

## 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

岩 田 誠<sup>†</sup> 寺 田 浩 詔<sup>†</sup>

筆者らは、要求定義水準の記述に半形式的な図的記述を導入し、以後データ駆動パラダイムに基づいて、これらの記述を実行可能プログラム生成の水準まで保存することによって、要求仕様記述水準でのソフトウェアの継承・保守を可能とする体系の確立を目指している。本稿では、図的援助を用いて半形式的に定義された要求仕様記述を忠実に継承し、これらに形式的な図的仕様記述を漸次付加することによって、最終的に実行可能なデータ駆動型プログラムを自然に生成する手法を提案し、この手法を実時間システム制御の例に適用し、具体的に述べる。

### Transformation Scheme for Diagrammatical Specification into Data-Driven Program

MAKOTO IWATA<sup>†</sup> and HIROAKI TERADA<sup>†</sup>

This paper describes a systematic software production and maintenance environment in which diagrammatically specified requirements could be converted to executable programs within the framework of the data-driven paradigm. In order to preserve required functions and behaviors of target systems through software specifications, various semiformal diagrammatical representations are adopted as natural requirement descriptions in the proposed specification environment. Furthermore, the diagrammatical software specifications are gradually refined by appending more detail definitions for data-structures and their processing algorithms and are then naturally transformed into executable data-driven parallel programs, since the data-driven paradigm is one of the most promising models with its natural representation capability of software specifications as well as its fine-grain parallel execution capability without any side-effects. This paper proposes a transformation scheme which transforms diagrammatical software specifications for history-sensitive systems which can not be represented within pure data-driven principle. This paper also concretely illustrated that the proposed scheme generates data-driven program from example diagrammatical software specifications for a realtime control system.

#### 1. はじめに

柔らかな要求仕様定義から始まり実行可能なプログラムの生成に至る、一貫的なソフトウェア生産・保守体系の構築は、ソフトウェア工学に課せられた、大きな課題である。

本稿では、要求仕様定義水準の記述に半形式的な図的記述を導入し、以後データ駆動パラダイムに基づいて、これらの記述を実行可能プログラム生成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のソフトウェアの継承・保守を可能とする手法を提案する。

すなわち、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。たとえば、(1)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、(2)このシステムに対する外部データ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、(3)システムに対する入・出力相互間の時間的關係を示す、シーケンス図などが広く用いられている。

本稿では、このような図的援助を用いて半形式的に定義された要求仕様をプログラム仕様記述の段階まで忠実に継承し、これらの半形式的な要求仕様定義に、形式的な図的ソフトウェア仕様記述を漸次付加して詳細化し、最終的に実行可能なプログラムを直接生成する手法の概要を述べる。

これまでに、ソフトウェア開発手法として、構造化分析・設計技法<sup>1)</sup>やジャクソン法<sup>2)</sup>などが提案され

<sup>†</sup> 大阪大学工学部情報システム工学科

Department of Information Systems Engineering, Faculty of Engineering, Osaka University

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<sup>9)</sup>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<sup>4)</sup>、
- 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は、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。

これに対して特に本稿では、要求仕様記述がソフトウェア仕様記述に継承された段階で、ソフトウェア実現のために必要な、形式的定義を順次付加することによって、データ駆動型プログラムが自然に生成できることを、実時間システム制御の例を用いて、具体的に明らかにする。

## 2. データ駆動パラダイムによる図的仕様記述体系 AESOP の概要

### 2.1 データ駆動パラダイムの特徴

データ駆動原理は、処理機能を表すノード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、「あるノードの全ての入力に（データの存在を表す）トークンが揃った時、そのノードの演算が実行可能になる」という単純な処理実行の原理である。したがって、(i)処理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、並列処理を素直に暗黙に表現でき、かつ、副作用のない並列実行を規定する<sup>5)</sup>。

これらの性質を利用して、これまでにも、データ駆動パラダイムによる次のような試みが個別に行われている。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、仕様記述する試みが多くなされている<sup>1)~3),6)</sup>。また、プログラム水準では、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<sup>7)</sup>。さらに、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の試作が進められている。筆者らも、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動的データ駆動型プロセッサ Qv-x を試作してその性能評価を進めている<sup>8),9)</sup>。

