《論文特集》

論 19-8

設計支援システムにおける並列プログラミングのためのマクロデータフローモデル†

上野晃嗣*・合田憲人**・原　辰次**

ABSTRACT Computer aided design tools (CAD tools) are necessary for current design processes. A user of these tools needs to program execution order of multiple CAD tools and to execute the programmed tools effectively. This paper proposes a macordataflow model that enables both efficient programming among multiple CAD tools and efficient execution of the tools. The proposed model provides a user visual-programming environment to program execution order of multiple CAD tools, or tasks, and executes the tasks effectively by calculating an earliest executable condition of each task automatically. In addition, the model enables selective task re-execution, in which selected tasks are re-executed in efficient manner, and helps a user to design systems through try and error. The authors developed the prototype of programming environment based on the proposed model. A user of the environment is able to draw a visual program that shows dependence among CAD tools, which are implemented as external software, and the environment invokes execution of the tools in effective order. The prototype showed that the proposed model enables proper and efficient programming among CAD tools, and that the programming environment is able to be an effective programming tool for system designers.

1. はじめに

近年，コンピュータ性能の飛躍的な向上にともない，分野を問わず，比較的計算量の大きい設計支援CAD (Computer Aided Design) ツールが，シミュレーションソフトウェアや特性解析，設計計算ツールなど多数存在し，ユーザは複数のCADツールによる処理を組み合わせて設計作業を行うようになった．従来は，そのような処理の組合せと実行を実現するために，ユーザはCUI (Character User Interface) 上のプログラミングとバッチ処理的な実行を行っていた[1,2]．しかし，こうした設計作業は試行錯誤的な処理手順変更，パラメータ調整などを含む数多くの繰り返しによって行われる作業であるため，ユーザ自身が繁雑なデータ変換やプログラミングを用いてこれらのツールの実行を管理することが次第に困難となり，設計作業におけるツールの実行を統一的，かつ効率的に管理するソフトウェアの必要性が明らかになってきた．

こうした要求に対し本論文では，ツール群の実行を管理し，任意箇所からの効率的な再実行が可能な独自のマクロデータフローモデルを提案する．また，本モデルを用いた設計作業のための総合的なユーザインターフェースを提供するために構築した統一的プログラミング環境について述べる．

提案するマクロデータフローモデルでは，ユーザがツール間の依存関係を表すプログラムを作成することにより，各ツールが実行可能となる最早の条件が自動的に算出され，ツール群の効率的な実行が可能である．モデル中の各要素は隣接要素の情報のみによって独立に動作するため，要素の組み変え（ツール間の処理手順の変更）やパラメータ調整等のプログラムの変更と，作成したプログラムの実行がシームレスに行える．さらに独自のアルゴリズム拡張により，既に実行が終了した任意箇所のツールの再実行開始と，入力値が更新されたツールのみの選択的な再実行が可能であり，これによって試行錯誤的繰り返し作業の支援を行う．また，本モデルを用いて記述されたプログラムでは，ツール間の並列性が自動的に抽出されるため，適

A Macrodataflow Model for Parallel Programming on CAD Systems. By *Kouji Ueno* (Corporate Research & Development Center, Toshiba Corporation), *Kento Aida* and *Shinji Hara* (Tokyo Institute of Technology).

*㈱東芝研究開発センター
**東京工業大学大学院総合理工学研究科

†2000年3月17日受付

切な処理系に実装することにより，並列処理による実行速度向上を容易に実現可能である．

統一的プログラミング環境は，提案するモデルを内部モデルとした，ビジュアルプログラム作成，編集，および実行のためのユーザインターフェースである．本環境は，特定のCADツールに特化したいわゆる統合環境のように機能を囲い込むのではなく，処理に必要な機能を全て外部の既存CADツールを用いて実現する．本環境によりユーザは，数多くのツールを統一的に扱うことができ，それらによる処理の柔軟な組合せと実行が可能となる．

本論文では，まず2章において，本論文が対象とする設計作業として制御系設計作業を例にあげ，複数のCADツールを用いた設計作業について述べる．次に3章において，提案するマクロデータフローモデル，設計作業において求められる再実行の概念，およびプログラミング環境の概要について述べる．4章では，プログラミング環境におけるビジュアルプログラムの動作について述べ，5章では，マクロデータフローモデルの基本的な動作を各要素の状態遷移によって説明し，これに状態を一つ追加することによって任意箇所からの効率的な再実行が可能となることを述べる．最後に6章において，Java言語により構築した本プログラミング環境のプロトタイプの動作例を紹介する．

2. 対象とする設計作業

本章では，本論文が対象とする設計作業について，最適制御系設計を例に挙げて説明する．

2.1 複数ツールの組合せ

本論文では，作業目的の達成のために多数のCADツールを組み合わせて用いなければならないような作業環境を扱う．このような作業においては，処理手順や与えるパラメータなどの試行錯誤的な調整の過程において，必要な処理を行うツールの再実行が度々行われる．また，複雑な構造を持ったデータを扱うツールの利用においては，データフォーマット変換の作業が必須である．

