アクタ・モデルを介したデータフロー・モデルによるオブジェクト指向の実現

大槻泰則† 榎本進††

並列処理の細粒度なアーキテクチャであるデータフロー・アーキテクチャは，アクタ・モデルに類似している．またオブジェクト指向には，その基礎になる唯一のモデルとしてアクタ・モデルがある．これらのモデルの近親性から，オブジェクト指向言語の処理系を実現する上で，データフロー・アーキテクチャを使用することが有効であると考えられる．本研究は，第1にオブジェクト指向言語のデータフロー・モデルへの翻訳方法について，提案を行うことを目的とする．第2にデータフロー・モデルによるオブジェクト指向の実現の適合性を検証する．また第3に本アプローチでの言語に具備されるべき機能を考察する．今回，並列オブジェクト指向言語を設計し，これをデータフロー・グラフに翻訳するトランスレータと，データフロー・グラフを仮想的に実行するシミュレータを用いて，実現方法の性能等の検証を行った．その結果，探索問題などの並列問題の計算について，本アプローチは有効であることが示された．

Implementation of Object-Oriented Language by Dataflow Model through Actor Model

Yasunori Otsuki† and Susumu Enomoto††

Dataflow architecture is similar to actor model. Actor model is an only basis model for object-oriented paradigm. From a point of view of parallelism and mechanism, dataflow approach is effective for implementation of object-oriented language. The purposes of this research are presenting a method of translation of object-oriented language into dataflow model, inspecting suitability of this implementation, and considering essential functions of the language in this approach. We designed a parallel object-oriented language and inspected efficiency of implementation by using a translator and simulator. It shows that this approach is effective for parallel problems such as search problems.

1. まえがき

並列処理の細粒度なアーキテクチャであるデータフロー・アーキテクチャの並列性とその駆動方式は，アクタ・モデル[7),9)]に類似している[3),5)]．またオブジェクト指向言語には，その基礎になる唯一のモデルとしてアクタ・モデルがある．また，並列性やデータの引渡し機構の点から，オブジェクト指向言語とデータフロー・アーキテクチャは近親性があるため[3)]，オブジェクト指向言語の処理系の実現において，データフロー・アーキテクチャの使用が有効であると考えられる．ところが，オブジェクト指向言語を分散処理環境で並列処理させる研究[6),12)]はなされているものの，これをデータフロー・アーキテクチャで処理させる方法については，オブジェクトの性質である記憶や階層の実現方法についての問題提起に留まっている[3)]．

われわれはすでに研究のあらましを発表しているが[13)]，本論文ではオブジェクト指向言語からデータフロー・モデルへの具体的な翻訳方法を提案する．そしてこれを実際に実現し，サンプルプログラムを用いて定量的な実験を行い，本翻訳方法の有効性を確かめた．以下2章では翻訳の概観について述べ，3章ではデータフロー・モデル，アクタ・モデル，オブジェクト指向モデルについて定義を与える．4章でデータフロー・モデルとアクタ・モデルの対応付けを行い，5章でオブジェクト指向モデルとアクタ・モデルの対応付けを行い，オブジェクト指向言語の処理系を実現するためのデータフロー用アクタ・モデルを定める．また6章ではデータフロー用アクタ・モデルによりオブジェクト指向言語の機能を実現するための方法について述べる．7章で今回設計した並列オブジェクト指向

† 三菱電機株式会社
Mitsubishi Electric Corp.
†† 東京理科大学理工学部情報科学科
Faculty of Science and Engineering, Science University of Tokyo

言語をデータフロー・マシン上へインプリメントするため，データフロー用アクタ・モデルを中間媒体とした翻訳方法を提示し，8章ではトランスレータとシミュレータを用いて，実現方法の性能等の検証を行った結果について報告し考察を行う．

用語については，主に情報処理学会誌，コンピュータソフトウェア学会誌[3)~9)]から引用する．

2. 翻訳の概観

オブジェクト指向言語をデータフロー・モデルへ翻訳するにあたり，あらかじめオブジェクト指向言語の生成規則について，アクタ・モデルを使用して記述した翻訳規則を定めておく．トランスレータは第1フェーズとして，オブジェクト指向言語の構文を解析し，翻訳規則に基づきデータフロー用アクタ・モデルへ変換する．続いて第2フェーズとしてデータフロー用アクタ・モデルからデータフロー・グラフ（またはデータフロー言語）への変換を行う．

今回提示するトランスレータの概観を**図1**に示す．

3. モデルの定義

本章では一般的なデータフロー・モデルとアクタ・モデルおよびオブジェクト指向モデルについての定義を与える．

3.1 データフロー・モデル

データフロー・モデルの計算は，ノードと呼ばれる処理単位間でデータが流れることにより，行われる．

ノード：加減乗除などの基本演算を実行するもの（細粒度[1)]の処理単位）をノードと呼ぶ．ノードは他のノードとは独立に（分散制御性[3),4)]）同時に（並列性）計算を行う．原則として，ノードは1つまたは複数の入力データに対する計算結果である出力データを，そのノードやデータを送ってきたノード以外の1つまたは複数のノードへ渡す．ノードは計算の実行中に生成・消滅することはない（非創生性）．

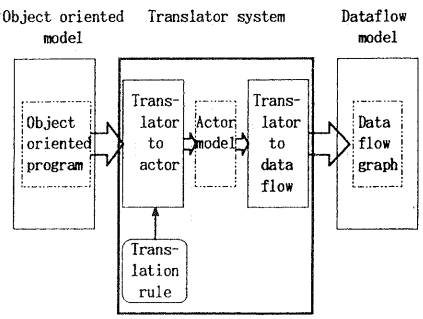


図1 トランスレータの概観
Fig. 1 A survey of translator system.