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行可能プログラムに変換されるため、仕様記述水準でデータ駆動型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。この試みの一つが図的仕様記述体系 AESOP (Advanced Environment for Software Production) であり、システム設計者が、要求仕様記述水準でソフトウェアの定義・検証を行え、かつ、可搬性のあるソフトウェアの継承・保守が可能になる体系の確立を目指している<sup>10),11)</sup>。

### 2.2 図的仕様記述の処理体系 AESOP の基本構成

本仕様記述処理体系では、図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

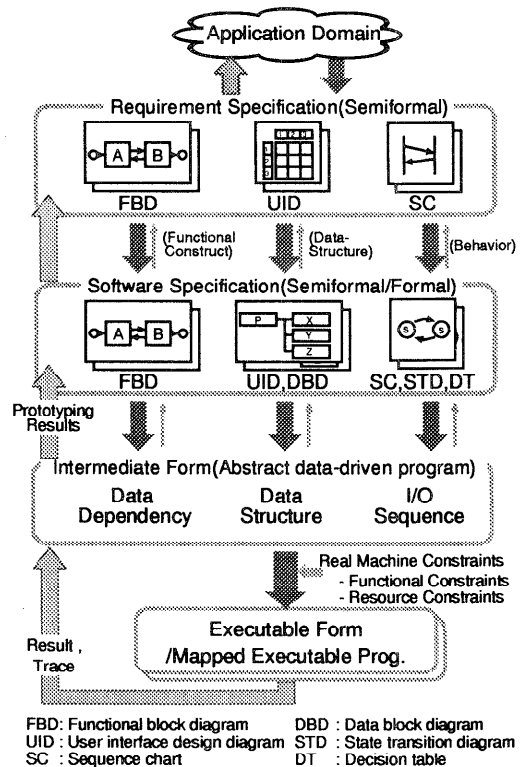


図1 AESOPのソフトウェア仕様記述処理体系

Fig.1 An overview of AESOP software specification processing system.

記述の情報をソフトウェア仕様水準へ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。プログラム仕様記述の水準では、これらの記述のうちソフトウェア実現される部分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ソフトウェア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う機能を実現している。さらに、本体系では、特定の実行機械の機能的制約 (Functional Constraints) や資源制約 (Resource Constraints) を考慮した実行可能プログラムを ADP から自動生成する方式をとっている。この実行機械上での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。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 を対象とすれば、一貫した仕様記述処理体系の構築が可能である。

図 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は、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 (Functional 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、システムの振舞を表現するシーケンス図 SC (Sequence Chart) を提供している。さらに、主としてソフトウェア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) を提供している。

### 3. 状態を伴う処理の図的仕様記述法と処理モデル

前章に述べた図的仕様記述体系 AESOP で一般的なシステムを定義するには、純粋なデータ駆動原理では規定できない履歴依存性のある処理を扱える、拡張したデータ駆動パラダイムを確立することが技術的な問題となる。本章以降に述べる、状態を伴う処理のプログラム生成手法は、アルゴリズムに関する履歴依存

処理である状態遷移処理を本データ駆動パラダイムに組み入れるための提案である。

#### 3.1 状態を伴う処理の多面的仕様記述手法

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

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

**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。

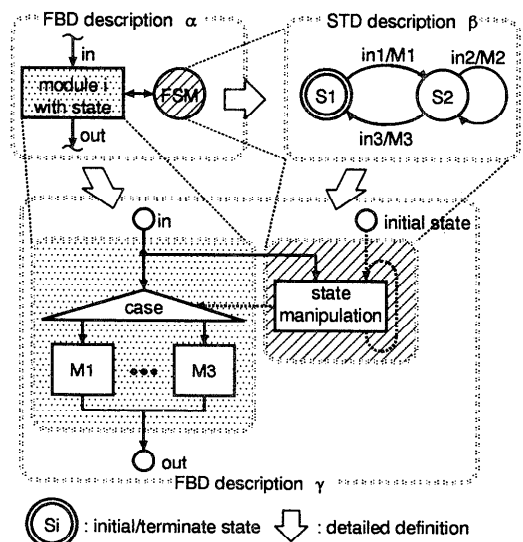


図 2 状態遷移処理の多面的図的仕様記述手法

Fig. 2 Multilateral specification method for the state transition process.