このような作業を必要とする代表的なものとして，制御系解析・設計が挙げられる．制御系設計作業とは，図1のように，設計対象となる制御系をブロック線図の形で構築し，これに対して種々の解析ツール，設計ツール，シミュレーションツールを適用し，設計結果を何度も解析・シミュレーションで確かめながらパラメータを変更していく作業である．

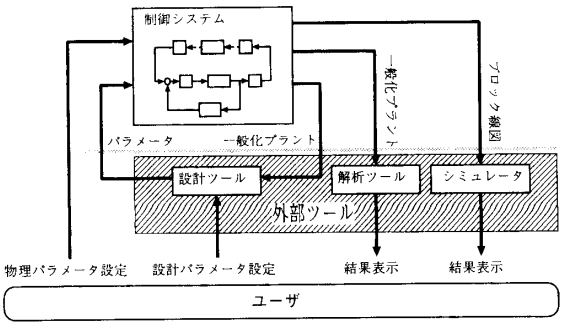

図1 制御系設計作業

2.2 設計作業

ここでは制御系設計作業の例として，最適制御問題を解いてコントローラを設計する作業を挙げる．

ここで考える最適制御問題とは，

$$\begin{cases} \dot{x}=Ax+Bu \\ y=Cx \end{cases} \qquad (1)$$

で表される多入力線形システムを制御対象としたとき，

$$J=\int_0^\infty [x(t)^T Qx(t)+u(t)^T Ru(t)]dt \qquad (2)$$

で表される2次形式評価関数Jを最小化する制御入力uを求める問題である[9]．ここで，QとRは重み行列と呼ばれる設計パラメータで，正定行列である．

このとき，Jを最小にする最適制御入力は

$$u(t)=-R^{-1}B^T Px(t) \qquad (3)$$

で与えられるので，

$$F=-R^{-1}B^T P \qquad (4)$$

が求める状態フィードバック則（コントローラ）となることが知られている．ただし，Pはリカッティ方程式の定常解，すなわち

$$A^T P+PA+Q-PBR^{-1}B^T P=0 \qquad (5)$$

を満たす正定対称行列である．

この最適制御設計を行うためには，ユーザは図2のようなフローチャートで表される作業を繰り返す必要がある．

図2において，ノードはCADツールによる特定の処理を表し，その間の線は主にデータの流れを表す．最上段の3つのノードは，式(2)で表される評価関数Jのパラメータとなる重み行列Q, Rと，制御対象(A, B, C)を表す．

ユーザは，入力パラメータQ, R，および制御対象(Plant)のA, B, Cを最適制御設計ツール（中央のノード）に与えることにより，最適制御問題を解いて，状態フィードバック則（F）を得る．その後，制御対

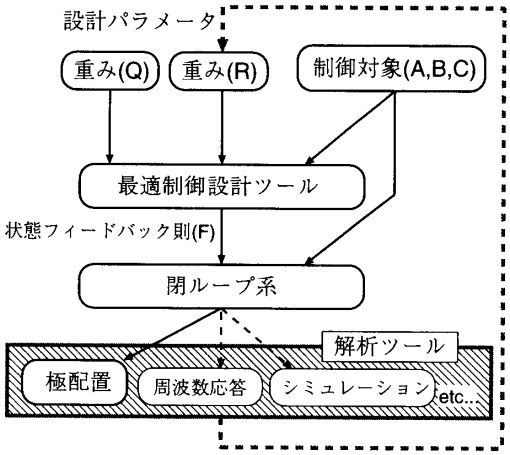

図2　最適制御系設計問題

象（A, B, C）と得られた状態フィードバック則（F）を用いて閉ループ系を構成する．そして，この閉ループ系の特性を調べ，所望の性能が得られているかを確かめるため，解析ツールを用いる．ここで用いられるツールは，例えば極配置のグラフ表現やシミュレーションを始めとする様々な解析ツールが考えられる．

また以上の作業では，ユーザはパラメータを様々に変化させながらツールの繰り返し実行を行う必要がある．その手法には次のようなものが考えられる．

- 解析ツールの結果により得られた特性を観察し，手動で Q, R を変更する．
- ユーザのポリシーを反映したエキスパートシステムを用いて，解析ツールの出力から自動的に Q, R を変更する．
- あるアルゴリズムによって自動生成される系列を Q, R として用いる．

3.　提案手法の概要

本章では，まず提案するマクロデータフローモデルと，設計作業に求められる再実行の概念について説明し，さらに，本モデルを用いて構築する統一的プログラミング環境の概要を述べる．

3.1　マクロデータフローモデル

2章で述べた設計作業では，原則として各ツールは，ツールの全ての入力データが与えられること，および4章で説明する制御依存を満足すること，により実行が開始される．また作業開始後に，パラメータの一部が更新されるというような状態変更が発生した場合，その状態変更によって影響を受けるツールのみが選択的に再実行される必要がある．これを本論文では**選択的再実行**と呼ぶ．この場合，ツールが再実行される条件は，ツールの全入力データのうち状態変更によって更新される全ての入力データが新しく与えられること，および制御依存を満足すること，である．