トークン：データはノード間でトークンにより授受される．1つのトークンは1データを伝達する．

アーク：トークンはノード間を結合しているアークを伝わっていくものとする．ここで，あるノードがアークを介して出力データを渡す対象となるノードを出力先ノードと呼ぶ．

発火：ノードが計算を行うために必要なトークンが，ノードに到着することにより計算が始まることを発火と呼ぶ．計算に必要なトークンは，ノードへ非同期に到着する（同時に到着するとは限らない）．発火のための条件を発火規則と呼ぶ．

3.2 アクタ・モデル

アクタ・モデルの計算は，アクタと呼ばれる処理単位間でメッセージを授受することにより，行われる．

アクタ：①計算や処理を行う部分，②データを格納する部分（記憶領域），③データそのもの，の3つを統合したものをアクタと呼ぶ．ここでアクタは基本演算が複数組み合わされたもの（粗粒度[1)]の処理単位）とする．アクタは他のアクタとは独立に（分散制御性）同時に（並列性）計算を行う．アクタはメッセージに従って計算を行い，アクタ内部の記憶領域の内容（アクタの状態）を更新することができる（履歴依存性[2),3)]）．アクタは，1つのメッセージに対する計算結果をメッセージを用いてそのアクタまたは他のアクタに渡す場合（双務性）と渡さない場合（片務性[5)]）の2種類の挙動を持つ．アクタは計算の実行中に生成・消滅することが許される（創生性）．

メッセージ：メッセージはアクタ間における計算の要求[9)]または計算の結果の返答[9)]である．メッセージは計算に必要なデータ（パラメータ）を含み，メッセージ内に継続[7),9)]（計算結果の送り先）を含むことができる．あるアクタは，そのアクタ自身にメッセージを送ることができる（再帰性）．メッセージは，そのパラメータとして計算に必要な複数のデータを含む．またアクタを含むこともできる（自己適用性）．

イベント：メッセージMのアクタTへの到着をイベントと呼ぶ．ここでTをターゲット[9)]と呼ぶ．

能動化：アクタが計算を行うために必要なメッセージが，アクタへ到着することにより，計算が始まることを能動化と呼ぶ．計算に必要な複数のデータはアクタへ同期的に到着する（同時に到着する）．計算は，アクタが受け取ったメッセージに対する挙動の記述であ

るスクリプト[9]に従い行われる．能動化のための条件を能動化規則と呼ぶ．

3.3 オブジェクト指向モデル

オブジェクト指向モデルの計算は，オブジェクトと呼ばれる処理単位間でメッセージを授受することにより，行われる．

オブジェクト：①計算や処理を行う部分，②データを格納する部分（記憶領域），③データそのもの，の3つを統合したものをオブジェクトと呼ぶ．ここでオブジェクトは，基本演算が複数組み合わされたものである．オブジェクトは，他のオブジェクトとは独立に（分散制御性）同時に（並列性）計算を行う．オブジェクトはメッセージに従って計算を行い，オブジェクト内の記憶領域の内容（オブジェクトの状態）を更新することができる（履歴依存性）．オブジェクトは，1つのメッセージに対する計算結果を，メッセージを用いてそのオブジェクトまたは他のオブジェクトに渡す場合（双務性）と，渡さない場合（片務性）の2種類の挙動を持つ．オブジェクトは計算の実行中に生成・消滅することが許される（創生性）．オブジェクトにはクラスとインスタンスがあり，インスタンスはクラスから生成された処理単位の実体である．

メッセージ：メッセージはオブジェクト間における計算の要求または計算の結果の返答である．オブジェクトは，このオブジェクト自身にメッセージを送ることができる（再帰性）．メッセージは，そのパラメータとして計算に必要な複数のデータを含む．またオブジェクトを含むこともできる（自己適用性）．

メッセージ・パッシング：メッセージをオブジェクトへ送ることをメッセージ・パッシングと呼ぶ．また，メッセージを受け取るオブジェクトをレシーバと呼ぶ．

能動化：オブジェクトが計算を行うために必要なメッセージが，オブジェクトへ到着することにより，計算が始まることを能動化と呼ぶ．計算に必要な複数のデータは，オブジェクトへ同期的に到着する（同時に到着する）．計算は，オブジェクトが受け取ったメッセージに対する挙動の記述であるソリッドに従い行われる．

階層性：クラス継承は，通常のモジュールの階層に類似している．クラスAのスーパークラスとしてクラスBを定義した（単一継承）場合，クラスAのオブジェクトはクラスBの機能を使用できる．

多態性：複数のクラスにおけるメソッド定義が同一インタフェースであっても，同じメッセージ・パッシングを行った際，レシーバの生成されたクラスの種類によって，そのメッセージは多様な挙動を起こす．これを多態性[10]（型多形態[5]）と呼ぶ．

4. データフロー・モデルとアクタ・モデルの対応付け

本章ではデータフロー・モデルを表現するアクタ・モデルを定める．このため，双方のモデルの構成要素を，**表1**に示すように対応付けることを前提とし，データフロー用アクタを定める．

4.1 粒　　度

モデルの定義の章で示したアクタは粗粒度であるが，本章では，データフロー・モデルのノードに対応する，基本演算を行う細粒度のアクタを定める．粗粒度のアクタは，このアクタの組み合わせと考えられるが，これについては翻訳方法の章で述べる．

4.2 履歴依存性

ノードは非履歴依存性であるが，アクタは履歴依存性である．ここでは，アクタの記憶機能の制限を行う．アクタは1基本演算について非履歴依存性（広義の非履歴依存性）であるものとする．すなわち，データフロー用アクタは1つの基本演算における，計算に必要な複数メッセージ（ノードに入力される複数のトークンに相当）が到着するまでの一時的記憶を持つものとする．

4.3 片務性／双務性

アクタは片務性と双務性の性質を持つ．またメッセージには要求と返答の2種類がある．一方，ノードは双務性であり，トークンにはノードに対する計算要求であるとか，ノードからの計算結果であるという区別がない．このため，計算結果の返答は出力先ノードに対する要求となる．またデータフロー・モデルでは，アクタ・モデルのように，計算途中で他のアクタへデータを渡してその結果を受け取り，再度データを他のアクタへ渡すような挙動はない．

アクタによりデータフロー・モデルのノードを表現

表1 モデルの対応関係
Table 1 Correspondence between the models.