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

これらの仕様記述の表記法を用いれば、図 2 に示すように、状態遷移プロセスが多面的かつ階層的に定義できる。図 2 左上の FBD 記述  $\alpha$  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 2 右上の STD 記述  $\beta$  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 2 下の FBD 記述  $\gamma$  として定義される。

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

### 3.2 状態を伴う抽象データ駆動型処理モデル

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

本節では、前者の状態遷移処理を中心に、その解釈実行規則を提案する。

#### 3.2.1 ハイブリッド・データ駆動型実行原理

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

**定義 1 (Stream)** 線形順序を持つ要素からなり、すべての要素が必ずしもそろっている必要がないという性質を持つデータ構造をストリームと呼ぶ<sup>12)</sup>。 □

**定義 2 (Firing rule)** あるノードの全ての入力にトークンが揃い、かつ、入力アーク上の利用可能なトークン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。 □

定義 2 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。

#### 3.2.2 状態を伴うデータ駆動型処理構造

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

**定義 3 (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) は良形 (*well-formed*) である<sup>13)</sup>。 □

**性質 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 2** 良形な DDP では、定義 2 の発火規則を有する場合でも、性質 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 (以下、FSM 構造と呼ぶ) を図 3 に示すように定義し、これをトークンに関する生成・消費に着目した関数性を満たす処理ノードとして局所化する形態で状態を伴う処理を規定する。

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

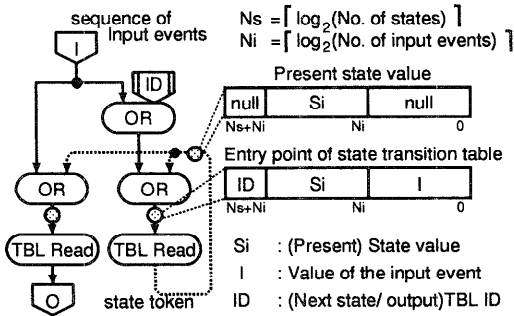


図3 FSMの抽象データ駆動型プログラム構造

Fig. 3 An abstract data-driven program construct for the finite state machine.

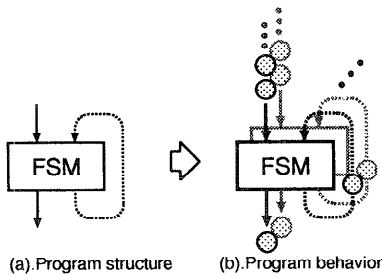


図4 FSM 構造の動的データ駆動型多重実行

Fig. 4 Dynamic data-driven execution of the FSM construct.

**性質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良形であれば、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。□

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

#### 4. データ駆動型プログラムの生成手法

本章ではまず、多面的かつ図的な AESOP 仕様記述からの抽象データ駆動型プログラムへの直接生成手法を提案し、本手法によれば、図的仕様記述からデータ駆動型プロセッサ  $Qv-x^{(8),(9)}$  上で高度並列に実行可能なプログラムが原理的に直接生成可能なことを示す。

さらに、本手法を基礎にすれば、多面的に定義され

た仕様記述相互間での変換を介した対話的な仕様記述支援環境が容易に構築できることを示す。

##### 4.1 抽象データ駆動型プログラム要素の生成

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

AESOP では、記述能力や抽象データ駆動型処理モデルの将来的な拡張性を残すために、プリミティブな図的記述要素に対する生成規則を定め、これらの規則により生成されたプログラムの構成要素を、その時点で既に生成されているプログラム情報に統合する手法を採用した。

