JNC TN9400 99-053

分散協調型計算によるプラント シミュレーション手法の検討

(研究報告)

１９９９年３月

核燃料サイクル開発機構
大洗工学センター

本資料の全部または一部を複写・複製・転載する場合は、下記にお問い合わせください。
〒319-1194　茨城県那珂郡東海村村松4番地49
核燃料サイクル開発機構
技術展開部　技術協力課

Inquiries about copyright and reproduction should be addressed to:
Technical Cooperation Section,
Technology Management Division,
Japan Nuclear Cycle Development Institute
4-49 Muramatsu, Tokai-mura, Naka-gun, Ibaraki 319-1194,
Japan

© 核燃料サイクル開発機構（Japan Nuclear Cycle Development Institute）
1999

JNC TN9400 99-053
1 9 9 9 年 3 月

分散協調型計算によるプラントシミュレーション技法の検討

(研究報告)

米川 強[*1], 吉川 信治[*1], 瀬谷 義一[*2]

要 旨

原子力プラントに代表される、複雑で大規模な流体循環システムの設計、評価に不可欠な非定常流動計算シミュレーションを、従来のような行列計算を用いないで行うアルゴリズムを考案した。このアルゴリズムを高速炉の2次主冷却系構成を対象に検証した。その結果、流体計算に収束計算を用いることによる計算量の増加は、熱交換器に代表される機器モジュールの温度計算量に比べて十分小さく、このアルゴリズムの利点である流路ネットワーク構造の動的変化への対応等の特徴を損なわないことを確認した。

本手法により、プラントの全体モデル、特に流路の破断や隔離のように、境界条件の構造そのものが不連続に変化する現象を模擬できるモデルを容易に構築でき、シミュレーションの柔軟性を向上させられる可能性がある。

[*1] 大洗工学センター システム技術開発部 ビーム利用技術開発Gr

[*2] 原子力システム(株)

Study on Method of Plant Simulation under Distributed Cooperative Calculation

Tsuyoshi YONEKAWA[*1], Shinji YOSHIKAWA[*1],
Yoshiichi SEYA[*2]

Abstract

A new algorithm has been proposed for one-dimensional liquid dynamics has been proposed, which is indispensable in design and estimation of complex and large-scale fluid circulation systems like nuclear power plants. This algorithm, with no need for matrix calculation in the contrast with the conventional approach, has been validated on an equipment piping configuration to a FBR secondary heat transport system, to demonstrate that increase of CPU load due to iterative process in hydraulic calculation to reach convergence in each time step is enough small to retain the inherent advantage of this algorithm such as capability to follow dynamic configuration change of the piping network.

This approach enables to easily build whole plant model capable to handle incontinuous configuration changes such as piping ruptures and isolations, to suggest potential in enhancing plant simulation flexibility.

[*1] Beam Technology Development Group, System Engineering Technology Division, Oarai Engineering Center, Japan Nuclear Cycle Development Institute

[*2] Nuclear Energy System Inc.

目　次

図リスト

1. 緒言

原子力発電所のような液体が循環する系(ループ)を多数有する大規模プラントのシミュレーションは、プラント設計、特性評価、並びに運転員教育に不可欠である。しかしながら、そのためのソフトウェア構築には多数の構成機器の数学モデル(機器モジュール)を緊密な整合性を保ちながら製作し、統合していくプログラミング作業が要求される。特に多数の構成機器によって形成される循環流路内の流動計算に大規模な行列を用いる解法[1]を利用していることに起因している。

上記の流動計算で扱うような行列の解法としては、線形代数を応用して一度に解を導く手法[2]が一般的に用いられている。一方、行列を用いることなく、式毎に決められた変数に誤差を吸収させる処理を構成する式全体に対して反復して解を導く手法[2]もある。流路ネットワーク内の流動計算にこれを適用することにより、機器や配管の数学モデルを異なる計算機上に分散配置し、計算機ネットワークを介してプラントモデルとして統合し、全体シミュレーションとして行なうことも可能になる。この場合は、プログラミング言語等の開発環境に対しての制約が事実上解消される。さらに、

(1) 構造物により隔てられた異なる液体循環ループが、その構造物の破損により単一のループに変化する事象

(2) 配管破断によって新たな圧力境界が出現する事象

(3) 弁の誤閉による流路が遮断する事象

等に代表される流路ネットワーク構造の動的な変化に容易に対応でき、これらの現象を模擬することが可能となる。

次章以降、上記で示した行列演算を用いないで流動計算を行う手法(以後、これを"分散協調的計算手法"と呼ぶ)の具体化、この手法を基にしたプラントの基本要素(機器、配管)をオブジェクトモデル化したシミュレーションシステムの試作結果について述べる。

2. 分散協調的流動計算手法

本手法では、プラントの構成機器(熱交換器、配管等)の圧力、流量、流量の時間変化率等で表現される流動式群を行列形式として扱わない解法を開発することに主眼を置いている。このような流動式群は構成機器とその接続状態から導くことができるので、隣接する機器の間での局所的な数値計算のみで解くことができれば、行列形式での処理を回避でき、かつ、動的な流路ネットワーク構造の変化にも"計算中に"対応できる。以下に、本手法で提唱する局所的な数値計算の繰返しによって、行列形式で解法した場合と等価な解を得るアルゴリズムを示す。

2.1 対象システム

Fig.2-1(a)に示すIRACS(Intermediate Reactor Auxiliary Cooling System:補助炉心冷却系)に空気冷却器を用いた高速炉の2次系(ポンプ廻り、止め弁等は簡略している)を対象とする。この例では分岐点と合流点を各々1個持ったものであるが、任意の数であってもよい。このような構成からジャンクション(流体の分岐点または合流点)とノード(構成機器)による、Fig.2-1(b)で示す接続関係へ変換する。

2.2 解法アルゴリズム

前節でノード/ジャンクションにモデル化した流路ネットワークの流量等を算出するアルゴリズムを以下に示す。

[Step-1]

