ソフトウェア工学 121－17
(1998. 11. 6)

ストリーム指向システム開発のための対話的仕様記述環境

唐澤 圭† 岩田 誠‡ 村上 孝三† 寺田 浩詔‡

† 大阪大学大学院 工学研究科 情報システム工学専攻
〒 565-0871 大阪府吹田市山田丘 2 番 1 号
Phone:06-879-7804 Fax: 06-875-0506
E-mail: {karasawa, murakami}@ise.eng.osaka-u.ac.jp
‡ 高知工科大学 情報システム工学科
〒 782-8502 高知県香美郡土佐山田町宮ノ口 185 番地
Phone: 08875-3-1111 Fax: 08875-7-2220
E-mail: {iwata, terada}@info.kochi-tech.ac.jp

あらまし システムの構造と機能を直観的に理解できる図的な表現方法は、システムの機能仕様の多くに取り入れられている。しかし、現状では、プログラムとして構想し記述する段階で、問題の持つ構造を素直に反映することが困難な文章型記述への写像を余儀なくされる。本稿では、図的な仕様が最も効果的な分散型のストリーム処理を対象に、システムエンジニアが図的な仕様記述段を階的に詳細化することを助ける対話的な仕様変換手法を提案する。そこでは、部分的な仕様記述図が副作用なく解釈できる図的な仕様の特徴を利用した加法的変換手法を採用する。最後には、本手法を CG の位置処理を設計する過程を通じて、その実用性を例証する。

キーワード ストリーム処理、図的仕様記述、対話的環境、一対一解釈手法、部分的検証手法、3 次元ＣＧ処理

An Interactive Specification Environment for Stream-Oriented Processing

Kei Karasawa† Makoto Iwata‡ Koso Murakami† Hiroaki Terada‡

† Department of Information Systems Engineering, Graduate School of Engineering, Osaka University.
2-1 Yamadaoka, Suita, Osaka, 565-0871 Japan.
Phone: 06-879-7804 Fax: 06-879-7760
E-mail: {karasawa, murakami}@ise.eng.osaka-u.ac.jp
‡ Department of Information Systems Engineering, Kochi University of Technology.
185 Miyanoguchi, Tosayamada-cho, Kamigun, Kochi, 782-8502 Japan.
Phone: 08875-3-1111 Fax: 08875-7-2220
E-mail: {iwata, terada}@info.kochi-tech.ac.jp

Abstract Diagrammatic representations are commonly employed in most of the system function specifications, because it is important to intuitively comprehend the structure and functionality of the system. In current software development environment, however, programmers are usually forced to rewrite these specifications into parallel and distributed program codes having totally different structure by using a sequential-assignment-based language. This paper proposes an interactive transformation scheme whereby the system engineers can gradually refine diagrammatic specifications. The scheme proposed is described with special emphasis on its application to stream-oriented processing in which diagrammatic specifications are effective. By utilizing the feature of partial interpretation of diagrams, the scheme is based on an incremental manner. Finally, practicability of the methodology is illustrated through a design process of a geometric calculation for computer graphics.

key words stream-oriented processing, diagrammatic specification, interactive environment, one-to-one interpretation, 3D CG processing

1. はじめに

並列処理システムの生産性の向上は、並列分散型情報処理の一般化を背景に、非常に深刻な課題となっている。この課題に対応するため、我々は、図的な仕様に明示される並列性を直接継承した実行システムを生成する環境 AESOP (Advanced Environment for System Oriented Production) の研究・開発を進めている[1),2)]。本稿では、まず、データの到着順序が非決定的なストリーム指向の並列処理システムに対するプロトタイピングのための、図的仕様記述の局所的な解釈手法を提案する。そして、これを補助するために、仕様変更によるシステム挙動変化の検証結果を対話的に提示する手法を提案する。