データフロー・モデル	アクタ・モデル
ノード	アクタ
トークン	メッセージ
アーク	イベント
出力先ノード	ターゲット

するため，データフロー用アクタを次のように定める．原則としてアクタは複数のメッセージが到着することにより行われる1つの計算について，双務性（広義の双務性）を持つものとする．すなわち返答は1計算について必ず，かつその場合のみ行われるものとする．さらに返答と要求の区別はせず，アクタAがアクタBにメッセージを送った場合，アクタBは計算結果をアクタAへ送ることはないものとする．

4.4 創生性

データフロー・モデルとアクタ・モデルの創生性の違いについても調整する必要がある．

このため，アクタの創生性の制限を行い，データフロー用アクタは静的に存在するものとする．しかしオブジェクト指向モデルにおいて創生性がないことは，このモデルに基づいたオブジェクト指向言語の使い勝手を非常に悪くするものである．創生性の実現に向けての検討を，オブジェクト指向言語の機能の実現の章で行う．

4.5 能動化規則

データフロー・モデルの発火規則とアクタ・モデルの能動化規則には相違がある．この相違を**図2**に示す．

アクタの能動化規則の概念的拡張を行うことにより，データフロー・モデルをアクタ・モデルにより表現できる．ここでは，データフロー用アクタは，以下の挙動をするものとする．

①アクタは複数のメッセージに対し，計算に必要なメッセージがすべて揃うまで返答（他のアクタへの要求）を差し控えておく．

②アクタは複数のメッセージに対し，この入力を一時的に記憶しておく．

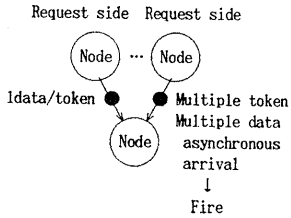
③すべてのメッセージが揃ったことを検知した時点でこれらに対し計算し，結果を他のアクタへ送る（広義の双務性）．

④アクタの内部状態をクリアする（広義の非履歴依存性）．

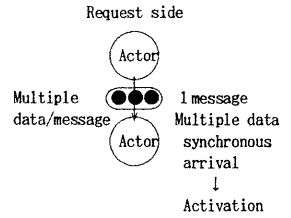
この概念的拡張を**図3**に示す．

4.6 データフロー用アクタ

前述の検討結果により定めた，データフロー用アクタの機能を**表2**に示す．



(a) Firing rule of dataflow model



(b) Activation rule of actor model

図2 発火規則と能動化規則
Fig. 2 Firing rule and activation rule.

表2 データフロー用アクタの機能
Table 2 Function of actor for dataflow model.

項目	アクタ	ノード	データフロー用アクタ
演算数	複数演算	単一演算	単一演算
状態保持（記憶）	永久的	一時的*1	一時的*1
返答／要求の区別	有	無	無
返答の有無	有または無	有*2	有*2
データ返答先	1アクタ	複数ノード*2	複数アクタ*2
データ要求先	複数アクタ	複数ノード*3	複数アクタ*3
生成・消滅	動的	静的	静的
計算の契機	1メッセージ	複数トークン	複数メッセージ*4
データ要求元	1アクタ	複数ノード	複数アクタ*5
データ到着形態	複数データ／メッセージ	1データ／トークン	1データ／メッセージ
データ到着タイミング	複数データ同期的	複数データ非同期	複数データ非同期

*1 1演算の結果生成まで状態保持
*2 演算の結果を返答とみなした場合
*3 演算の結果を要求とみなした場合
*4 1メッセージ群
*5 1アクタ群

5. オブジェクト指向モデルとアクタ・モデルの対応付け

本章ではオブジェクト指向モデルの構成要素に着目し，これらとアクタ・モデルの構成要素を**表3**に示すように対応付けることを前提とし，データフロー・モデルと間接的に対応付けることを考える．そしてオブジェクト指向モデルの機能を構成するため，データフロー用アクタを拡張したアクタ・モデルを提示する．

5.1 粒　　度

インスタンスにおけるメソッドは，オブジェクト指向モデルの変数や演算子をアクタとみなし，その間のデータ授受をアクタ間のメッセージとみなすことにより，アクタ・モデルを用いて表現できる．インスタンスをこのように表現することにより，インスタンス内のメソッドをデータフロー用アクタを用いて記述できる．

後述の翻訳規則を用い，インスタンス内をデータフロー用拡張アクタへ翻訳できる．ここでは翻訳規則により，インスタンスをデータフロー用アクタのモデルへ翻訳でき，さらにデータフロー・グラフへ翻訳できるという前提で議論を進めていく．

5.2 インスタンスの唯一性

あるインスタンスへの複数回のメッセージ・パッシングを，そのままアクタ・モデルにより表現する場合，同一インスタンスを示すアクタがシステム内に複数回現われる．インスタンスに履歴依存性や再帰性を持たせる場合，実体が1つだけ存在することが必要であり，また1インスタンスについて重複したアクタ・モデルによる表現は，データフロー・グラフへ翻訳する際，グラフの展開効率の低下を招く．

そこであるインスタンスへの複数メッセージに対し，そのインスタンスを表現するアクタを再使用させることにより解決を図る．すなわち，1つのインスタンスを表現するデータフロー・グラフを，システム内で1つだけとし，再使用可能とすることにより問題点を解決する．データフロー・モデルにおけるこの状態を**図4**に示す．これを実現するため，アークを複数のメッセージ・パッシング側ノードと接続する機構を設ける．

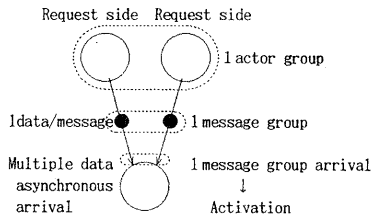


図3 能動化規則の概念的拡張
Fig. 3 Conceptual extension of activation rule.

表3 モデルの対応関係
Table 3 Correspondence between the models.