全ジャンクション毎に、そこに接続している各々のノードについて、当該ノード以外のノードの流体慣性値(*1)を全て掛け合わせた結果を、各ジャンクションにおける当該ノードの端重み(接続ノードの流体慣性値の逆数の比に相当)とする。ジャンクションに接続する全てのノードの端重みを合計した結果を当該ジャンクションの重み合計とする。また、ジャンクションにおける質量保存誤差項(*2)を0とする。この処理により質量保存誤差を、各ノードの"流体慣性の逆数"に比例して配分すること、また、ポンプのように流体慣性が0のノードに接するジャンクションでも計算を行うことが可能になる。

例) あるジャンクションに接続するノードが3つ(A,B,C)ある場合、

Aの端重み＝ Bの流体慣性値×Cの流体慣性値

Bの端重み＝ Cの流体慣性値×Aの流体慣性値

Cの端重み＝ Aの流体慣性値×Bの流体慣性値

そのジャンクションの重み合計＝Aの端重み＋Bの端重み＋Cの端重み

(*1) 流体慣性は流路長さ/(重力加速度×流路断面積)で表わされ、流路内の流れの変化のしやすさを示す指標

(*2) 質量保存誤差項はジャンクションにおける流入量(及び変化率)と流出量(及び変化率)との差。理論的には常に0(非圧縮性流体の場合)

[Step-2]

前回計算時の各ノードの圧力差(圧力損失項)を用い、圧力境界のジャンクションを起点として接続状態に応じて順次枝状に各ジャンクションの圧力近似値を算出する。また、圧力差が参照されたノードの圧力差参照フラグを1(True)とする。

例)

圧力差参照フラグがTrueのジャンクションJ1の圧力設定値がP1の場合、これに接続するノードに関して、そのノードの他方のジャンクションJ2の圧力差参照フラグがFalseなら、そのノードの前回の圧力差ΔPからJ2の圧力を、

J1→J2の流れならば、(P1-ΔP)とし、J1←J2の流れならば、(P1＋ΔP)に設定する。

[Step-3]

[Step-2]で圧力差が参照されなかったノードについて、両側のジャンクションの圧力と前ステップの流量から当該ノードの流量変化率を算出する。Fig.2.1の場合、ノード3の圧力差を用いてジャンクション2とジャンクション3の圧力近似値を算出した場合は、圧力差参照フラグが0(False)であるノード4に関して流量変化率を算出することになる。

[Step-4]

質量保存則より各ジャンクションでの流体の流入量と流出量は等しいので、流量変化率の算術合計は0でなければならないが、[Step-3]で(流量変化率を)再計算しているため質量保存則は成立していない。そこで、各ジャンクションで流量変化率の算術合計を集

計する。つまり、全ノードについて現在の流量変化率を上流側のジャンクションの質量保存誤差項からは減じ、下流側のジャンクションの質量保存誤差項には加算する。

[Step-5]

全ジャンクションについて、Step-4で算出した質量保存誤差項を接続しているノードに配分する。この時の配分比はノードへの流れ易さを反映させるため、それぞれのノードの端重みに反比例させる。この時、質量保存誤差項をすべてノードに配分すると、誤差が収束しないことが考えられるので、誤差に1以下の正の係数をかけた値を修正量とする。本報告書では、この係数を0.5に設定した。

流入側のノードについては、その流量変化率から配分した質量保存誤差項を減じ、流出側のノードについては加算する。前記の例の場合、ノードAに対しては(Aの端重み/重み合計)×(質量保存誤差項)/2を配分する。Fig.2.2参照。

[Step-6]

各ジャンクションにおける質量保存誤差項がしきい値以下に減少するまで、[Step-2]からの処理を繰り返す。この処理過程において、質量保存則あるいは運動量保存則上の局所的な誤差が流路ネットワーク全体に伝播していく。これによって全体を行列計算で一括して解法した場合と等価な解が得られる。これを補足すると次のようになる。

(前提)　図2-1に示す各ノードについて、

(上流側ジャンクション圧力) - (下流側ジャンクション圧力) = $f_i*W_i^2+I_i*dW_i/dt$

が成立するものとする。但し、iは該当するノードを指す添字、Iは流体慣性値、fは圧力損失係数、Wは流量とする。

ここで、各ノードの流量時間変化率と各ジャンクションの圧力を求める連立方程式は、

各ノードの運動量保存則から:

$$P_i - P_1 = I_1*dW_1/dt+f_1*|W_1|*W_1$$

$$P_3 - P_0 = I_2*dW_2/dt+f_2*|W_2|*W_2$$

$$P_1 - P_3 = I_3*dW_3/dt+f_3*|W_3|*W_3$$

$$P_1 - P_2 = I_4*dW_4/dt+f_4*|W_4|*W_4$$

$$P_2 - P_3 = I_5*dW_5/dt+f_5*|W_5|*W_5$$

各ジャンクションの質量保存則から:

$$dW_1/dt = dW_3/dt + dW_4/dt$$

$$dW_4/dt = dW_5/dt$$

$$dW_3/dt + dW_5/dt = dW_2/dt$$

これを行列を用いて書き換えると:

$$\begin{pmatrix} I_1 & & & & 1 & & \\ & I_2 & & & & & -1 \\ & & I_3 & & -1 & & 1 \\ & & & I_4 & -1 & 1 & \\ & & & & I_5 & -1 & 1 \\ 1 & & -1 & -1 & & & \\ & & & 1 & -1 & & \\ & -1 & 1 & & 1 & & \end{pmatrix} \times \begin{pmatrix} dW_1/dt \\ dW_2/dt \\ dW_3/dt \\ dW_4/dt \\ dW_5/dt \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} P_i - f_1*|W_1|*W_1 \\ -P_0 - f_2*|W_2|*W_2 \\ -f_3*|W_3|*W_3 \\ -f_4*|W_4|*W_4 \\ -f_5*|W_5|*W_5 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{(Eq.1~8)}$$