手戻りの削減により生産性を高めるための有効な手段として、仕様の確認に極めて有効なプロトタイピングを中心にシステムを設計する成長型プロトタイピング手法がある。これを並列処理システムに適用した場合、設計者の構想したアルゴリズムと実現されたプロトタイプとの間で、処理の非決定的な挙動が一致している必要がある。これにより、図的な仕様記述に表れる本質的な処理構造が、結果として得られる並列実行プログラムに直接反映されれば、システム設計者は、了解性の高い図的な設計インタフェースを介して、システムの挙動を把握することが可能となる。これに加えて、並列処理システムの挙動の解析結果を仕様にフィードバックすれば、設計者は対話的にシステムを改良できるようになる。

現状のシステム開発環境では、システム仕様として構想し記述する段階においては、機能ブロック図やデータ構造図などの並列性を自然かつ明らかに表現できる図的な表現方法を用いている[4),5)]。しかしながら、プログラムとして実装する時には、問題の持つ構造を素直に反映することが困難な、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。更に、並列化コンパイルにおいては、データフロー解析によるプログラムリストラクチャリングなどを施して、問題と関係ないプログラム技術のためのアルゴリズムを除去し、並列性を抽出しなければならない。そのため、設計者の構想したアルゴリズムと実行システムの処理構造は、しばしば一致しない。このギャップを埋めるため、システムの設計の際に、経験豊富なシステムエンジニアの力を借りなければならない。

一方、ペトリネットや純粋なデータフローなどの並列処理モデルは、しばしばグラフ形式で表され、図的な機能表記と親和性が高い。この性質を活用し、図的な仕様記述から直接抽出した並列処理に関する情報から並列処理モデルに基づく中間プログラムを生成し、これを介して目的プログラムを生成する手法が提案されている。しかしながら、これまで提案された手法では、成長型プロトタイピングに必要な加法的なプログラム生成手法は提案されていない。また、ストリーム指向の並列処理システムが一様でない関係を持つデータ系列を入力する場合、データ値や型により処理を選択する必要があり、到達可能性の検証が困難であるという問題がある。

本稿では、ストリーム指向のアルゴリズムを拡張して、一様でない関係を持つデータ系列の要素を抽象データ型として表わせば、動的データ駆動プログラムを加法的に生成可能であることを示す。更に、要素間の関係を動的データ駆動プログラムの識別子 (タグ) 付きトークンへ写像すれば、到達可能性の検証が可能となることを示す。以下の章では、機能構造の典型的な記述法である機能ブロック図に加えて、ストリーム要素の抽象データ型を表現するために、オブジェクト指向開発設計の標準表記法となっている統一モデリング言語 (Unified Modeling Language: UML)[6)]に用いられているクラス図とオブジェクト図を用いて、ストリーム指向のシステム記述法を提案する。次に、これらの仕様記述に、一対一変換手法を用いれば、動的データ駆動プログラムに直接に変換できることを示し、変換された動的データ駆動プログラム上での検証法とその結果を対話的にフィードバックする手法を提案する。最後に、本手法を 3 次元 CG レンダリングに適用した際の AESOP システムの動作を説明し、その実用性を例証する。

2. ストリーム処理の仕様記述

機能ブロック図など図的な機能構造表現の特徴の一つは、データの流れに従ってアルゴリズムを素直に表現できることにある。このような機能構造表現は、了解性、部品凝集性、再利用性において優れた特徴を持ち、生産性を大きく向上させる効果がある。また並列処理の観点から見れば、図的な機能構造表現は、同時並行処理可能なモジュールを明らかに記述できる特徴がある。更に、この機能ブロックがストリーム・データにより通信すれば、データ並列[3)]の構造を利用して、パイプライン型の並列性も表せる。

これまでにもストリーム処理の研究は、信号処理[7)]、マルチメディア処理[8)]、事務処理[9)]など、広く行われている。しかしながら、これまでのストリーム処理対象は、ベクトル、行列、メッシュ、集合などの単純な規

則性を持った構造のデータに対して、同一処理を施すようなアルゴリズムに制限されている。

本章では、より複雑な構造のデータに対してストリームを適用する。その際、部品化と局所的解釈が可能となるよう、一様でない関係を持つデータ系列の要素に抽象化を施すことにより、本質的に必要な構造と変化する関係を分離して記述する手法を提案する。そのため、ストリームを同一のクラスに属する要素(オブジェクト)の集合とみなす。更に、オブジェクトの多相性を用いて、パイプライン型の並列性を明示したまま、各要素に異なる処理を施すアルゴリズムを採用する。このように、データ抽象の概念を採用すれば、データ間の依存関係と処理間の依存関係を別に記述できるため、仕様の了解性が高まる。また、データ中心のアプローチにより、部品とアーキテクチャの再利用性も高まる。