オブジェクト指向モデル	アクタ・モデル
インスタンス	アクタ
インスタンス間のメッセージ	アクタ間のメッセージ
メッセージ・パッシング	イベント
レシーバ	ターゲット

5.3 トークンの競合

アクタの再使用を許す場合，データフロー・モデルにおいて，トークンの競合[3]が新たに問題となる．

この問題を解決するため，「色付きトークン[3]」（タグ付きトークン[1]またはカラーリング[2]）の機能をインプリメント上の前提条件とする．トークンの色付けの範囲はインスタンスを表現するデータフロー・グラフ内とし，これによりトークンの競合が回避される．このため，アーク上のトークンの順序性の保持や同期制御を行わずに済み，複数のメッセージに対する計算を同時に行うことができる．さらに再帰的なメッセージ・パッシングが可能となる．この色付きトークン方式は，データフロー・グラフの一時的な疑似的複写とみなすことができる．

5.4 メッセージ・パッシングの同期

オブジェクト指向モデルのメッセージには複数のデータが包含されていてよい．このためインスタンスを表現するアクタへ複数のデータを同時に渡すことが必要になる．この場合，データフロー・モデルにおいて，同一データフロー・グラフへの非同期なトークン

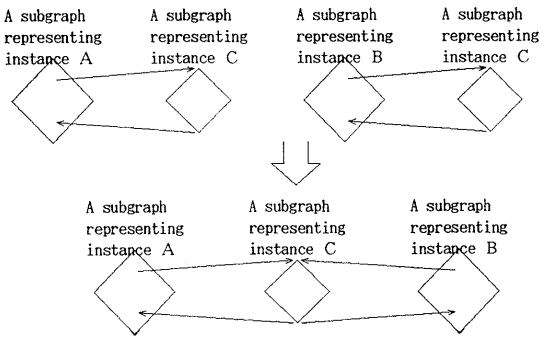


図4 データフロー・グラフの再使用
Fig. 4 Reuse of dataflow graph.

の到着によるトークンの競合が発生する．この理由は複数のトークンの間での同期がとれていないためである．

そこでデータフロー用拡張アクタに対するメッセージは，複数のデータを渡すことができるものと決め，データフロー・モデルにおいては，「構造化トークン[3),4)]」という複数のデータを1つのトークンにまとめたものを導入することをインプリメントの前提条件とする．

5.5 データフロー用拡張アクタ

前述の検討結果から導き出された，データフロー用拡張アクタの機能を**表4**に示す．

6. オブジェクト指向言語の機能の実現

本章では，オブジェクト指向言語をデータフロー・モデルにより実現するため，データフロー・モデルに必要となる機能ならびに実現方法について述べる．

6.1 履歴依存性

記憶機能を実現する上で生ずる問題点として，記憶領域の競合，排他制御を導入することによるデッド・ロックの発生，記憶資源獲得の制御などがある．今回実現する記憶機能は，以下のとおりである．

処理の干渉：あるメッセージに対する処理が実行している最中に，新しいメッセージによりその処理が行われる「干渉[6)]」を許すものとする．

共有変数：インスタンス変数は，オブジェクト指向言語の必要条件であるのでサポートし，インスタンス変数へのアクセス方式（排他／非排他制御）をユーザに選択させることにより，競合問題を解決する．またオブジェクト間の共有変数は，「オブジェクト間のデータ授受はメッセージだけである」という原則に反し，本来のオブジェクト指向ではないと考え，本実現ではサポートしない．

6.2 記憶機構

インスタンスにおける履歴依存性をデータフロー用拡張アクタを用いて実現する方法が問題となる．

ここで，機能的にあるインスタンスの一部または機能を実現するための下位モジュールに相当するインスタンスを，「サブインスタンス」と呼ぶ．すると記憶領域はインスタンスに密着して一連の環境を表現するサブインスタンスとみなせる．この場合，インスタンス内から記憶機能を実現するサブインスタンスへのアクセスは，メッセージを介して行える．この記憶機構と動作は**図5**と**図6**に示すように，データフロー・グラフ上をループするトークンにより実現できる．ここでENTRは複数の入力アークを持つノードで，EXITはENTRの入力アークに対応する出力アークへデータを出力する．DCOLは後述の下位色を脱色するノードである．また記憶領域へアクセスする方法を**図7**に示す．

本方式は①一般に認知されているデータフローの仕組みを用いて実現できる，②記憶領域へのアクセスの排他/非排他制御の機能を備えても，現実的なノード数で実現できる，という理由で最も今回の実現に適していて，かつ正統なデータフロー・モデルに従っていると考えられる．

6.3 制御機構

記憶機能と制御の問題として2つの解決すべき問題

表4 データフロー用拡張アクタの機能
Table 4 Function of extended actor for dataflow model.

項　　目	オブジェクト	データフロー用アクタ	データフロー用拡張アクタ
演 算 数	複数演算	単一演算	単一演算
状態保持（記憶）	永 久 的	一時的*1	一時的*1
返答／要求の区別	有	無	無
返答の有無	有または無	有*2	有*2
データ返答先	1アクタ	複数アクタ*2	複数アクタ*2
データ要求先	複数アクタ	複数アクタ*3	複数アクタ*3
生成・消滅	動　　的	静　　的	静的（疑似動的）
計算の契機	1メッセージ到着	複数メッセージ*4	複数メッセージ*4
データ要求元	1アクタ	複数アクタ*5	複数アクタ*5
データ到着形態	複数データ／メッセージ	1データ／メッセージ	複数データ／メッセージ
データ到着タイミング	複数データ同期的	複数データ非同期	複数データ同期的

*1 1演算の結果生成まで状態保持
*2 演算の結果を返答とみなした場合
*3 演算の結果を要求とみなした場合
*4 1メッセージ群（同色のもの）
*5 1アクタ群

がある．第1に，データフロー・モデルにおいては，データが揃い次第計算を始める先行評価[3]という性質がある．このため，オブジェクト指向モデルをデータフロー・モデルにより実現した場合，入力パラメータを伴わないメソッドは，その呼び出し前に実行される．結果としてユーザの意図しない状態の更新が行われることがある．第2に，メソッド間で記憶領域を介してのみ互いにインタフェースをとる場合，メソッドが互いに並列に動作するため，メソッド間で正しいデータのやり取りが行われなくなる場合がある．

対応策として，第1の先行評価の問題については，パラメータ・データが揃ってもメッセージ・パッシングが実行されないようにし，メソッドが起動したことをトリガとしてパッシングを行う機構とする．一方条件文では then パート，else パートの起動を，各パートが数式のとき先行評価，メッセージ・パッシングのとき遅延評価（条件評価後に評価）により行う機構とする．第2の問題については，ある条件が揃った時点で次の計算を始めることを許す同期指示の機能をメッセージ・パッシングの構文に持たせる．これはデータ依存関係を制御依存関係に変換する機能を持つ．

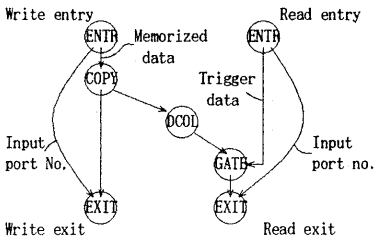


図 5 記憶機構
Fig. 5 Memory mechanism.