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

**定義4 (Abstract Data-Driven Program)** 抽象データ駆動型プログラム  $ADP$  は、5つ組  $ADP=(N, P, A, ST, TBL)$  で表現される。

$N$ : ノード  $n_i$  の集合

$P$ : ポート  $p_i$  の集合

$A$ : アーク  $a_i$  の集合

$ST$ : ストリーム  $st_i$  の集合

$TBL$ : スタブ表 ( $STBL_i$ ) および状態遷移表 ( $STT_i$ ) の集合 □

**[変換手法 a] 機能とその間の接続関係の変換:** 機能ブロック図中の、モジュール  $m$ 、端子  $l$ 、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、データ依存アークの集合  $A$  として変換される。また、ファイルアクセスが明示される場合には、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] 振舞情報の変換:** 下位階層が未定義の機能モジュールに対して、その入出力の因果関係を抽出し、これからスタブの入出力情報 ( $Stub TBL$ ) を構成し、早期プロトタイピングを可能にする。さ

|       | 図的仕様記述       | 仕様記述の形式的定義                                                                                                                                                                                                               | 抽象データ駆動型プログラムADPの要素の生成規則                                                                                                                                          |
|-------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 機能    | 機能ブロック図<br>  | $FBD = (M, T, L)$<br>M: 機能モジュール $m_i$ の集合<br>T: 端子 $t_i$ の集合<br>L: リンク $l_i$ の集合                                                                                                                                         | $ADP = (N, P, A, ST, TBL)$<br>$M \rightarrow N$<br>$T \rightarrow P$<br>$L \rightarrow A$                                                                         |
| データ集合 | データブロック図<br> | $DBD = (D, O)$<br>D: (副)データ $d_i$ の集合<br>O: 順序関係                                                                                                                                                                         | $\begin{cases} d_i \rightarrow ai \\ D \rightarrow ai, sti \\ O \rightarrow \phi \\ O \rightarrow sti \end{cases}$                                                |
| 振る舞い  | シーケンス図<br>   | $SC = (M, D, S)$<br>M: 機能モジュール $m_i$ の集合<br>D: 信号線データ $d_i$ の集合<br>S: シーケンス $s_{ij}(d_i \rightarrow d_j)$ の集合                                                                                                            | $M \rightarrow N$<br>$\begin{cases} d_i \rightarrow ai \\ d_i \rightarrow sti \\ s_i \rightarrow STBLi \text{の要素} \\ s_i \rightarrow STTi \text{の要素} \end{cases}$ |
|       | 状態遷移図<br>    | $STD = (Q, X, Z, \delta, \omega)$<br>Q: 有限状態集合 ( $Q \ni q_i$ )<br>X: 入力集合 ( $X \ni x_i$ )<br>Z: 出力集合 ( $Z \ni z_i$ )<br>$\delta$ : 遷移関数 ( $Q \times X \rightarrow Q$ )<br>$\omega$ : 出力関数 ( $Q \times Z \rightarrow Z$ ) | $STD \rightarrow ni + STTi$                                                                                                                                       |

N: ノード  $n_i$  の集合  
P: ポート  $p_i$  の集合  
A: アーク  $a_i$  の集合  
ST: ストリーム  $sti$  の集合  
TBL: スタブ表 (STBLi) と状態遷移表 (STTi) の集合

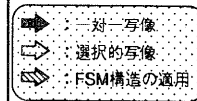


図5 図的仕様記述からの抽象データ駆動型プログラム要素の生成規則

Fig. 5 A set of generation rules for the diagrammatical specification into the abstract data-driven programs.