この抽象データ型の表記には、オブジェクト指向開発設計の標準表記法となっている UML を用いる。但し、標準の UML には解釈の曖昧性が含まれるため、制約キーワードを定義し、一意に解釈できるよう拡張する。また、オブジェクト間の関連には、オブジェクトのライフサイクルに関わるような高度な制約も定義できるが、本稿では、ストリーム処理に関わる処理の先行関連を規定する関連のみを対象とする。

2.1 ストリーム要素の関連

パイプライン型の並列処理を高位な仕様記述水準で利用できるようにするため、単純な規則性を持つ構造を対象に適用されていたストリームを、次のように拡張して定義する。すなわち、「ストリームとは、同一のクラスに属するオブジェクトの集合であり、その要素間には、処理の順序関連を定義することができる。」この定義により、例えば要素間に木構造の関連を定義すれば、ストリームはメッシュなどよりも複雑な構造を取ることができる。

このオブジェクト間の関連は、親オブジェクトに対応する子オブジェクトの数を基準に以下のように分類できる。

- 関連なし (集合型関連)
- 1 対 1 (線形順序関連)
- 1 対 n (木型順序関連)
- n 対 n (グラフ型順序関連)

本稿では、CG のデータ定義に必要なグラフ型順序関連以外の関連について、その表現法と処理の検証法を提案する。以下では、これらの関連を "Independent", "Linear Ordered", "Tree Ordered" と呼ぶ。図 1は、これらの関連を UML のクラス図を用いて定義している。

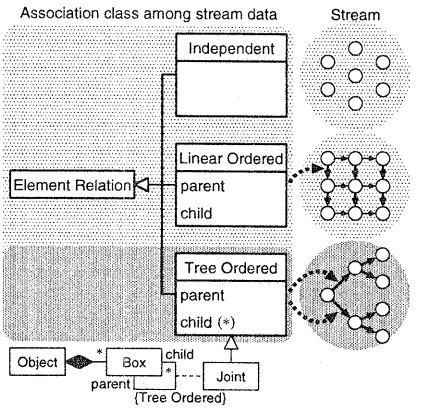

図 1　ストリーム要素間の関連クラスの分類

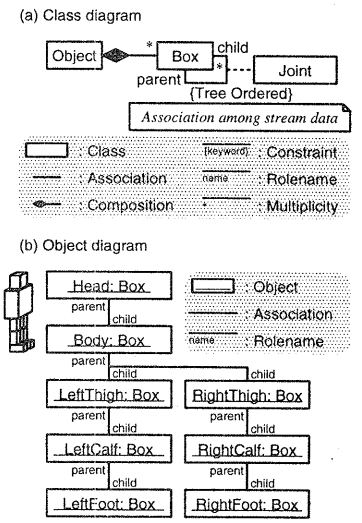

図 2　CG オブジェクトを対象としたストリームデータの定義例

"Independent" クラス以外は、parent と child の属性を持つが、"Tree Ordered" クラスでは、複数の child 属性を持つことができる。図 1 下に示されているように、図 2のクラス図の例では、クラス間の関連 "Joint" クラスは、"Tree Ordered" クラスを継承している。

図 2は、CG の box を組み合わせて作る CG object のクラス図 (a) と、その具体値としてロボットを表現したときの UML のオブジェクト図 (b) を示している。各オブジェクト box に対しては、Transform 計算、Light 計算、Appearance 計算、レンダリング処理などが共通

の処理である。そのため、各処理におけるオブジェクト間の依存関連を明示すれば、各処理を同時またはパイプライン型に並列処理できる。例えば、ロボットを動かすために、各 Box の Transform 計算するときには、上位ブロックからの相対位置を用いる。そのため、ロボットを構成する Box には、上から順に依存関連がある。しかし、左右の足は独立に計算できるため、それらのオブジェクト間には依存関連がない。オブジェクト図 (b) は、この木構造の関連を表している。