本報告で提案する手法は、

·Eq.1~Eq.5までを満たすdW_1/dt~dW_5/dt、P_1~P_3の暫定値を定める。([Step-1]から[Step-3]に相当)

·Eq.6~Eq.8までの右辺が質量保存誤差となるので、この誤差が半分になるようなdW_1/dt~dW_5/dtをEq.1~Eq.5の対角線上の係数の逆数に比例した修正量で再計算し、これに合わせてP_1~P_3も再計算する。([Step-4]~[Step-5]に相当)

上記の考え方を各ジャンクションにおける質量保存誤差項に応用することで行列計算を行うことなく流動式全体の解法を行っている。

[Step-7]

しきい値以下に達したら、各ノードの流量を更新(変化分＝流量変化率×時間刻み幅を加算)し、[Step-1]に戻ってシミュレーションの時刻を時間刻み巾分進め、次の時刻の流量変化を算出する。

2.3 流動計算手法の評価

前節で示した流動計算の解法の妥当性を評価するため、Fig2.3に示すようなポンプと流体抵抗を有する一巡の流路を対象にして、理論解との比較を行った。

図に示すように、この場合の流量F(T)の理論解は時刻Tと一巡の流体抵抗(圧力損失)係数αによって一義的に以下のように定まる。

$F(T) = 1/(1+\alpha*T)$

一方、この流動式を1階の差分式で表すと、

差分化(その1)

$\{F(N+1) - F(N)\}/\delta T = -\alpha*F(N)*F(N)$より、

$F(N+1) = F(N) - \delta T*\alpha*F(N)*F(N)$

ここで、F(N)は時刻Tにおける流量、F(N+1)は時刻T+δTにおける流量

差分化(その2)

$\{F(N+1) - F(N)\}/\delta T = -\alpha*F(N+1)*F(N+1)$より、

$F(N+1) = \{-1+SQRT(1+4*\delta T*\alpha*F(N)\}/(2*\delta T*\alpha)$

以上の結果を比較したものがFig.2.3のプロット図である。この図からわかるように単純差分式に比べ、本手法を用いた解法が理論解に近い結果となっており、本手法の妥当性を確認することができた。