このような機能を実現するためのデータフローモデルには，大別して二つの手法が考えられる．

静的な手法　各パラメータから各ツールへの依存情報を静的に計算し，プログラム中に全て埋め込む．

動的な手法　パラメータ変更に伴う情報をツール間で伝達し，再実行の必要性を実行時に判断する．

静的な手法では，一度構築したプログラムにおいては，生成された依存情報を利用して，実行の管理における高速動作が期待できるものの，プログラムの規模に対する依存情報の生成に要する計算量のオーダーが大きいため，プログラムを試行錯誤的に変更することが困難となる．また，プログラムの構造と実際に実行されるツールとの対応が複雑になり，ユーザインターフェースの内部モデルとしてはふさわしくない．

一方，動的な手法においては，プログラム全体にわたる依存情報の計算を必要としないため，プログラムの変更は容易である．また，個々のツールの再実行の必要性が実行時に判断されるために，ツールの実行時に若干のオーバーヘッドが生じるが，その量は一つのツールに対しては常に一定で，モデルの規模による変動が無く，それに比較すると設計作業で使用するCADツールは計算量が非常に大きいので，相対的に無視できる．さらに，プログラムの構造は実行されるツールと完全に対応するために対話的なユーザインターフェースにふさわしく，試行錯誤的作業の支援となる．

上記の考察により，本論文が提案するマクロデータフローモデルは，動的な手法を用いて各ツールの実行と選択的再実行を実現している．本モデルはツールに対応するタスクと，タスク間の依存関係によって構成され，各タスクは，他のタスクの状態変化が依存関係を通じて伝搬されることにより，実行または選択的再実行に必要な情報を動的に得ることが可能である．

3.2　統一的プログラミング環境

従来，設計作業におけるツール群による処理の組合せを実現するために，CUIによるプログラミングが利用されていた[1,2]が，作業が複雑で困難であり，ツール間のデータ形式の変換をユーザが行う必要がある等の問題があった（**図3**）．

一方，現在広く受け入れられているGUIをこうしたソフトウェアに導入することも試みられているが，本論文が対象とするような設計作業では特定のCAD

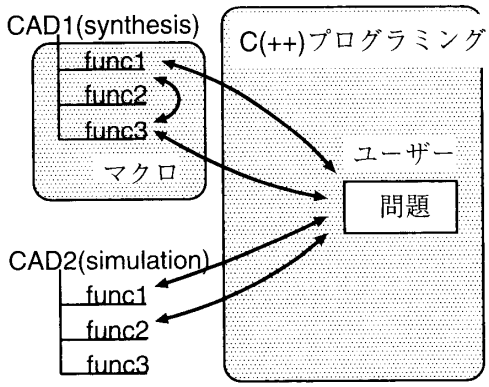

図3　旧来の CUI

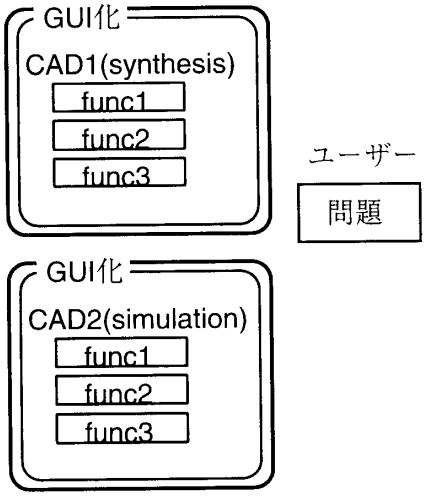

図4　単純な GUI 化

ツールに特化した GUI 化が行われているにとどまり[3]，任意のツールを組み合わせられない等，柔軟性，拡張性が乏しいという問題がある（図4）．

これらの問題に対し本論文が構築するプログラミング環境は，柔軟性，拡張性を有するユーザインターフェースを実現するため，**GUI 部を CAD の機能群から独立**させており，以下のような特徴を持つ．

- ビジュアルプログラミングの導入によるユーザ作業の容易化[4~8]
- さまざまな外部ツールとの柔軟な組み合わせ
- 異なるツール間でのデータ形式の自動変換
- 再実行機能による試行錯誤的作業の支援

本論文で構築するプログラミング環境は，図5に示すように，ビジュアルプログラムを構築，実行する**プログラム制御部**，および，外部ツールの実行とデータの入出力を行う**外部インターフェース部**の二つの部分に分かれている．

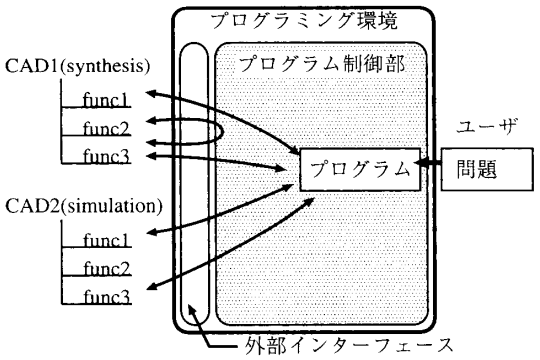

図5　プログラミング環境

プログラム制御部では，ユーザが設計作業を表現するビジュアルプログラムを作成し，実行する機能を提供する．ビジュアルプログラムは提案するマクロデータフローモデルを内部モデルとしており，効率的な並列実行，最適化された再実行が可能である．