2.2 ストリームの表現と解釈

クラス間の関連の表記法は曖昧性が含まれており、ストリーム処理を定義するには、これを取り除く必要がある。そのため、クラス図とオブジェクト図の関連に制約を付ける。ここでは、クラス図に、ストリームに必要なデータ構造と要素間の関連情報を定義する制約として、以下の 2 つを用意する。

- 線形順序関連：組み立て型に集約されたクラス間の関連に "{Linear Ordered}"
- 木型順序関連：組み立て型に集約されたクラス間の関連に "{Tree Ordered}"
  ただし、集合型のストリームには、関連リンクが存在しない。

これにより、クラス図からは静的なデータ構造を抽出でき、オブジェクト図からはインスタンス毎の処理の先行関係、すなわちシステムの挙動を抽出できる。

図 3は、図 2の例から抽出されるデータ構造と処理の先行関係リストを表している。すなわち、図 2(a) のクラス図からは、図 3(a) のデータ構造を抽出できる。また図 2(b) のオブジェクト図からは、図 3(b) に表されたような、処理の制約関係を抽出できる。但し、ここではデータ構造に関わるクラス間の関連を組み立て型集約関連 (Composition)、制約関連 (Constraint) に限定する。その場合、データ構造の階層構造に対応する関連は、Composition のみである。

2.3 ストリームに対するシステムの挙動

ストリーム要素の定義から抽出される処理の先行関係リストは、システムの挙動を表している。図 4は、処理間のデータ依存関係を分類している。図 4(a) の処理モデルでは、ストリーム要素間の依存関係を利用しないため、下に描かれたように、同時並行に実行できる。図 4(b),(c) のモデルは、Shift 関数により、ストリーム要素間の依存関係を利用するため、下に描かれたように、処理に先行関係がある。入力データに依存関係を持つ図 4(b) のモデルは、出力データに依存関係を持つ図 4(c) のモデルより、並列性が大きくなる。

本表記法により、オブジェクト定義から、データの

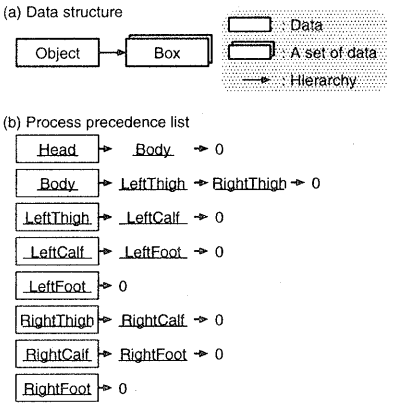

図 3　ストリームの定義からの抽出される構造と挙動の情報

入出力タイミングに関わらないシステム挙動を表現でき、並列処理の到達可能性の解析に必要な情報を抽出できる。

3. 対話的な仕様記述環境

仕様記述と並列処理モデルが連携していれば、ユーザは、了解性の高い仕様記述上でアルゴリズムの正当性を確認できる。そのためには、図的な仕様に表れるデータ依存関係を直接にアルゴリズムとデータ構造へ変換して、並列処理モデルに基づく挙動の検証結果を直接に図的な仕様上にフィードバックさせる必要がある。これには、仕様記述の構成要素と並列処理モデルの構成要素間の変換関数が、逆変換可能な関数でなければならない。ここでは、この用件を満たす、一対一の変換関数を導入する。

また、図的な仕様から並列処理モデルを生成して、その振舞いを検証する手法は、これまでにも数多く提案されている[10)~13)]。[10)]では、UML の図的表記から得られる情報を、ペトリネットへ変換し、振舞を検証する手法が提案されている。また、[11)]では、信号流れ図から同期的データフロー・プログラム[14)]を生成して、到達可能性を検証する手法が提案されている。しかしながら、前章で述べたストリーム処理では、要素間の関係に依存して動的に処理が変わるため、ペトリネットや同期的データフロー・プログラムでは、表現力が不足している。