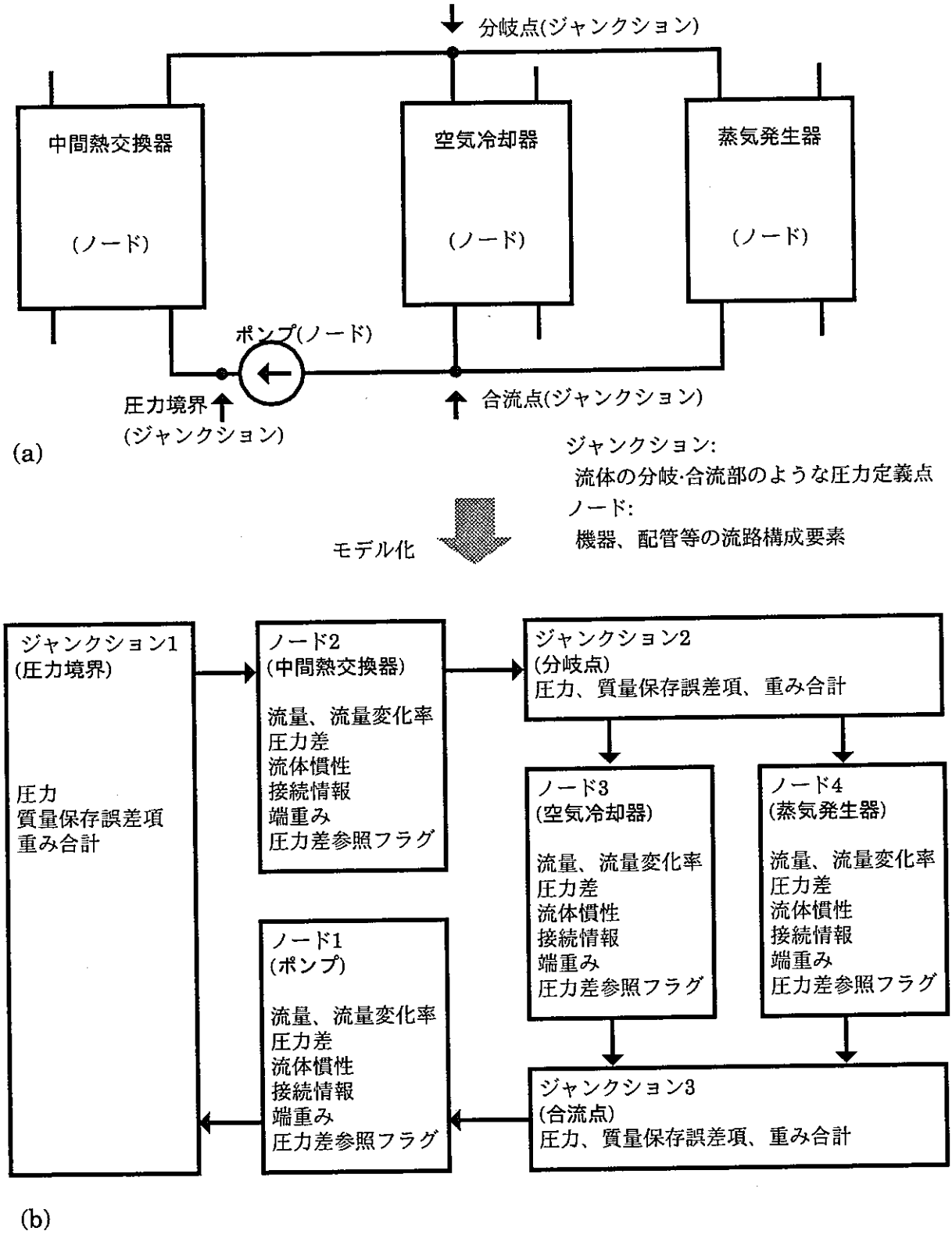

Fig. 2.1 簡略化した2次主冷却系ループのモデル化

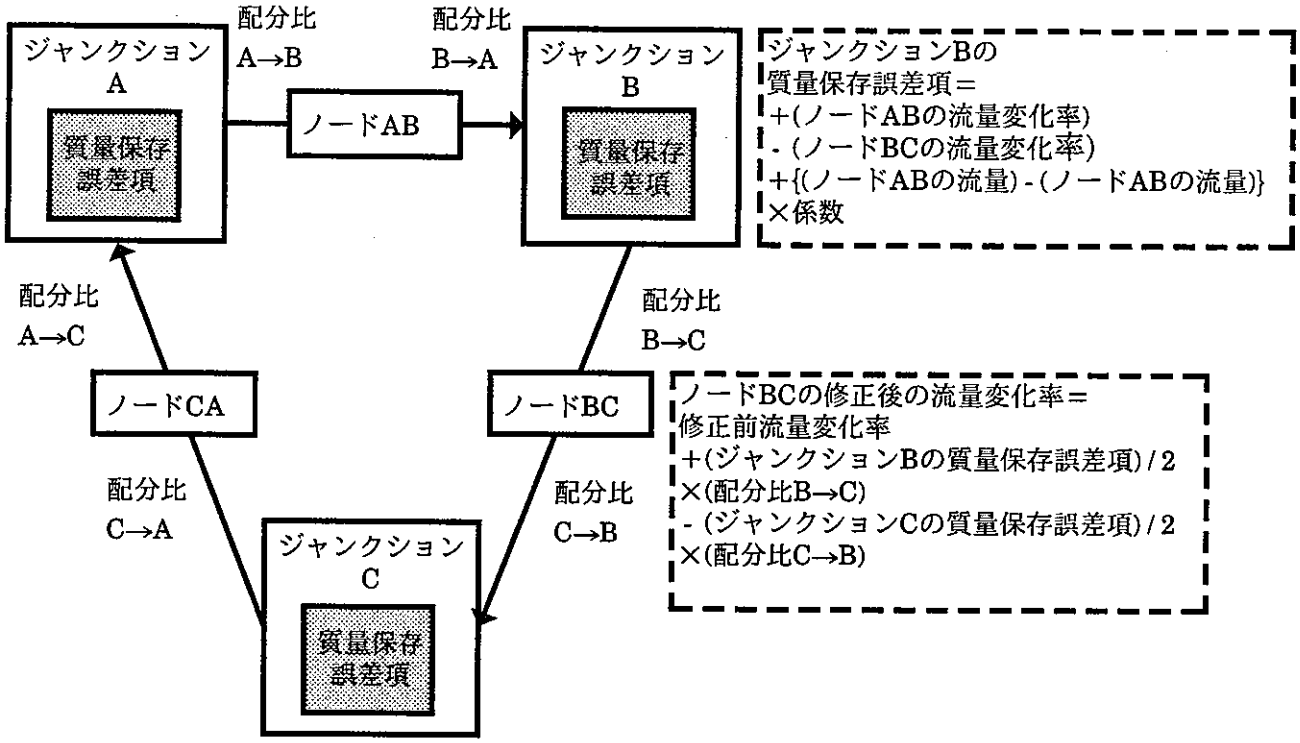

Fig. 2.2 ジャンクションにおける質量保存誤差の配分方法

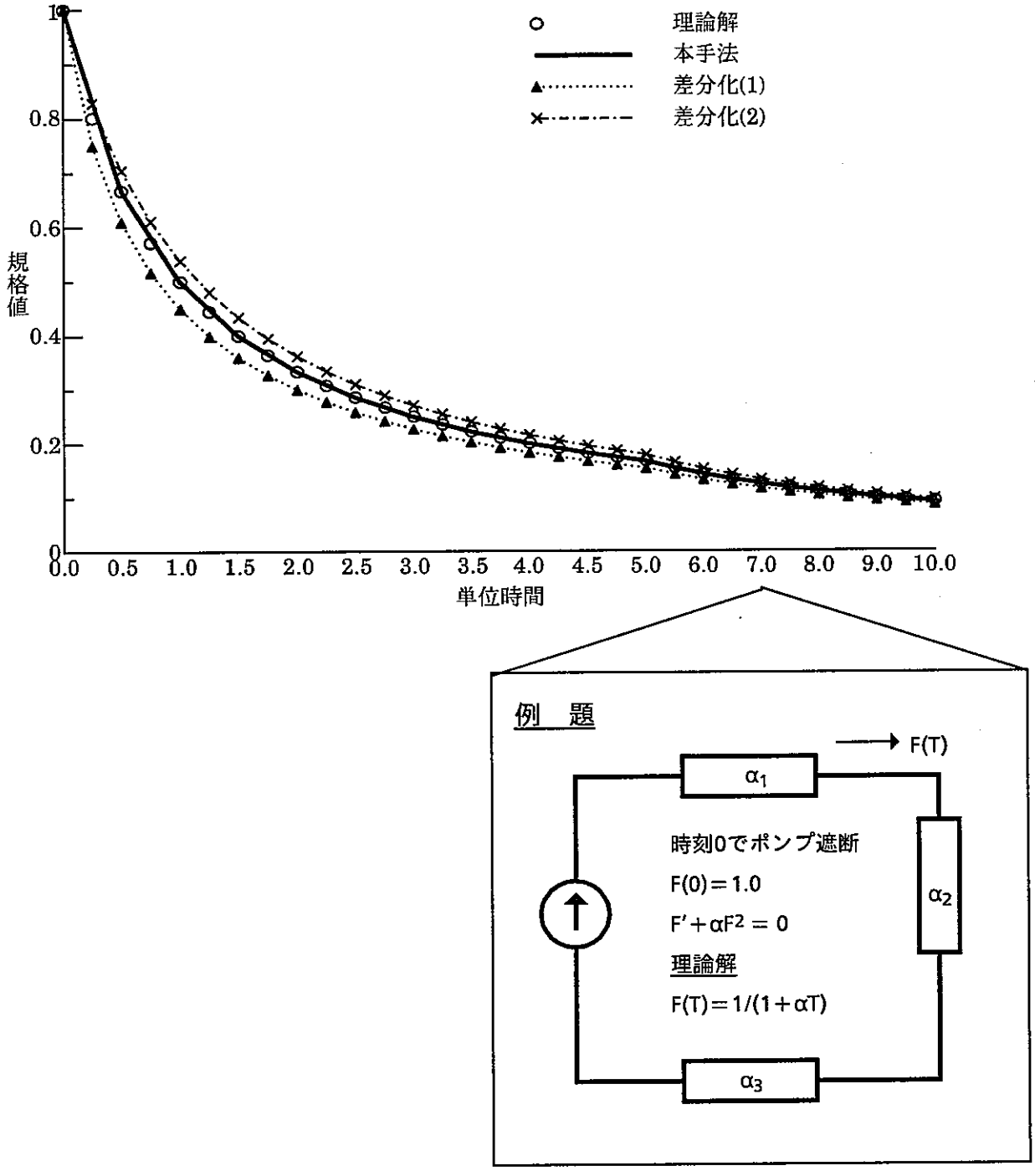

Fig. 2.3 計算手法の妥当性

3. シミュレーションシステムの試作

2章で示したアルゴリズムを実装したシミュレーションシステムをFig.3.1に示す。このシステムは、Windows環境で利用されている汎用のプログラム開発環境("*Delphi*")[3]を用いて作成したものである。プログラミング上の特徴としては、隣接する機器の結合状態を"動的"に管理するため、各ノード(機器モデル)は隣接するジャンクションの情報を格納するモデル構造(詳しくはAppendix-1参照)とするとともに、Fig.3.2に示すようにプラントシミュレーションモデルの開発時及び実行時に、こうした情報を画面上から直接指定できるようしている。

3.1 システムの概要

今回試作したシステムは主要な課題である分散協調的流動計算に加え、伝熱計算機能を有する機器モデル(ノード)も作成した。近年のプログラム開発では、このようなモデルをオブジェクトとして扱う、いわゆる"オブジェクト志向"型の開発アプローチが一般的になりつつある。こうしたアプローチは本報で提唱している分散協調型計算と最も適合するものである。さらにこのオブジェクト自体に内部状態の表示機能を持たせ、よりモジュール化の度合いを高めている。