外部インターフェース部においては，モデルの動作に伴って実行されるタスクの実際の処理を実現するために，外部の CAD ツールを起動し，必要なデータの入出力を行う．この際に必要となる CAD ツール等のソフトウェア起動，データ構造の変換は外部インターフェース部が自動的に行うため，ユーザはこれらを意識する必要はない．

これらにより，本環境は従来よりも柔軟で効率的な CAD ツール群の組合せと実行を実現する．

3.3　関連研究

マクロデータフローモデルは主に並列処理分野において，並列性検出のためのタスク間依存関係を示すモデルとして使用されてきた[11~14]．本論文が提案するマクロデータフローモデルの基本的な考え方は，これらの研究とほぼ同様の「入力パラメータが揃ったツールから順次実行」というものであるが，本モデルにおける一度実行終了したプログラムの一部の選択的再実行という機能は今まで提案されていない．また，提案するモデルは一つのツールの実行開始条件の生成，評価を実行時に動的に行うために，そのためのオーバーヘッドが生じるが，各ツールの計算量はこれらのオーバーヘッドが無視できる程度に大きいため，並列処理環境においては大幅な処理速度の向上が見込める．

一方，ビジュアルプログラミングの内部モデルとしては，既存言語の文法構造を図式化したもの[6,7]や，オブジェクト指向に基づくもの，データフローを用いるもの[4]などがあるが，やはり本モデルのように選択的再実行を考慮したものは提案されていない．

科学技術計算におけるビジュアルプログラミングの実用化例としては，Advanced Visual Systems 社による科学技術計算用可視化ツール AVS[10]におけるビジュアルプログラミング機能などが有名であるが，これは用途がデータ可視化を目的としたデータ加工に限られており，また条件分岐などの基本的なプログラム記述能力も乏しい．本論文のプログラミング環境が提供するプログラミング機能は，ネイティブプログラム言語と同等の記述能力を有し，汎用の構造となっている．

4. ビジュアルプログラミング

本章ではプログラム制御部が提供するビジュアルプログラミング機能，および，これにより作成されたプログラムの実行方式について説明する．

4.1 ビジュアルプログラミング機能

本プログラミング環境では，ユーザはビジュアルプログラミング機能を用いて外部ツールの実行手順を決定（プログラミング）する．**図6**に，このビジュアルプログラム例を示す．図中，左がC言語で書かれたプログラム，右が同等の機能を実現するビジュアルプログラムである．このようにビジュアルプログラムでは，ツール（図左のf1( )～f6( )の関数）をノード，ツール間の実行順序関係をエッジにより表現する．

4.2 ビジュアルプログラム構成要素

提案するビジュアルプログラムは，ノードで表されるツールと，有向エッジにより表される依存関係の相互接続によって構成される．また，エッジの矢印の先を下流，元を上流と呼ぶ．なお，以後はツールをタスクと呼ぶ．

タスクには以下の種類が存在する．

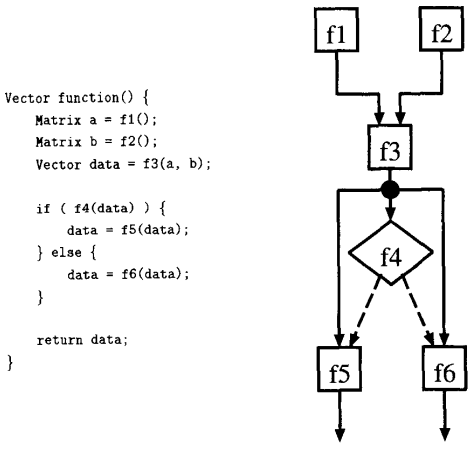

図6 ビジュアルプログラム例

通常タスク 矩形で表される．本タスクは外部ツールを意味し，実際には本タスク内で外部ツールの呼び出しが行われる．

条件分岐タスク 菱形で表される．条件判断によって下流タスクの実行を制約する．

分岐タスク 黒丸で表される．単一の入力データを複数の出力に転送する．

合流タスク 白丸で表される．複数の入力と単一の出力を持ち，入力データを逐次，出力に転送する．

一方，依存関係には以下の種類が存在する．いずれの依存関係も，上流側タスクの実行終了後に下流側タスクの実行が行われることを保証する（ただし，後述するように再実行時は動作が異なる）．

データ依存 実線のエッジで表される．タスク間のデータフローを表し，上流側のタスクの実行終了と共に出力データを下流へ転送する．また，データの型チェック機能を持ち，上流側の出力データと下流側の入力データが同一の型である事を保証する．

制御依存 破線のエッジで表される．条件判断により下流のタスクのうち1つのタスクの実行が許されることを表現する．主に条件判断による実行経路の変更を実現するために導入され，条件分岐タスクが下流タスクの実行をコントロールするために用いる．

各タスクは，上記のデータ依存および制御依存が満足された時点で実行を開始する．これより本方式は実行時における並列性を実現している．

4.3 選択的再実行

本論文が提案するマクロデータフローモデルでは，従来のマクロデータフロー型並列処理系の仕組み[11~14]と異なり，任意のタスクからのプログラムの選択的再実行を許し，かつタスクの再実行回数の最適化を実現している．