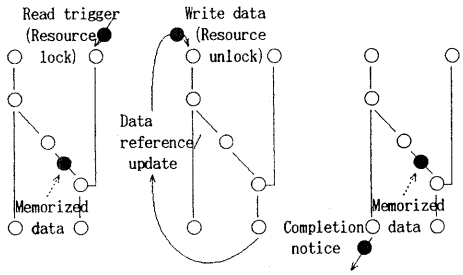


図 6 記憶動作
Fig. 6 Memory actions.

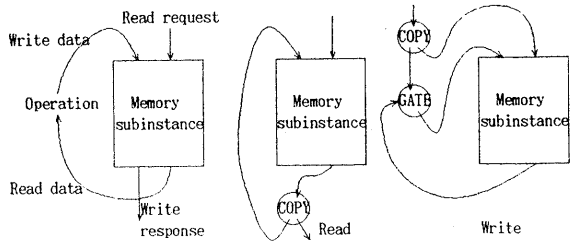


図 7 アクセス方法
Fig. 7 Access methods.

6.4 片務性／双務性

オブジェクト指向モデルにおいて，一般に記憶機能を有するオブジェクトへメッセージを送った場合，それに対する返答は必要ない．このため，記憶機能を持つモデルにおいて，片務性の実現が必要となる．

具体的な実現機構としては，メッセージによる要求に対しては従来どおり何らかの返答を返すものとする．この理由は，色の使用後もその色で色付けされたまま残っているトークンである「ファントム・トークン[3]」の発生を抑止するためである．そこで要求側には返答されたトークンを廃棄する機構を設ける．

6.5 創 生 性

インスタンスの創生性を実現する上での問題は，クラスを表現するデータフロー・グラフをインスタンスを生成するごとに動的に複写する方法である．

インスタンスの区別は，インスタンスの状態（記憶領域の内容）の区別であると考えられる．したがってインスタンスの生成ごとに一意に色を獲得し，インスタンスの初期状態として，その色のトークンを記憶領域の部分にセットすることにより，インスタンスが新たに生成されたことになる．

そこで色の階層化を導入し，その上位色はインスタンス自身を意味し，下位色は前述のメッセージごとの現在の処理の色として，インスタンス内で並列化された処理単位を意味するようにする．ノードの発火は，例えば2つのトークンにおいて，上位色が一致し，かつ下位色も一致しているときに発火する「同色発火」とし，記憶機能の部分では，上位色のみが一致しているときに発火する「系統色発火」を適用し，同一インスタンスの異なった処理単位間でのデータの受渡しを実現する．この機構により，クラスと各インスタンスを表現するデータフロー・グラフの多重化が実現できる．この概念を図8に示す．ここでは上位色が青であるインスタンス1と上位色が赤であるインスタンス2の

トークンがともにメソッド部と記憶部を表現しているデータフロー・グラフ（クラスの実体）上を重畳して（共用して）流れている状態を例示している．さらにあるインスタンス内で同時に処理されるメソッドについては，その下位色が異なる3種類のトークンがともにデータフロー・グラフを流れている状態を示している．

6.6 再帰性

再帰性を実現する上で，色付きトークン方式は有用であるが，再帰するインスタンスにこのインスタンスの所在を認識させる方法が問題である．これについては自己適用性の箇所で解決する．

6.7 自己適用性

インスタンスAをメッセージ・パッシングによりインスタンスBへ伝達する機能を実現する場合，第1に，生成されたインスタンスを保持するための機構，第2に，あるインスタンスの存在を知ったときのそれに対するメッセージ・パッシングの仕組みが必要となる．

今回，第1の問題については，変数と構造化トークンを使用してインスタンスの保持を行い，構造化トークンの中身をクラス名とインスタンスの色のペアで表現する．その構造化トークンをメッセージのパラメータとすることにより，他のインスタンスへインスタンスを送り，アクアインタンシス[9]（アクタが知りうるアクタ）の実現を図る．このため，構造化トークンを操作するノードとして，構成ノード／分解ノードを用意する．第2の問題については，CALL ノード[3),4)]を使用したノード間の動的結合の機構を採用する．これは CALL ノード，Symbol ノード，および RETN ノードで実現される．**図9**はこの概念図である．今，クラス cls のインスタンスを変数 var が保持していて，このインスタンスへメソッド名 msg，パラメータ para なるメッセージを送る場合，メッセージ式は var msg : para となる．このとき，インスタンスは構造化トークン〈cls, color〉で表され，メッセージは構造化トークン〈msg, para〉で表される．CALL ノードはこれらのトークンにより発火し，シンボルノード cls へトークンを送り，RETN ノードは送り元へ計算結果を返す．

6.8 階層性

クラス C1，C2 のスーパークラスが同じである場合，クラス C1，C2 ごとにスーパークラス部分をデータフロー・モデルへ翻訳することは，データフロー・グラフが膨大な量になるため，現実的な翻訳方法ではない．

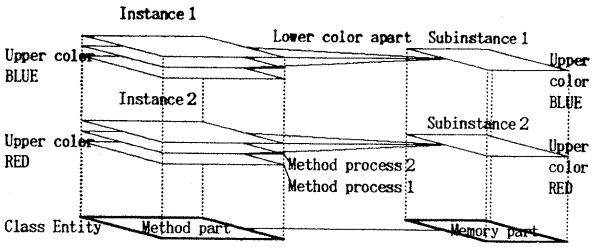


図 8 クラスのデータフロー・グラフの多重化
Fig. 8 Multiplized class dataflow graph.