本章では、データの到着順序が非決定的なストリーム処理の挙動を解析するためのタグ処理命令を導入した抽象データ駆動プログラム (ADP) を定義し、要素間の関係を ADP へ写像する変換法を提案する。また、

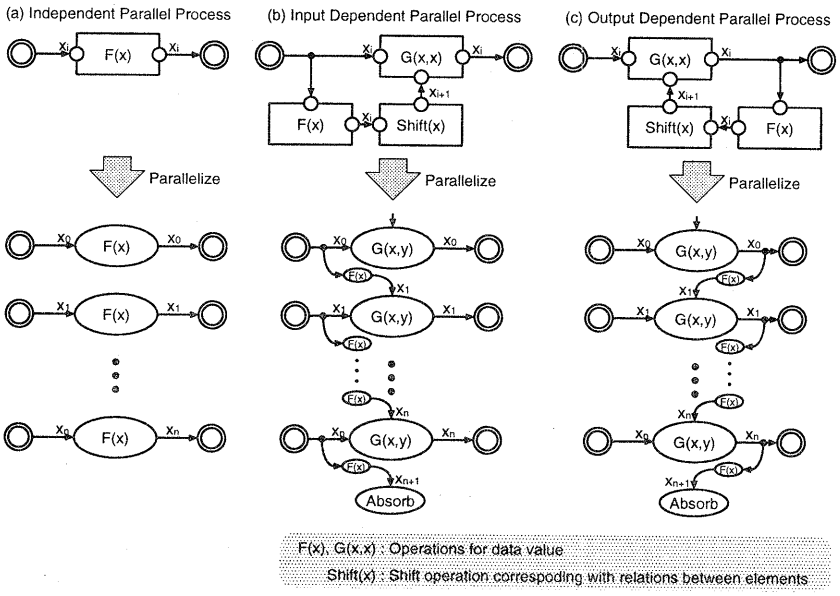

図4　ストリーム処理におけるシステムの挙動

仕様変更によるシステム挙動変化の検証結果を対話的に提示する手法を提案する。

3.1　抽象データ駆動プログラムの要素と解釈/検証法

抽象データ駆動プログラムは、ノード集合とポート集合とトークン集合の組である。その細かな要素と動作の定義は、本稿の対象外であるが、[15]で詳細を論じている。ここでは、ストリーム要素間の順序関係を表すタグ (TG) の構成を定義する。

TG は、オブジェクト間の関連を反映する、独立、線形、木型の3種のドメインから値を取る。独立と線形のタグ値については、[15]で定義している。木型の隣接関係を表すために、木のランク毎に独立な次元を用意する。すなわち、親オブジェクト毎に一つのタグ値空間を持ち、子オブジェクトが増える毎に、その中に値を取る。この方式を取れば、仕様上でのオブジェクト間の関連の変更に簡単に対応できるが、タグ空間に無駄な領域ができるため、実行形式に変換する際には最適化が必要となる。この最適化手法に関しては、今後の課題として残されている。

ストリーム処理の挙動は、タグ付きADP上での到達可能性と残留トークンの有無を検出することにより検証できる。これらの検出は、ADPにおいて、有限回のデータ駆動実行[15]により、全ての入力トークンが出力へ移動するか否かを検査すれば良い。このデータ駆動実行をタグ付きADPへ拡張するため、木構造のタグ処理命令を導入する。ここでは、図4の処理を表現可能にするタグ処理命令として、要素間の関係を辿る "Shift", 要素の追加/削除を行なう "Append"/"Delete",TGドメインを変更する "DomainTransfer" という写像関数を用意する。これらの関数を適用して、ストリーム処理において残留トークンを生じない条件は、処理構造にループがないこと、ストリーム要素の順序関係に子から親への関連がないこととなる。

3.2　追加的変換手法

仕様記述からデータ駆動プログラムを生成するため、仕様記述の要素からデータ駆動プログラムの要素への写像関数を定義する。図5は、仕様記述の要素とデータ駆動プログラムの要素をクラス図で表している。

仕様記述図の情報は、図5(a)に示されるように、データフロー図、クラス図、オブジェクト図とも、アイコンから構成される。そのアイコンには、Node, Port, Arc, Box, Line などの派生クラスがあり、さらに細かく分類される。一方、抽象データ駆動プログラムは、データフローグラフを表す Node, Port とデータを表す Token とその型を表す Stream から構成される。