このアルゴリズムはユーザが作成したビジュアルプログラム全体に対して行う最適化ではなく，タスク，依存関係などの要素の独立動作の組合せによって実現する機能なので，実行中にプログラムの組み変え，パラメータ変更，再実行が自由に行えるという利点がある．

ここでは**プログラムの再実行を，任意のタスクに再実行指令を発する**ことと定義する．例えば**図7**において，左の図のように，Task A の再実行を行った場合，Task C が一度だけ再実行される．中央の図のように，Task A, B それぞれが一度ずつ再実行された場合，

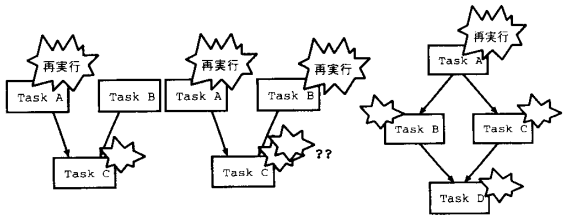

図7 選択的再実行動作

Task C は基本的に二回再実行される．しかし，右の図のように，Task A が再実行された場合，Task D は，Task B, C 双方の終了を待ってから初めて一度だけ再実行されなければならない．

このように，プログラムを効率良く実行するためには，ある一箇所から発したプログラム再実行によるタスクの再実行回数を最適化する必要がある．これを実現するために，提案モデルは以下の二つの独自の特徴を持つ．

- 一度実行されたタスクは前回の入力データ値を保持し，再実行時には更新された値以外は前回値を用いる．
- タスクの実行可能性を依存関係を通じて伝搬させ，それを用いてタスクの再実行タイミングを決定する．

タスクの実行可能性とは，あるタスクが実行される可能性を意味する．あるタスク A に実行可能性があるための条件は，A が必要としているデータを出力する上流のタスク全てが実行中か，実行可能性がある状態であることと定義する．

4.4 マクロデータフローモデルの動作設計方針

提案するビジュアルプログラミング機能は，試行錯誤を含むユーザの作業の効率化を目指している．ユーザは常に作業を行っているので，（ビジュアル）プログラムは常に実行中であり，その間にユーザが作業の進行に応じてプログラムの変更を行っても，プログラムの実行が途切れないことが望ましい．そのような動作を可能とするために，提案モデルでは**各要素は独立動作する**ものとする．

従って，各要素は隣接要素の持つ情報のみによって状態遷移を起こし，その連鎖によってプログラム全体が動作する．このため，ユーザが無限ループやデッドロックを招くプログラムを作成してしまう可能性があるが，本環境はユーザインターフェースとしてビジュアルプログラミング機能を提供しているので，そのような場合にはユーザによる適切な修正が入ることを前提としている．

5. マクロデータフローモデル

本章では前章で述べたプログラムの実行制御を実現するためのマクロデータフローモデルについて述べる．

5.1 基本的な状態遷移

本マクロデータフローモデルでは，プログラムを構成するタスク，データ依存，制御依存の三要素それぞれが，True, Unstable, False の 3 つの状態間で遷移し，この状態遷移により，タスクの実行が制御される．

各要素の基本的な状態遷移モデルを**図8**に示す．

この図において，要素のとる 3 状態 F, U, T はそれぞれ False, Unstable, True の略である．c1, c2, c3 はそれぞれの状態間を遷移するための条件である．Unstable から False への遷移の条件は条件 c1 の不成立であり，すなわち状態が Unstable であるためには c1 が成立し続けなければならないことを表す．

5.2 タスクの状態遷移

タスクは後述する状態遷移条件によって，図 8 と同様の状態遷移を行う．それぞれの状態の意味を次に示す．

False 待機状態．タスクの実行可能性が無い状態．

Unstable 準備状態．タスクの実行可能性が存在する状態．

True 実行状態．タスクが起動され，処理に必要な外部コマンドを呼び出す状態．

なお，初回実行後，タスクは全てのデータ依存の入力値を保持し，再実行時において更新可能性の無い入力値についてはこの値（前回値）を用いる．

タスクの状態遷移条件については5.5節で述べる．

5.3 データ依存関係の状態遷移

データ依存関係はデータフローを表し，その状態は基本的に上流タスクの状態を反映する．しかし下流タスクは，上流タスクの実行終了とともにデータ転送が終了した時点で実行可能となるため，Unstable から True へ遷移するための条件 c2 はデータ転送の終了が

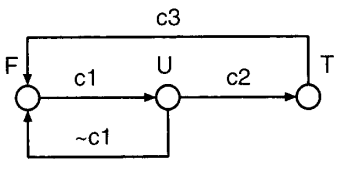

図8 状態遷移モデル

必須となっている．上流タスクの実行が終了するとデータ依存関係は上流タスクからデータを取得し，一時的にデータを保持すると同時に，下流タスクへデータの更新を通知する．その後，下流タスク側の条件が整った時点で，下流タスクへデータを転送する．