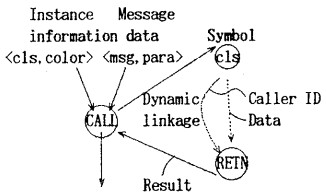


図 9 グラフの動的結合
Fig. 9 Dynamic linkage of graph.

このため，スーパークラスを表現するデータフロー・グラフを複数のクラスにより共用する方法を採る．インスタンスが生成されたとき，インスタンスが属するクラスと当該クラスのスーパークラスのデータフロー・グラフ双方の記憶機構に，当該インスタンスの色の初期データのトークンを送る．またクラス内では，どのメソッドにも該当しないメッセージは，スーパークラスへデレゲーション[8]（メッセージを他のクラスへ委任して処理させること）するようにしておく．これは創生性の実現で行った，クラスのデータフロー・グラフの多重化により実現できる．**図 10** に階層性の実現の概念図を示す．ここでは例としてクラスCのスーパークラスがクラスB，クラスBのスーパークラスがクラスAである場合，クラスCのインスタンス1がスーパークラスで定義されたメソッドを実行するために，次々と上位のクラスへメッセージをデレゲーションしている様子を示している．このときクラスA，Bにおいては，おのおののクラスのインスタンスが1つずつ創生されたように見える．

7. 翻訳方法

本章での主な論点は，データフロー・モデルを表現する細粒度のアクタを用い，クラスを表現する粗粒度のアクタを表現する方法である．まず，インプリメントするオブジェクト指向言語を，Smalltalk のサブ

セットに近いものとし，言語は複数のクラス宣言を持ち，この中にスーパークラス宣言，インスタンス変数宣言，複数のメソッド宣言を記述する．メソッドを逐次型言語のような文列で定義し，文と文のデータの依存関係に反しない範囲で各文が並列に実行されるものとする．文と文のデータの依存関係は文の中の変数に基づく．文は代入文，条件文，メッセージ文，リターン文（メソッドの計算値を返す）などがある．ここでの翻訳はオブジェクト指向言語を入力とし，アクタ・モデルのイベント列を出力するものである．イベントは C〈== request : [A] reply-to : [B] なる形式で表され，メッセージを送るアクタ，継続アクタがおのおのアクタA，Bであるメッセージがターゲット Cへ到着することを示す．

最初に説明のために，**図 11** のようなオブジェクト指向言語の代入文の生成規則を例にとる．翻訳の過程として，初めに解析木を生成する．ここで V_N, V_T の各要素をアクタと考えると，解析木に基づきアクタ同士のデータ依存関係が求められる．例えばある代入文の解析木が**図 12**（a）のような形のとき，文の値は左辺の変数の値であるので，その文に対応するアクタ（var, Expr, + など）の依存関係は図12（b）のようになる．実際の翻訳においては，解析木を上昇型に構文主導型翻訳[11]をし，アクタ間の依存関係を求める．ここで用いる翻訳規則は，アクタ・モデルの表記法を用いて**図 13** のように表すことができる．

次にクラスの翻訳であるが，インスタンス変数については，それに対応する翻訳規則に基づきイベント列を生成する．メソッド中の文列の翻訳は，上記の代入文の翻訳に基づくが，文に含まれる変数による文と文の依存関係の順番で行う．**図 14** に翻訳手順の概念図

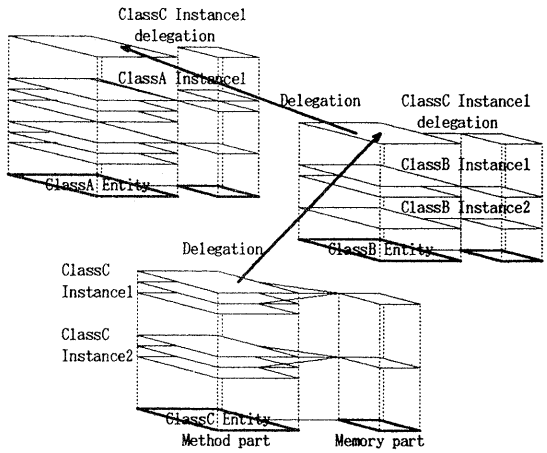
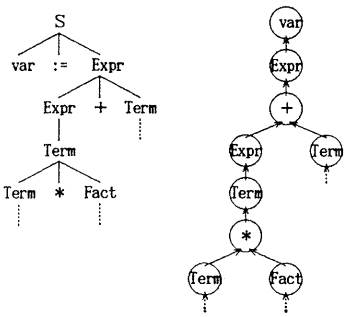


図 10 階層性の実現方式
Fig. 10 Implementation architecture of hierarchy.

G = (V_N, V_T, P, S)
V_N= {S, Expr, Term, Fact}
V_T= {var, cnst, :=, +, *}

P: S →var := Expr
Expr→Expr + Term
Expr→Term
Term→Term * Fact
Term→Fact
Fact→cnst
Fact→var

図 11 代入文の生成規則
Fig. 11 Production rules.



(a) Parse tree (b) Dependency of actors

図 12 代入文の解析木とアクタ
Fig. 12 The parse tree and actors for an assignment statement.

S →var := Expr	var <==request:[Expr] reply-to:[Φ]
Expr→Expr + Term	+ <==request:[Expr] reply-to:[Expr]
	+ <==request:[Term] reply-to:[Expr]
	Expr<==request:[+] reply-to:[Φ]
Expr→Term	Expr<==request:[Term] reply-to:[Φ]
Term→Term * Fact	* <==request:[Term] reply-to:[Term]
	* <==request:[Fact] reply-to:[Term]
	Term<==request:[*] reply-to:[Φ]
Term→Fact	Term<==request:[Fact] reply-to:[Φ]
Fact→cnst	Fact<==request:[cnst] reply-to:[Φ]
Fact→var	Fact<==request:[var] reply-to:[Φ]
Production rules	Semantic rules representing dependency of actors

図 13 翻訳規則
Fig. 13 Translation rules.