これらの要素間を変換する関数を用意すれば、仕様記述から直接に抽象データ駆動プログラムを生成できる。これらの要素間の関係を表1にまとめた。

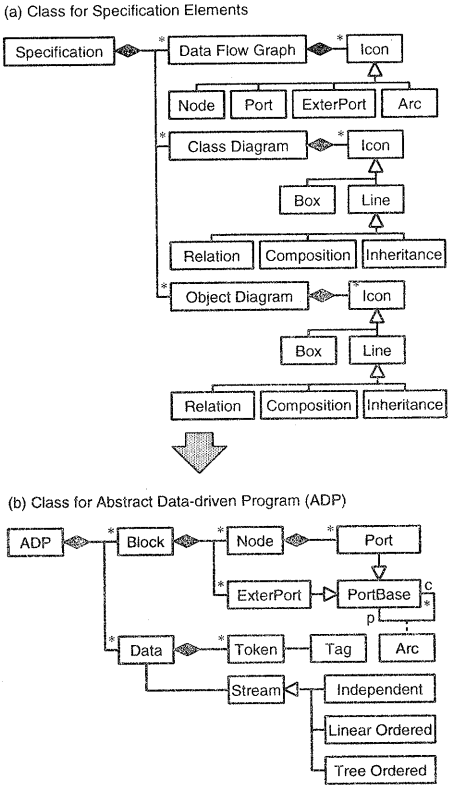

図5　仕様情報と抽象データ駆動プログラムの情報構成

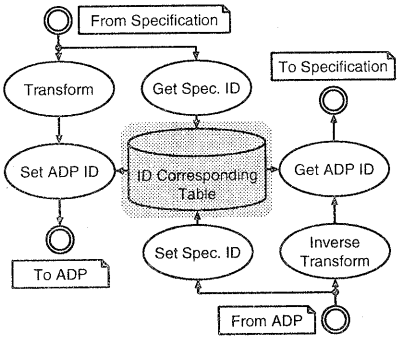

図6　変換/逆変換の処理構造

仕様記述要素	抽象データ駆動プログラム (ADP) 要素
Data Flow Graph	
Node	Node
Port	Port
External Port	External Port
Arc	PortBaseID, PortBaseID
Class Diagram	
Box	Data
Relation	Stream
Composition	DataID, DataID
Inheritance	—
Object Diagram	
Box	Token
Relation	Tag
Composition	—
Inheritance	—

表1　仕様情報と抽象データ駆動プログラムの変換表

3.3 検証結果のフィードバック

並列処理動作の誤りは、抽象データ駆動プログラム上のポートにトークンが残ることで見つけられる。よって、このポートに相当するデータフロー図のノードまたはポートと、トークンに相当するオブジェクトに警告を出せば、アルゴリズムの修正を促すことができる。但し、ストリーム要素の Shift 処理にファントムトークンが残る場合は、データフロー図の Shift 関数、クラス図の制約名、オブジェクト図の関連の全てに警告を出す必要がある。また、このどれかを変更する場合は、その変更を逆変換により、仕様へフィードバックし、異なる図の間での整合を取る。

このように、抽象データ駆動プログラムから仕様記述へフィードバックするためには、図6のように、仕様記述のアイコンの識別子と抽象データ駆動プログラムの識別子の対応を保存しておき、これを介して、情報を逆変換する。逆変換のパターンは、表1を右から左へ読めば良い。

4. 評価と実装

CG オブジェクトの位置計算処理に本手法を適用した例を図7に示す。図7では、2.3 節で述べた Shift 関数を破線矢印により表している。その結果、オブジェクトを全て独立なデータとするアルゴリズムと比較して、機能ブロック図に表れるノード数を 1 つに減らすことができた。これは、多態性を用いて処理構造の再利用性を高めた効果である。これは、ストリーム処理によるパイプライン型の並列性を利用するアルゴリズムとなっているが、線形順序関係のみの要素で構成されるストリームで表した時よりも、同時並行性を含むためより並列度の高いアルゴリズムとなっている。

