Xu, M. Iwata, T. Takine, and K. Murakami: "Flow-Thru Processing Concept and its Applications to Soft-Computing," *Proc. of 4th International Conference on Soft Computing*, pp. 105–108 (Oct. 1996).

[63] T. Shibata and T. Ohmi: "A Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operations," *IEEE Trans. on Electron Devices*, Vol.39, No.6, pp.1444–1455 (Jun. 1992).

[64] 唐沢, 笠原, 種田, 岩田, 寺田: "AESOP における仕様記述相互間の変換手法," 第 49 回 情報処理学会全国大会予稿集, 4M-10, pp. 5–175–5–176, (1994-09).

[65] 種田, 笠原, 唐沢, 岩田, 寺田: "AESOP におけるプロトタイピング手法," 第 49 回 情報処理学会全国大会予稿集, 4M-11, pp. 5–177–5–178, (1994-09).

[66] C. E. McDowell and D. P. Helmbold: "Debugging Concurrent Programs," *ACM Computing Surveys*, Vol.21, No.4, pp.598-622 (1989).

[67] 堀内一: "データ中心システム設計," オーム社, p.228 (1988).

[68] 石川: "アクティブデータベース," 情報処理, Vol.35, No.2, pp.120–129 (1994-02).

[69] "Problem Set for the Fourth International Workshop on Software Specification and Design," *Proc. of 4th International Workshop on Software Specification and Design* (1987)

[70] 二宮, 大塚, 和泉: "高品位テレビの衛星1チャンネル伝送方式–MUSE–," 信学論 D, Vol.J68-D, No.4, pp.647–654 (1985-04).

[71] R. Prieto-Diaz and P. Freeman: "Classifying Software for Reusability," *IEEE Software*, Vol.4, No.1, pp.6–16 (Jan. 1987).

[72] V. Vemuri: "Artificial Neural Networks: an Introduction," *Artificial Neural Networks: Theoretical Concepts*, IEEE Computer Society Press (1988).

[73] R. Tanase: "Distributed Genetic Algorithms," *Proc. of International Conference on Genetic Algorithms '89*, pp.434–439 (1989).

[74] 犬束, 水野, アルハッジ, 岩田, 寺田: "疎な結合を持つ相互結合型神経回路網の動的データ駆動型並列実現法," 第45回 情報処理学会全国大会予稿集, 7E-4, pp.2–51–2-52 (1992-10).

[75] 森, 岩田, 寺田: "並列遺伝的アルゴリズムとその動的データ駆動型実現法," 第49回 情報処理学会全国大会予稿集, 4H-6, pp.2-247–2-248 (1994-09).