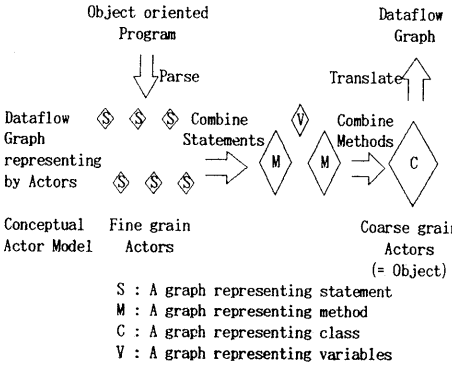


図 14 翻訳手順
Fig. 14 Translation process.

を示す．この図で示している手順は次のとおりである．①入力されたオブジェクト指向プログラムの文を上記手順でアクタ・モデルで記述された文を表すグラフSへ翻訳する．②インスタンス変数を表すグラフVを生成するとともに，文と文の依存関係に基づき複数のSを結合し，メソッドを表すグラフMを生成する．③V，Mの参照関係に基づき，クラスを表すグラフCを生成する．④グラフをデータフロー言語へ置き換える．

8. 実験と考察

8.1 実験内容と結果

今回設計した言語と翻訳方法について評価を行うため，プログラムをデータフロー・グラフに翻訳するトランスレータと，データフロー・グラフを入力として実行する仮想データフロー・マシンを定め，そのシミュレータを作成した．

評価は各評価項目について測定するためのベンチマーク・プログラムと，定性的評価のためのアプリケーション・プログラムを使用して行った．仮想データフロー・マシンのノードの処理時間は，汎用マイクロプロセッサの演算実行時間を参考に，**表5**のとおり定めた．ここで1単位時間を1クロックと呼び，逐次処理を行った場合と並列処理を行った場合について実行完了までのクロック数をシミュレートした．

機能別分析を行うためのベンチマーク・プログラムの実行結果を**表6**に示す．ここで展開ノード数とは機能実現のために生成されるノード数で，実行ノード数とは機能実現のために実行されるノード数をいう．並列クロックとは仮想データフロー・マシンで機能を実行したときに要する時間（実時間に相当）で，逐次クロックとは複数のプロセッサを逐次的に実行したときに要する時間をいう．また，5つのアプリケーション・プログラムの実行結果を**表7**に示す．

8.2 インプリメントの考察

a．実現機能の評価

粒度（処理単位）：実験結果より，メッセージ・パッシングの実行コスト（実行時間）は他の演算等が平均4クロック程度であるのに比べて約6倍大きい．したがって，オブジェクトが細粒度であれば，並列効果を最も引き出せるが，この場合メッセージ・パッシングによるオーバヘッドが問題になる．しかしメッセージ・パッシングの実行コストが抑制できれば，細粒度のオ

表5 ノード処理時間
Table 5 Processing time of node.

ノード名	単位時間	ノード名	単位時間
加　算	2	コ ピ ー	2
減　算	2	マ ー ジ	1
乗　算	16	ゲ ー ト	1
除　算	16	出　力	2
剰　余	16	構　成	3
符号反転	2	構 成 1	1
論 理 和	2	分　解	3
論 理 積	2	入口 (ENTR)	8
論理否定	2	出口 (EXIT)	6
関係演算子	2	コール (CALL)	12
分　岐	6	リターン (RETN)	6
Tスイッチ	2	プ リ ン ト	2
Fスイッチ	2		
吸　収	1		
脱色 (DCOL)	1		
色 獲 得	6		
色 解 放	2		
色 設 定	1	（アーク	1）

表6 機能別実行時間と展開コスト
Table 6 Run-time and expand cost of functions.

機 能 項 目	展開ノード数	実行ノード数	並列クロック	逐次クロック
メッセージ・パッシング (self パッシング　引数1個)	8	10	35	39
メッセージ・パッシング (1インヘリタンス追加ごと　引数1個)	0	7	31	31
記憶領域アクセス (書込み＆読出し)	4	11	12	37
インスタンス生成 (インヘリタンスなし　インスタンス変数なし)	8	10	40	40
インスタンス生成 (インヘリタンスなし　インスタンス変数あり)	15	14	56	57
インスタンス生成 (1インスタンス変数追加ごと)	9	5	1	18
インスタンス生成 (1インヘリタンス追加ごと)	18	8	32	32

表 7 アプリケーション・プログラムの実行結果
Table 7 Execution result of application program.

プログラム	展開ノード数	実行ノード数	並列クロック	逐次クロック	並列度*	文の数
組合せ ($_5C_2$)	80	219	384	834	2.17	14
ハノイの塔 (3円盤)	108	589	1712	2604	1.52	17
行列式 (3×3)	269	508	249	1968	7.90	40
クイック・ソート (4要素)	605	2369	2319	7265	3.13	93
4クイーン	230	8645	3101	30552	9.85	29

* 並列度＝逐次クロック／並列クロック

ブジェクトは効率が良い．

履歴依存性：記憶領域へのアクセス機能の実行コストはデータの書込み読出し合せて 12 クロックとそれほど大きくなく，展開コスト（グラフの量）は4ノードとなった．

片務性／双務性：特に問題なく実現できた．

創生性：インスタンスの生成はスーパークラス（階層）やインスタンス変数（記憶領域）を持たない場合であっても実行および展開コストはそれぞれ 40 クロックおよび8ノードを要し，インスタンス変数を持つ場合は 56 クロックおよび 15 ノードにもなる．しかしインスタンス変数の個数が増加しても，1変数追加ごとに1クロックの増加に留まっており，これはおのおの変数が並列に初期化されるためと考えられる．