各状態の意味を以下に示す．

False 待機状態．データ更新可能性の無い状態．

Unstable 準備状態．データ更新の可能性が存在するが，実際にはデータが転送されてはいない状態．

True データ所有状態．上流タスクが全出力データを出力し，データ依存関係，即ちデータ依存エッジがデータを保持している状態．

状態遷移条件を以下に示す．

c1 上流タスクがUnstableである．

c2 上流タスクの実行が終了し，かつ，全出力データをデータ依存エッジに出力する．

c3 下流タスクがデータ依存エッジから全てのデータを受け取る．

ただし，下流タスクが一過性でないFalse状態（状態遷移条件c1が成立しない）である場合，データ依存エッジに転送されたデータは一旦下流タスクに転送され，前回値に設定された後に捨てられる．この動作は5.5節で述べるが，タスクの状態遷移条件に出力側データ依存関係の状態が含まれるため，下流タスクが何らかの原因で実行されない場合，データ依存エッジがデータを保持し続けることによって上流タスクが連鎖的に実行不可能となってしまう現象を防ぐ意味がある．なお，一過性のFalse状態とは，True状態から条件c3, c1が同時に成立した際に，Unstable状態に遷移する前に通るFalse状態を指す．

5.4 制御依存の状態遷移

制御依存も3状態を持つが，上記の2要素とは異なり，上流タスクの処理内容によって変則的な動作を行う．また，状態は順番に遷移せず，排他的条件によって一意に定まる．以下に各状態を示す．

False 実行禁止．上流タスクの分岐方向が本制御依存エッジの下流タスク以外である，即ち，この下流タスクが実行されない状態．

Unstable 未確定．上流タスクの状態がUnstableである状態．

True 実行許可．上流タスクの分岐方向が本制御依存エッジの下流タスクである，即ちこの下流タスクの実行が許される状態．

また，5.5節で述べるタスクの状態遷移条件の作成に，制御依存の論理積が必要になるので，**表1**に定義する．

表1 制御依存の状態値の論理積

値1	値2	値1 AND 値2
F	F	F
F	U	F
F	T	F
U	U	U
U	T	U
T	T	T

5.5 タスクの実行条件

タスクの実行開始条件は，タスクに入力するデータ依存，制御依存の状態により表すことができる．ただし，通常タスクと合流タスクとでは条件が異なる．

5.5.1 通常タスク

前節までの定義を用いて，通常のタスクの状態遷移条件は以下のように表せる．なお，タスクの初回実行/再実行の判断は，タスクが全てのデータ依存入力について前回値を保持しているか否かで行われる．

変数定義

$TASKi$：タスクi

$EC(TASKi)$：タスクiへの全ての制御依存入力の論理積

$DI(TASKi)$：タスクiへ入力するデータ依存の状態（下流側状態）の集合

$DO(TASKi)$：タスクiから出力するデータ依存の状態の集合

実行準備条件

タスク初回実行時の実行準備条件（c1）:

$$EC(TASKi) = !F$$
$$\forall d \in DI(TASKi),\ d = !F$$

タスク再実行時の実行準備条件（c1）:

$$EC(TASKi) = !F$$
$$\exists d \in DI(TASKi),\ d = !F$$
$$\forall d \in DO(TASKi),\ d = F$$

（注）実行準備状態中に $\{d | d \in DI(TASKi),\ d = !F\}$ は追加される．

実行条件

タスク初回実行時の実行条件（c2）:

$$EC(TASKi) = T$$
$$\forall d \in DI(TASKi),\ d = T$$

タスク再実行時の実行条件（c2）:

$$EC(TASKi) = T$$

$\forall d \in DI(TASKi),\ d = !U$

$\exists d \in DI(TASKi),\ d = T$

なお，タスクの実行終了条件（c3）は実際に起動した外部ツールの実行終了である．

5.5.2 合流タスク

タスクのうち，合流タスクのみは状態遷移条件が異なる．合流タスクとは，複数のデータ依存入力と単一のデータ依存出力を持つが，全てのデータ依存入力を同様に扱い，入力データが揃うのを待つことなく，入力されたものから順番に出力に転送するタスクである．また，合流タスクは前回値を必要とせず，データが更新され次第出力に転送するため，初回実行と再実行の動作に差異が無い．

実行準備条件（c1）:

$$EC(TASKi) = !F$$

$\exists d \in DI(TASKi),\ d = !F$

$\forall d \in DO(TASKi),\ d = F$

実行条件（c2）:

$$EC(TASKi) = T$$

$\exists d \in DI(TASKi),\ d = T$

なお，実行終了条件（c3）は出力データ依存へのデータ転送終了である．

5.6 実行時動作例

プログラムの実行時の動作例を**図9**に示す．図9は一度目の全体の実行が終了した後，Task Aに対して再実行指令を与えた場合のタスクおよびエッジの主な状態を表したものである．タスクの再実行時，全てのデータ依存入力はUnstable**以外**でなければならないと定めてあるので，実行可能性がデータフローよりも先に伝播する事により，Task Dの再実行開始タイミングを遅らせ，無駄な再実行を防ぐことが可能となっている．

それぞれのステップは以下のような状態を表している．また，最後の二つの図a, bは，ステップ3における分岐Task Bの処理結果，即ち分岐方向が異なった二つのケースを表す．