らに、状態が明示されるか、あるいは、状態が存在する可能性のあるシーケンス集合が検出された時点で、図3に示した処理構造からアクセスされる状態遷移表 (STT) として再構成する。

このように本手法では、個々のプログラム要素への1対1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、加法的変換 (インクリメンタルな変換) が可能である。また、新たな表現形式を追加した場合にも、その表現形式と抽象データ駆動型プログラムとの対応を生成規則として追加すればよく、既存の生成機構には何ら影響を与えない利点がある。さらに、本生成手法を基礎にして、各生成規則の逆変換規則を定義しさえすれば、後述する多面的仕様記述間の相互変換が容易に定式化できる。

#### 4.2 ADP 生成規則による対話的支援手法

対象システムの初期開発あるいは保守に限らず、利用者が当初から明確な要求仕様をイメージしていることは極めて稀であるため、利用者の仕様記述を完結する方向に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。

前節に述べた 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。例えば、図的仕様記述  $\alpha$  と  $\beta$  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。

このような特徴を活用して、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発生する観点から、

- 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、
- 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。

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

**機能** 機能の駆動条件の未定義・矛盾箇所 (巡回グラフ構造など)。

**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。

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

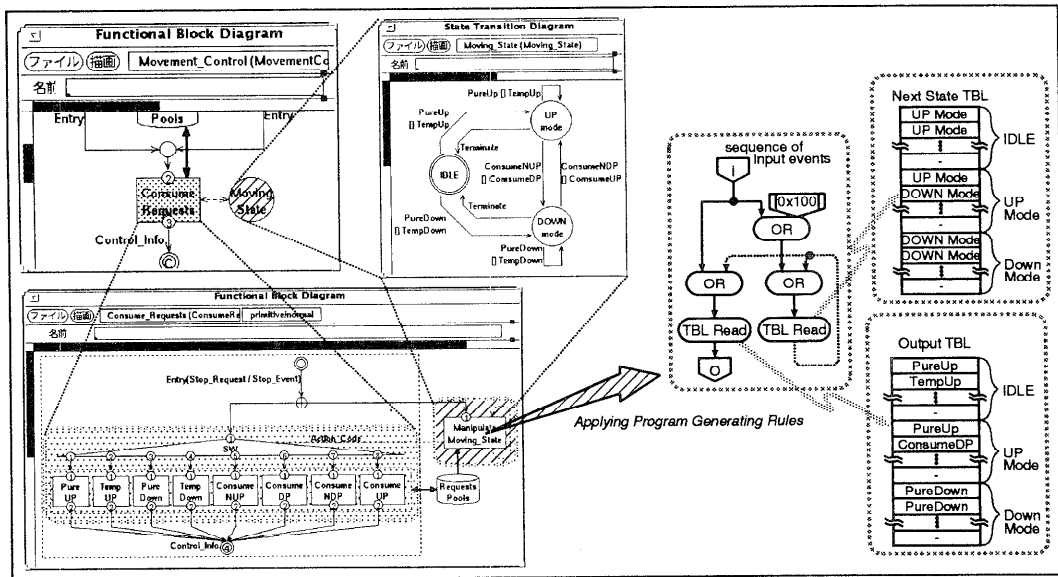


図6 リフト制御問題の仕様記述例とそのADPの生成例(一部)

Fig. 6 A partial example for specification and its ADP for the lift control problem.

#### 4.3 リフト制御問題への適用

AESOPの仕様記述手法ならびにプログラム生成手法の有効性確認のため、仕様記述のベンチマーク問題であるリフト問題<sup>14)</sup>を取り上げた。仕様化する際に留意した点は、(i)リフト稼働要求イベント群を対象とした典型的な生産者・消費者問題としてモデル化すること、ならびに、(ii)各リフトを制御する状態遷移処理が各リフトの状態とリフト稼働要求イベントに応じて同時並行に多重実行すること、である。

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

図6の右図は、この状態遷移モジュールの仕様から生成される抽象データ駆動プログラムの一部である。ここでは、図5の生成規則を適用して、図3に示したFSM構造を生成した例を示している。このプログラムは、データ駆動型プロセッサRAPID<sup>9)</sup>の命令セットへの変換、ならびに、タグ処理の挿入によって、多重実行可能なオブジェクトコードに容易に変換可能であることが確認された。

#### 5. おわりに

本稿では、データ駆動原理のシステム表現能力ならびに並列処理原理の自然さに着目して、半形式的な図的表現を許す要求仕様記述から高度並列に実行可能なデータ駆動型プログラムを生成する手法を、特に、状態を伴う処理に関して、提案し、本手法に基づく仕様記述支援手法の素案を示した。