"*Delphi*"では、上記の様なモジュール化されたオブジェクトをComponent[3]と呼ばれる単位で作成できる。Componentは、オブジェクト志向型言語C＋＋等における型(クラス)の概念に相当している。また、画面上からも見えるComponentはVisualComponentと呼ばれ、Windows環境でのプログラム部品としての再利用も可能になる。

作成したComponent[Class]を以下に示す。基本Componentの*WinCOMP*からプラントシミュレーションに必要な各種要素を派生させ、そこに内部状態(プロセス量)の表示機能を個々に付加している。Fig.3.3に各要素の表示画面を示す。

WinCOMP	基本要素	(VisualComponent, 流動計算)
├─(継承)─> *WinJunc*	Junction要素	(流動計算、伝熱計算)
├─(継承)─> *WinHXer*	熱交換器要素	(伝熱計算)
├─(継承)─> *WinPipe*	配管要素	(伝熱計算)
├─(継承)─> *WinPump*	ポンプ要素	(流動計算)
├─(継承)─> *WinValv*	弁要素	(流動計算)
WinPlant	計算制御要素	(非VisualComponent)

3.2 Componentの機能

各Componentは以下の機能を実現している。

(1) *WinCOMP*の機能

プラント機器(*WinHXer, WinPipe, WinPump, WinValv*)用流動計算機能の実装

→圧力損失計算、流量変化率算出等

下位の要素はここで実装された機能を利用する

(2) *WinJunc*の機能

接続状態同定機能の実装

→当該ジャンクションと接続しているノード(プラント機器)の同定を行い、マスバランスの誤差配分処理、圧力伝播処理のための情報を獲得する

接続ノードへのマスバランス誤差配分比計算の実装

圧力伝播処理の実装