再帰性：再帰性の実現はプログラミング上有用であった．また再帰性により繰り返し処理をサポートすることが可能となった．

自己適用性：構造化トークンを用いることにより，アクアインタンシスを非常に簡潔に実現できた．

階層性：スーパークラスを持つインスタンスの生成については，サブクラスとスーパークラスの初期化は並列に行われてはいるが，階層の増加に比例して実行コストが 32 クロック，展開コストが 18 ノードずつ増加している．またメッセージパッシングについては，展開コストは階層数に依存しないが，実行コストは上位の階層のメソッドが選択されるのに比例して 31 クロックずつ増加する．これについては階層の性質上やむをえないと考える．

b．翻訳機能の評価

展開効率：プログラムの展開コスト（グラフの量）は，クラスのデータフロー・グラフの多重化によりインスタンス生成回数とは無関係に一定であるため，実行に必要な総資源量に対する展開効率は良いものとなっている．一方，今回のアプリケーション・プログラムでは，制御処理（メッセージ・パッシング等に係る処理）の部分がおよそ 1/3 を占めており，従来の逐次型の処理系に比べ制御処理の部分が多い．

最適化：今回最適化を行っていないため，最適化が今後の課題である．

翻訳規則：今回実現した言語は代入文に基づくものである．文と文の間に変数による依存関係があるため，文の実行順序を求めることが翻訳の負担になった．しかし関数型言語ではそのような依存関係がないため，関数型オブジェクト指向言語の方が今回の実現にはより適しているといえる．

c．アプリケーションの考察

実験結果から，計算を部分項に分けた展開式で計算できる問題（行列式の計算）や，1つの問題を複数の部分問題として処理する探索問題（4クイーンの計算）について，データフロー・マシンで7～9倍の並列度を示し，並列オブジェクト指向言語は非常に有効であることが確認された．

d．言語機能の考察

言語の構文は，データの依存関係が文法的に明確な構文，すなわち計算の因果関係が明示的であることが望ましい．さらに記憶資源を管理し制御する機能が言語に必要である．一方，先行評価による副作用を回避するため，構文によって遅延評価を行う機構が必要になる．また記憶領域をアクセスする構文間に明示的な順序関係を付けるために，同期指示機構も備わっていなければならない．

e．ハードウェアへの要求

具体的な実現に向けては，メッセージ・パッシング機構を高速に実現するための機能がパッケージされたノード等を備えたハードウェアが要望される．また，ハードウェアで記憶領域を制御する機能を持つノードが実現できた場合，処理は格段に速くなると考えられる．この機構は純粋なデータフローではないが，現実的なアーキテクチャである．

9. むすび

本研究ではデータ駆動型データフロー・アーキテクチャを用い，オブジェクト指向を実現する方法の一例として以上に述べたような言語と翻訳方法を提示した．

その結果，アクタ・モデルを中間媒体として翻訳規則に使用し，オブジェクト指向言語からデータフローへの翻訳が可能であることが確認された．さらにオブジェクト指向の実現においてデータフロー・アーキテクチャが有効であることが示された．特に探索問題や部分計算に分けて計算する問題については，非常に良い実験結果が得られた．

効率の改善のために，データフロー・アーキテクチャの再考がなされるべきである．またソフトウェアの側には，今後オブジェクト指向にふさわしい並列アルゴリズムの開発が望まれる．

謝辞 本研究を進める上で東京理科大学の馬渕達也君との討論は有意義であった．よってここに感謝の意を表します．

参考文献

1) Veen, A. H.: Dataflow Machine Architecture, *ACM Computing Surveys*, Vol. 18, No. 4, pp. 365-396 (1986); 末吉，富田邦訳：データフロー計算機，bit 別冊 acm computing surveys '86，コンピュータサイエンス，pp. 173-203 (1988. 5).
2) 曽和将容：非ノイマンコンピュータはノイマンコンピュータを越えるか？ 1, 2, 3, 4, 5, bit, Vol. 20, No. 8, pp. 852-856, No. 9, pp. 996-1003, No. 10, pp. 1128-1137, No. 11, pp. 1238-1248, No. 12, pp. 1381-1388 (1988).
3) 雨宮真人：データフロー・アーキテクチャについて，コンピュータソフトウェア，Vol. 1, No. 1, pp. 42-63 (1984).
4) 雨宮真人：関数型言語とリスト処理向きデータフロー・マシン，情報処理，Vol. 26, No. 7, pp. 765-779 (1985).
5) 米澤明憲：オブジェクト指向型プログラミングについて，コンピュータソフトウェア，Vol. 1, No. 1, pp. 29-41 (1984).
6) 横手，所：並行オブジェクト指向言語 Concurrent Smalltalk，コンピュータソフトウェア，Vol. 2, No. 4, pp. 2-18 (1985).
7) 竹内彰一：オブジェクト指向の指向するもの，情報処理，Vol. 29, No. 4, pp. 295-302 (1988).
8) 上村 務：手続き型言語へのオブジェクト指向の導入，情報処理，Vol. 29, No. 4, pp. 310-317 (1988).
9) 米澤明憲：ACTOR 理論について，情報処理，Vol. 20, No. 7, pp. 580-589 (1979).
10) 石塚圭樹：オブジェクト指向プログラミング，p. 362，アスキー (1988).
11) Aho, A. V. and Ullman, J. D.: *Principles of Compiler Design*, p. 604, Addison-Wesley (1979)；土居範久訳：コンパイラ，培風館 (1986).
12) 米澤，柴山，Briot，本田，高田：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，Vol. 3, No. 3, pp. 9-23 (1986).
13) 大槻，馬渕，榎本：オブジェクト指向言語のデータフローグラフへの変換について，電子情報通信学会，1992 春季大会，D-135 (1992).

(平成 4 年 11 月 27 日受付)
(平成 6 年 1 月 13 日採録)

大槻 泰則

昭和 34 年生．昭和 57 年東京理科大学情報科学科卒業．同年沖電気工業(株)入社．平成 3 年東京理科大学大学院修士課程修了．同年三菱電機(株)入社．オブジェクト指向，並列処理に興味をもつ．

榎本 進（正会員）

昭和 24 年生．昭和 47 年東京工業大学電子工学科卒業．昭和 53 年同大大学院博士課程修了．工学博士．現在，東京理科大学理工学部情報科学科講師．プログラミング言語，言語の生成系，CAD の形状記述に興味をもつ．電子情報通信学会，日本ソフトウェア科学会各会員．