1. タスクAが再実行中の状態．この時B, B', B'', C, D各タスクの状態はUnstableである状態．
2. タスクAの実行が終了．タスクAの出力データがタスクB, Cに送信され，タスクB, Cが再実行を開始した状態．
3. タスクCの実行が終了．データは下流のデータ依存エッジに送信されるが，タスクDはタスクB側の入力データ依存エッジの状態がUnstableであるため，このデータを取得せず，再実行を開始しない状態．

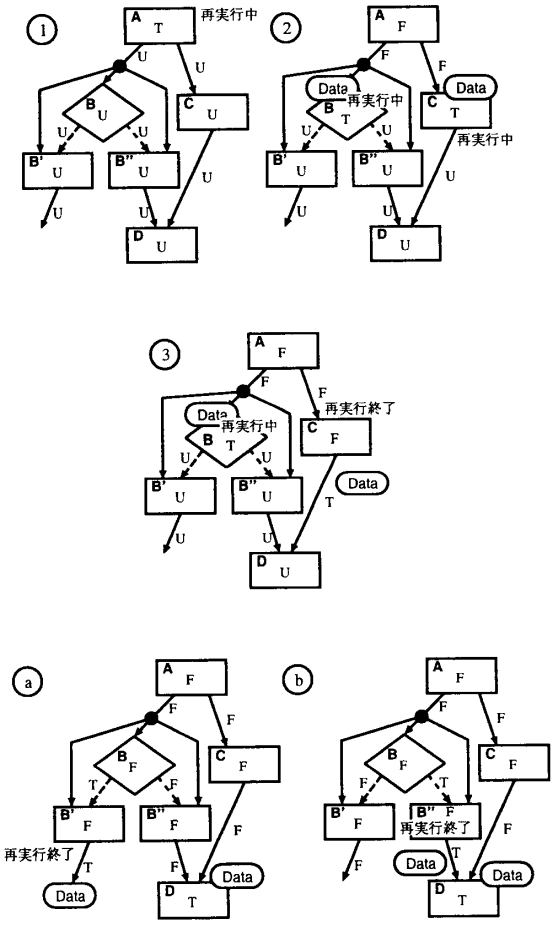

図9 再実行例

a. タスクBの実行が終了し，条件分岐がタスクB'方向に分岐したため，タスクB'が再実行された状態．タスクDは，タスクB''からデータが送信されないことが確定したため，タスクB''からのデータについては前回値を用い，再実行を開始する．

b. タスクBの実行が終了し，条件分岐がタスクB''方向に分岐したため，タスクB''が再実行された状態．タスクDは，タスクB''からデータが送信されたため，更新された値を取得し，再実行を開始する．

6. プロトタイプの実装

本論文が提案したマクロデータフローモデルの正当性，およびプログラミング環境の有効性を検証するため，以上で述べた方針とアルゴリズムを用いて，Java言語[16]により，プログラミング環境のプロトタイプを構築した．

6.1 例題のプログラム例

本プロトタイプを用いて2.2節で述べた最適制御系設計の例題（図2）をプログラミングした例を**図10**に示す．また，**図11**は，その実行中の画面の例を表したものである．図10中，ノードの横に書かれた文字列は，説明のために書き込んだものである．また，分岐タスク以外の全てのタスクは，外部プログラム呼出によって実現している．

プログラム中の記号については，角丸矩形がタスク，実線はデータ依存関係を表す．また図11では，タスクの状態を白抜き（False），グレイ（Unstable），白黒反転（True）により表している．依存関係については，黒（False または True），グレイ（Unstable）により表されている．さらに黒丸は分岐タスク，白丸は合流タスクを意味する．

OptimalControl タスクは Q, R, Plant の3入力を取る．Q, R, Plant と脇に書かれているノードが，図2中のノードと対応している．この例では Q, Plant を OptimalControl に渡す前に，それぞれ正定行列であるか，可制御であるかをチェックしている．また，R については，10倍しながら5回シーケンシャルに渡す，即ち，選択的再実行を行うようにプログラミングしている．

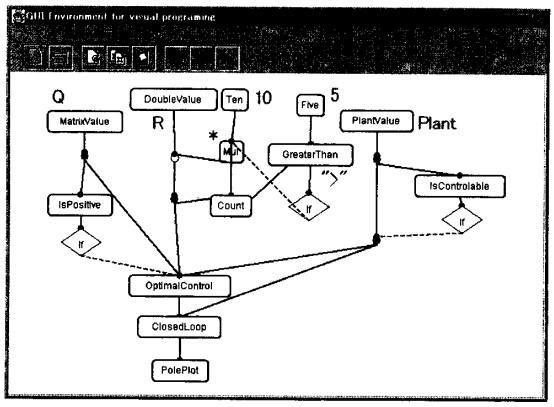

図10 例題のプログラム例

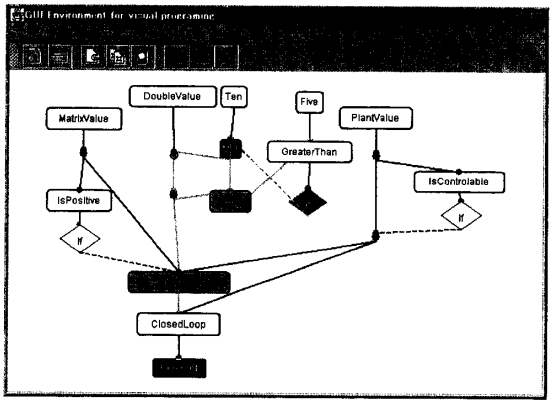

図11 実行画面

図11に示す実行中画面の例では，OptimalControl が一度実行された後も R に関するループが実行され，関連するタスクの再実行が行われる．また，タスク実行を含む全要素の状態遷移は並列に行われているため，プログラム中の複数要素が実行状態にある．