マスバランス計算の実装

接続ノードの流量計算機能のprocedure呼出し

(3) *WinHXer*の機能

熱交換器内伝熱計算機能の実装

熱交換器内温度分布表示機能の実装

property指定機能(パラメータの設定など)の実装

(4) *WinPipe*の機能

配管内伝熱計算機能の実装

配管内温度分布表示機能の実装

property指定機能の実装

(5) *WinPump*の機能

ポンプ吐出圧力特性の実装

回転数表示機能の実装

property指定機能の実装

(6) *WinValv*の機能

弁圧力損失特性(複数)の実装

弁開度表示機能の実装

property指定機能の実装

(7) *WinPlant*の機能

反復計算機構の実装→スレッドの採用

*WinCOMP*の下位要素の計算制御(計算停止･再開)機能の実装

*WinCOMP*の下位要素動的作成機能の実装

Summary(境界値等)表示機能等の実装

3.3 利用方法

試作システムは通常のWindowsプログラムとして動作することは勿論のこと、Fig.3.4に示すようにWWWブラウザ(InternetExploer,NetScapeNavigator等)上でも動作可能である。システムの基本的な操作方法は以下の通りである。

1)プログラム機動後、画面右下のExceute内の[Initialize]ボタンをクリックする。これにより初期計算が実行される。

2)初期計算の後、[Steady]ボタンをクリックして定常計算を実行する。
定常計計算は境界条件の元で各部流量、温度が収束条件に達するまで行われる。計算中はExceute下に"In Steady"と表示されている。定常計算が終了するとこの表示が"now Steady"に変わり、画面上で各部の温度分布表示が行われる。