本手法を実現するシステムを自己適用的に自律分散型に実現した。その機能構造を図8に示す。本システムは、利用者の記述に対し対話的に動作するため、エディ

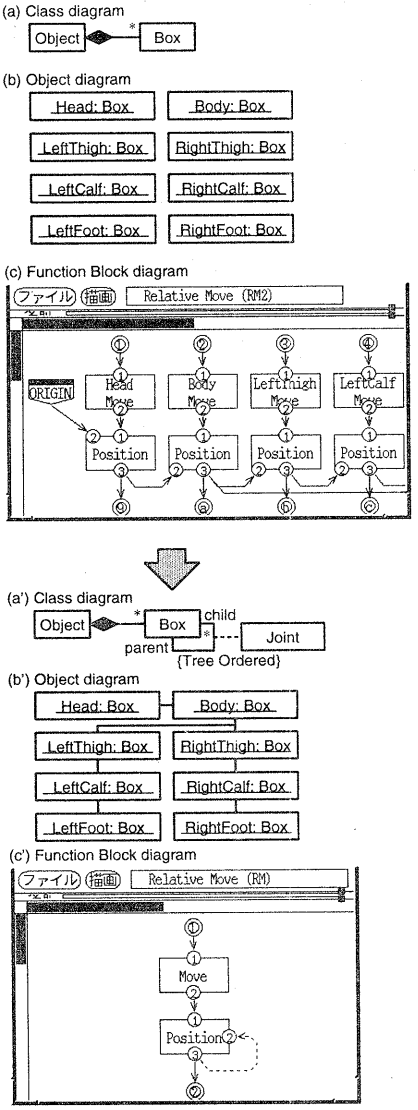

図7　CG位置計算におけるストリーム指向処理の例

タ上のアイコンを要素とするストリームに対する処理となる。そのため、図8右に、図4の Output Dependent Parallel Process Model が表れている。

5. おわりに

インスタンス間に関係を持つストリーム指向のプログラム解析は、単純な順序関係のみを対象にしていたが、その関係を拡張すれば、部品再利用度を高めることができる。本稿では、ストリーム指向の表現方法を用いた機能ブロック図、クラス図、オブジェクト図から得た情報を、データ駆動プログラム上で解析し、その結果を表現形式へフィードバックする手法を提案した。本手法を実装し、3D CG 処理に適用した結果、ストリーム処理構造の了解性を高めることが可能であることを示した。また、本手法を自己適用的に実装できることも示した。

本手法を拡張して、複雑なグラフ構造をもつ様なストリーム要素を対象にした場合、1つの要素は複数の要素に対して処理の依存関係を持つ。これには、図4のデータ依存グラフを変更し、処理 $G(x,y)$ が複数のデータが揃ってから駆動するように拡張して定義する必要がある。

また、ストリームの要素数が動的に変化するストリーム処理を実行する場合、実行中の動的な並列性の制御が必要となる。これには、実行マシンの状態をフィードバックする機構とそれによりストリームを調整する機構を持つハードウェアの開発を検討している。

謝辞

御議論頂いた AESOP 研究の関係各位に深く感謝する。なお、本研究の一部は、文部省科研費 (一般 B-2 05452363, 重点 知能の極限集積化) の援助によるものである。

参考文献

1) 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," 情処学論, vol.36, no.5, pp.1203–1210, May 1995.
2) 唐澤 圭, 新 吉高, 岩田 誠, 寺田 浩詔: "信号流れ図からのデータ駆動プログラムの対話的生成手法," 電学論 C, vol.116-C, no.11, pp.1307–1312, Nov. 1996.
3) W. D. Hillis and G. L. Steel, Jr.: "Data Parallel Algorithms," Communications of ACM, Vol.29, No.12, pp.1170–1183, Dec. 1986.
4) D. D. Hils: "Visual Languages and Computing Survey: Data Flow Visual Programming Language," J. Visual Languages and Computing, vol.3, pp.69–101, 1992.
5) K. S. Ellershaw and J. M. Oudshoorn: "Visualization Techniques for Various Programming Paradigms," Proc. IEEE TENCON'93, Beijin, China, pp.360–363, Nov. 1993.
6) 磯田 定宏: "オブジェクト指向モデリング," コロナ社, p.146, 東京都, Apr. 1998.
7) J. B. Dennis: "Stream Data Types for Signal Pro-