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

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

一方、本稿に述べたような仕様記述環境では、性能対費用比の削減を支援する機能もまた重要である。このため我々は、実行機械の特性にチューンした実行形式プログラム部品の開発・蓄積を効果的に支援できる環境を提供するために、データ駆動型プロセッサシステムの視覚的評価環境の構成法もまた並行して検討している<sup>15)</sup>。

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

する。

## 参 考 文 献

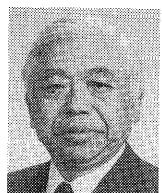
- 1) Ross, D. T.: Structured Analysis (SA) : A Language for Communicating Ideas, *IEEE Trans. Softw. Eng.*, Vol. SE-3, No. 1, pp. 16-34 (1977).
- 2) Jackson, M. A.: *System Development*, p. 418, Prentice Hall (1983).
- 3) Monarchi, D. A. and Puhr, G. I.: A Research Typology for Object-Oriented Analysis and Design, *Comm. ACM*, Vol. 35, No. 9, pp. 35-47 (1992).
- 4) Wyatt, B. B., Kavi, K. and Hufnagel, S.: Parallelism in Object-Oriented Languages: A Survey, *IEEE Softw.*, Vol. 9, No. 6, pp. 56-66 (1992).
- 5) 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).
- 6) Fuggetta, A.: A Classification of CASE Technology, *IEEE Computer*, Vol. 26, No. 12, pp. 25-38 (1993).
- 7) Shirasu, H.: Innovative Approach to Switching Software Design Using Dataflow Concept, *ISS, S-B4.3* (1987).
- 8) Terada, H., Iwata M. et al.: Superpipelined Dynamic Data-Driven VLSI Processors, Gaudiot, J. L., Bic, L. and Gao, G. R. eds., *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press (1995).
- 9) Komori, S. et al.: A50 MFLOPS Superpipelined Data-Driven Microprocessor, *Proc. ISSCC'91*, pp. 92-93 (1991).
- 10) 西川, 寺田 ほか: 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).
- 11) Iwata, M. and Terada, H.: Multilateral Diagrammatical Specification Environment based on Data Driven Paradigm, Gaudiot, J. L., Bic, L. and Gao, G. R. eds., *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press (1995).
- 12) 寺田, 西川 ほか: VLSI 向きデータ駆動型プロセッサ: Q-x, 信学論 (D), Vol. J 71-D, No. 8, pp. 1383-1390 (1988).
- 13) Dennis, J. B.: *Dataflow Schemas, Project MAC*, pp. 187-216, M.I.T. (1972).
- 14) Problem Set for the Fourth International Workshop on Software Specification and Design, *Proc. 4th International Workshop on Software Specification and Design* (1987).
- 15) 藤生, 岩田, 寺田: 動的データ駆動型プロセッサシステム Q-x の視覚的評価環境, 第 47 回情報処理学会全国大会論文集, 6 G-6 (1993).  
(平成 6 年 8 月 8 日受付)  
(平成 6 年 11 月 17 日採録)

### 岩田 誠 (正会員)



1986 年大阪大学工学部電子工学科卒業。1991 年同大学院博士課程単位取得後退学。同年大阪大学工学部情報システム工学科助手、現在に至る。データ駆動パラダイムを核とした、ソフトウェア環境および VLSI 向きデータ駆動アーキテクチャの研究に従事。電子情報通信学会, IEEE-CS 各会員。

### 寺田 浩昭 (正会員)



1933 年生まれ。1956 年愛媛大学工学部電気工学科卒業。1961 年大阪大学大学院通信工学専攻博士課程修了。同年大阪大学助手となり、講師、助教授を経て、1976 年同教授 (電子工学科)。1989 年から同情報システム工学科教授となる。1993 年から大阪大学大型計算機センター長を併任。交換機などの実時間高度並列処理向きの言語・アーキテクチャの研究に従事し、1989 年、電子情報通信学会業績賞、小林記念賞を受賞。電子情報通信学会副会長。IEEE シニア会員。電気学会、テレビジョン学会各会員。