3)過渡計算に移行するには[Resume]ボタンをクリックする。クリック後は、このボタンの表示はSuspendと表示され、この状態でクリックすると計算は一時停止する。以後ボタンをクリックするたびに再開･停止を行うことができる。

4)過渡計算中には、画面右下の9種類の操作量を変更することができる。操作量の変更は瞬時にシミュレーションプログラムに反映されるので、その結果を画面上で確認できる。
また、[Summary]ボタンをクリックすると、その時点の各コンポーネントの計算状態一覧が表示される。

5)終了するには[閉じる]ボタンをクリックする。

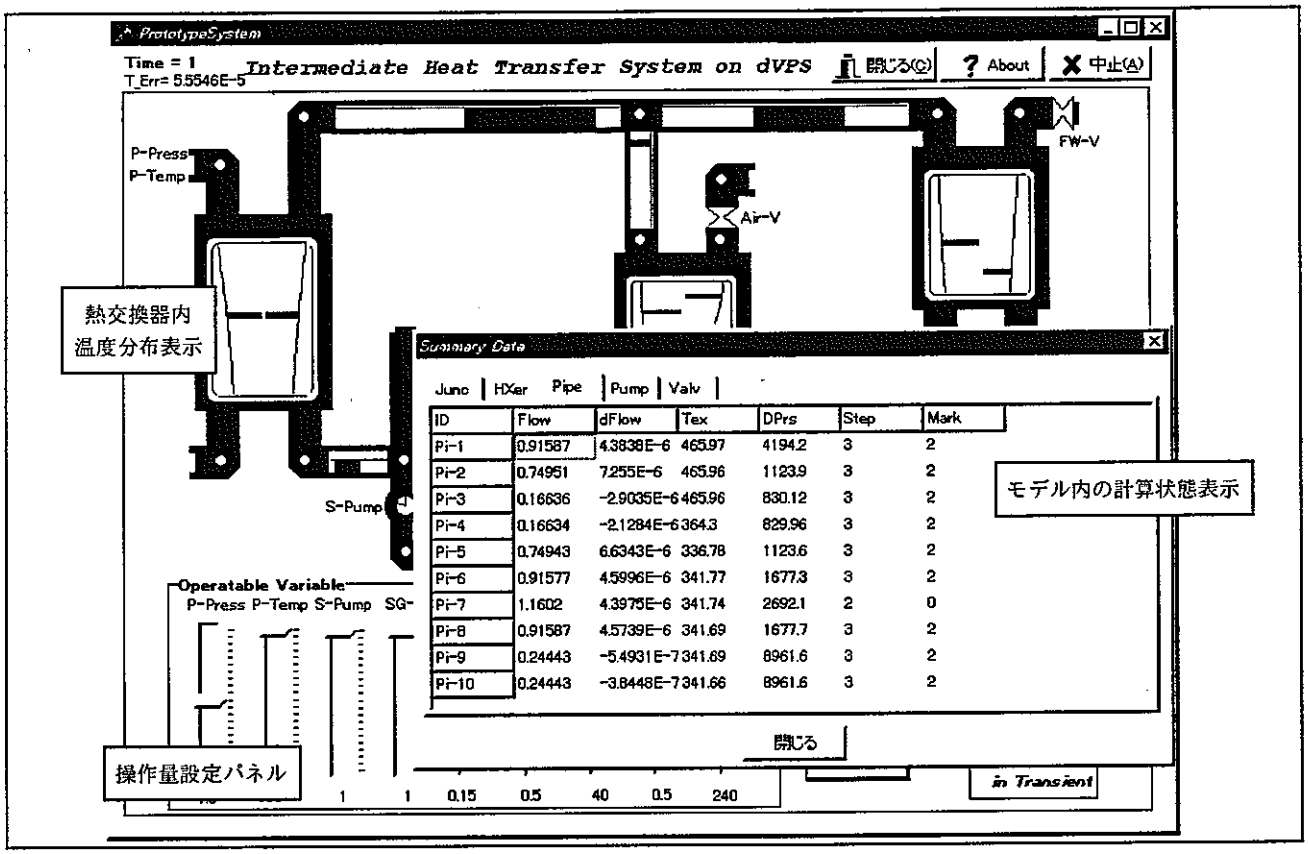

Fig. 3.1 分散協調型流動計算手法を実装したシミュレーションシステム

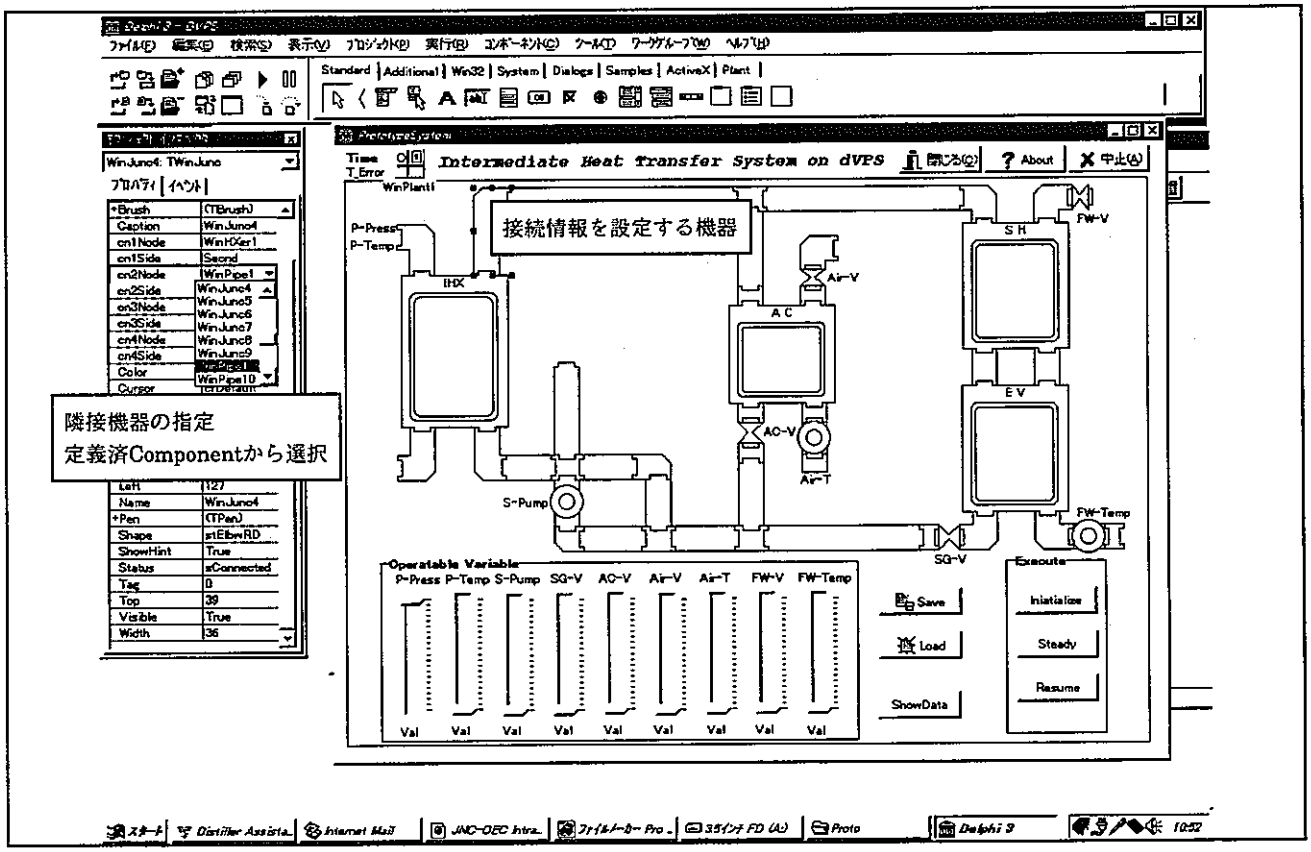

Fig. 3.2 画面上からの隣接機器接続情報の指定(モデル開発時)