また，解析結果を解釈し，ユーザの考え方に従って入力パラメータを決定するような何らかのエキスパートシステムが存在したとすれば，そうしたツールもまた上記のビジュアルプログラムに組み込み，作業全体のループをプログラムすることができる．

6.2 考 察

本プロトタイプによる検証により，提案するマクロデータフローモデルがプログラミング環境上のビジュアルプログラムの内部モデルとして正しく動作することが確認され，提案モデルの正当性が確認された．また，一度実行されたプログラムの選択的再実行やループの動作に対し，前回値の参照と Unstable 状態の導入により，提案モデルでは効率的なタスクの選択的再実行が可能であることが確認された．さらに，プログラミング環境中の外部インターフェースを実装することにより，マクロデータフローモデルの動作に伴うユーザの介在を必要としない外部プログラム呼び出しが実現できることが確認された．

これらにより，提案するマクロデータフローモデルによる外部ツールの柔軟な組合せと実行の統一的管理が可能であり，プログラミング環境がそのユーザインターフェースとして有効であると言える．

7. おわりに

本論文では外部ツール群の効率的な実行を管理し，任意箇所からの効率的な選択的再実行が可能な，独自のマクロデータフローモデルの提案を行った．また，提案したモデルに基づいて構築した設計作業のためのプログラミング環境について述べた．

提案したマクロデータフローモデルとプログラミング環境の正当性，有効性の検証を行った結果，提案するマクロデータフローモデルにより，効率良い外部ツール群のビジュアルプログラミング，および選択的再実行を含む実行制御が可能であることが確認された．

参 考 文 献

1) The MathWorks Inc: MATLAB User's Guide (1997)
2) 古賀，古田：数値処理と数式処理を融合した制御系CAD言語MaTX，計測自動制御学会論文集，**29**-10, 1191/1198 (1993)
3) The MathWorks Inc: Using SIMULINK (1997)
4) K. J. Schmucker: Rapid Prototyping using Visual Programming Tools, Proceedings of the CHI '96 conference companion on Human factors in computing systems: common ground April 13–18, 359/360 (1996)
5) B. A. Myers: Visual Programming, Programming by Example, and Program Visualization: A Taxonomy, Proceedings of the CHI '86 conference on Human facors in computing systems, 59/66 (1986)
6) N. Cunniff, R. P. Taylor, J. B. Black, Does Programming Language Affect the Type of Conceptual Bugs in Beginners' Programs? A Comparison of FPL and Pascal, Proceedings of the CHI '86 conference on Human facors in computing systems, 175/182 (1986)
7) B. J. Reiser, P. Friedmann, J. Gevins, D. Y. Kimberg, M. Ranney, A. Romero: A GRAPHICAL PROGRAMMING LANGUAGE INTERFACE FOR AN INTELLIGENT LISP TUTOR, Proceedings of the CHI '88 conference on Human facors in computing systems, 39/44 (1988)
8) E. M. Wilcox, J. W. Atwood, M. M. Burnett, J. J. Cadiz, C. R. Cook: Does Continuous Visual Feedback Aid Debugging in Direct-Manipulation Programming Systems?, Proceedings of the CHI '97 conference on Human facors in computing systems March 22–27, 258/265 (1997)
9) 小郷，美多：システム制御理論入門，156，実教出版 (1979)
10) http://www.avs.com/
11) R. G. Babb II: Parallel Processing with Large-Grain Data Flow Techniques, IEEE Computer, **17**-7, 55/61 (1984)
12) 本多，岩田，笠原：Fortranプログラム粗粒度タスク間の並列性検出手法，電子情報通信学会論文誌，**J73-D-I**-12, 951/960 (1990)
13) 本多，合田，岡本，笠原：実行開始条件による並列性検出手法—ループへの拡張，並列処理シンポジウムJSPP'93, 103/110 (1993)
14) M. Girkar, C. D. Polychronopoulos: Automatic Extraction of Functional Parallelism from Ordinary Programs, IEEE TRANSACTIONS ON PARALLEL AND DITRIBUTED SYSTEMS, **3**-2, 166/178 (1992)
15) 原，小林：制御系CADのための結合計算アルゴリズム：ブロック線図のグラフィック入力と伝達関数の記号処理，システムと制御，**29**-2, 105/114 (1985)
16) D. Flanagan，永松 訳：JAVAクイックリファレンス，O'Reilly JAPAN (1996)