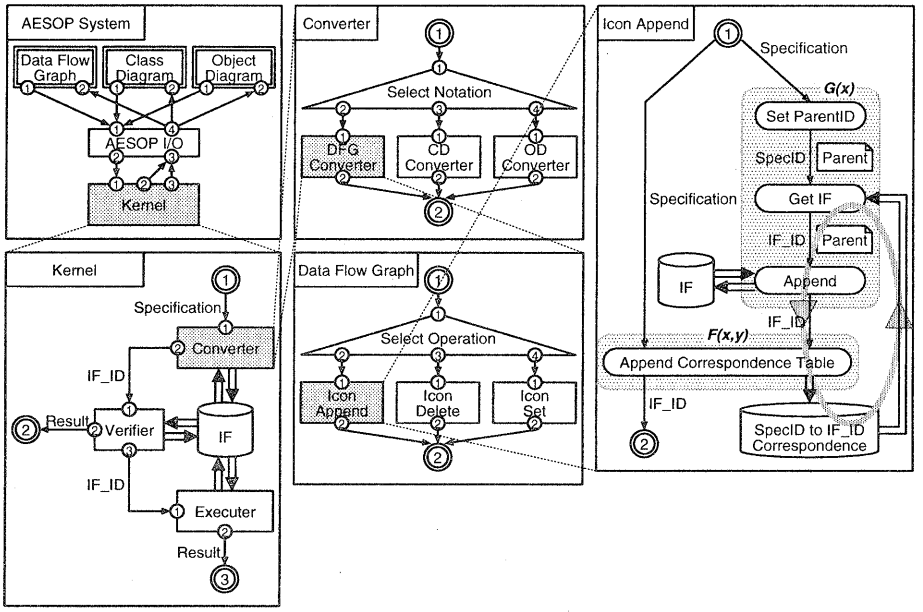

図8　AESOPプロトタイプシステムにおけるストリーム処理

cessing," in Advanced Topics in Dataflow Computing and Multithreading, ed, G. R. Gao, L. Bic and J. L. Gaudiot, pp.87–101, IEEE CS Press, Mar. 1995.

8) V. M. Bove, Jr.: "Multimedia based on object models: Some whys and hows," IBM Systems Journal, Vol.35, No.3&4, pp.337–348, 1996.

9) K. Kuse, M Sassa, I. Nakata: "Design and Implementation of a Stream Programming Language," Trans. IPS Japan, Vol.31, No.5, pp.673–685, May 1990 (in Japanese).

10) I. A. Coutts and J. M. Edwards: "Model-Driven Distributed Systems," IEEE Concurrency, Vol.5, No.3, pp.55–63, Jul./Sept. 1997.

11) E. A. Lee, W. Ho, E. E. Goei, J. C. Bier and S. Bhattacharyya: "Gabriel: A Design Environment for DSP," IEEE Trans. Acoustics, Speech and Signal, vol.37, no.11, pp.1751–1762, Nov. 1989.

12) K. Konstantinides and R. J. Rasure: "The Khoros Software Development Environment for Image and Signal Processing," IEEE Trans. Image Processing, Vol.3, No.3, pp.243–252, Mar. 1994.

13) A. Lendeczi, T. Bapty, G. Karsai, J. Sztipanovits: "Modelling Paradigm for Parallel Signal Processing," The Australian Computer Jour., vol.27, no.3, pp.92–102, Aug. 1995.

14) E. A. Lee and D. G. Messerschmitt: "Synchronous Data Flow," Proc. of the IEEE, vol.75, no.9, pp.1235–1245, Sept. 1987.

15) K. Karasawa, M. Iwata and H. Terada: "Direct Program Generation Scheme for Stream-Oriented Processing," Proc. the 1997 Int. Conf. on Parallel Architecture and Compilation Tech., San Francisco, California, pp.295–306, Nov. 1997.