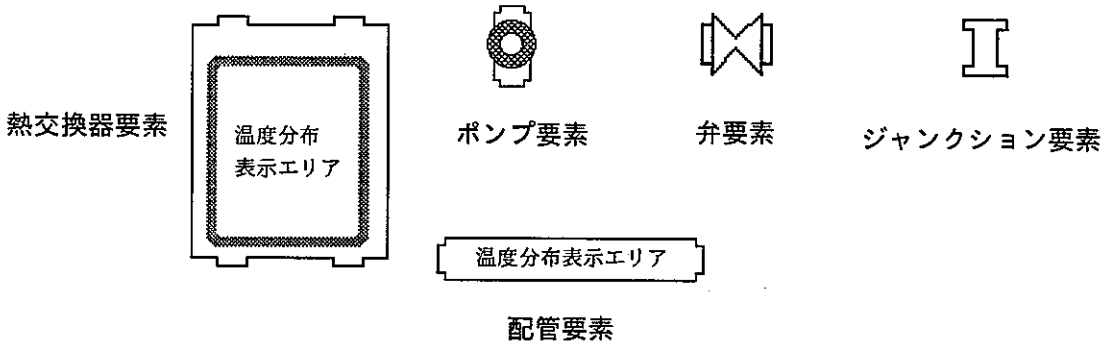

Fig. 3.3 作成したVisual Componetnt

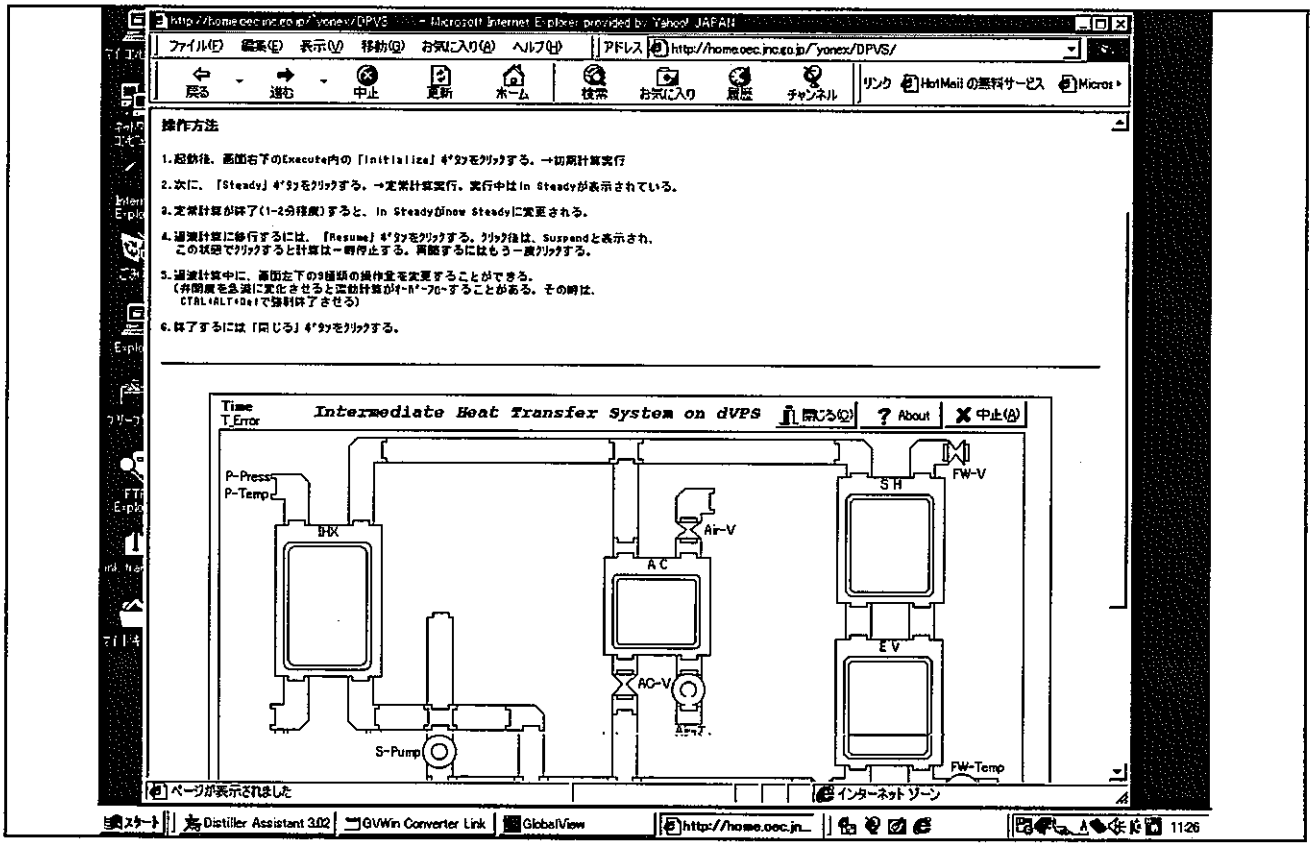

Fig. 3.4 WWW上からのシミュレーション･システムの実行

4. 今後の課題

本報で試作したプラントシミュレーションシステムは、以下に示す課題･制約を残しており、今後改良の余地がある。

(1) 急激な流動変化に対してる計算の安定性

例えば急激に弁開度を絞った場合などには、演算のオーバーフローが発生している。これは時間刻み幅を固定としているためで、時間変化率に応じて時間刻み幅の修正、あるいは、質量保存誤差項算定における係数調整等の機能を追加する必要がある。

(2) 通信機能の実装

複数の実行環境を連携したシミュレーションを実現するために、ソケット機構による通信機能を組み込む必要がある。この場合、より自律的システムとするにはマルチキャスト通信[4]を用いることが望ましい。マルチキャスト通信とは、グループ同報通信とも呼ばれ、1回のデータパケット送信によって、複数の通信相手(計算機)が同時にそのデータパケットを受信できる機構で、通常の1対1の通信(ユニキャスト通信)に比べ通信効率を高めることが可能となる。

(3) データのSave/Loadが不完全

計算途中での中断及び再開のためにデータ保存･初期化機能が必要となる。現システムでは保存は可能であるが、保存結果からの再開計算が正しく行われていない。

5. 結言

原子力プラントに代表される、複雑な流路を有する大規模な工学システムの設計、評価に不可欠なプラントシミュレーションにおいて必須の非定常の流動計算を、従来のような行列計算を用いないで行なうアルゴリズムを具体化し、高速炉の二次主冷却系構成を対象に本アルゴリズムを検証した。

流動計算における収束計算量の増加は、機器モジュールの伝熱計算量に比べて十分小さく、流路ネットワーク構造の動的変化への対応等の本アルゴリズムの利点を損なわないことを確認した。

本手法により、プラントの全体モデル、特に流路の破断や隔離のように、境界条件の構造そのものが不連続に変化する現象を模擬可能なモデルが容易に構築可能となり、シミュレーションの柔軟性の向上が図れる。また、従来こうした全体モデルは均質な開発環境下(言語、計算機)での製作に限定されていたが、プラント構成要素毎の機器モジュールを独立した開発環境で製作し、これらの連結によって全体モデルへ統合することも可能となり、ソフトウェア開発･保守のマンパワーの低減が期待できる。

6. 参考文献

[1] 須田 一則, 大草 享一, 他, ビルディングブロック型FBRプラントシミュレータの開発及び検証評価 : PNC ZN9410 95-351 (1995)

[2] 中村 明子, 伊藤 文子, FORTRAN数値計算とプログラミング : 共立出版, 東京(1970)

[3] Imprise社編, Object Pascal言語ガイド ,アプリケーション開発(開発者ガイド) : Delphi3.1マニュアル(1997)

[4] Wide Project編, インターネット参加の手引 1996年度版 ,pp.296-304 :共立出版(Bit別冊), 東京(1996)

Appendix-1 プロトタイプシステムプログラム仕様

A-1 プロトタイプシステムの概要

作成したプロトタイプシステムは以下の特徴を持っている。

◎ 各要素はDelphiの設計時にパラメータが指定されており、そのパラメータによって初期化されるため、新たにデータ入力の必要はない。また、計算中に変更することも可能としている。

◎ 流動計算はノード/ジャンクション法により行う。ノードとなれるのは熱交換器要素(*WinHXer*)、配管要素(*WinPipe*)、ポンプ要素(*WinPump*)、弁要素(*WinValv*)である。ジャンクション(*WinJunc*)は上記の要素間を接続するものである。但し、境界点となるジャンクションには一つだけノードが接続される。

◎ ノード/ジャンクションの接続情報は各々の要素自体に持たせている。この接続情報は、ノードに関しては上流側ジャンクションの名称(Delphi内部ではポインタで表現される)、ジャンクションに関しては接続するすべてのノードの名称およびノードの端点(熱交換器要素の場合、出入口部分)が相当する。このため、流動計算のフローはジャンクションを中心に行っている。すなわち、各々のジャンクションに接続されたノードの圧力損失を求め、これをもとにジャンクションの圧力分布を暫定的に定めてからジャンクションのマスバランス計算を行い、その誤差を補正項として接続されたノードの流量変化率を調整して流量を求めている。また、Fig.A-1に示す機構により、ノード/ジャンクションの接続状態が動的に変化しても計算が対応できるように、データ駆動型のプログラム構成としている。

◎ 伝熱計算は熱交換器要素、配管要素のみで行う。ジャンクション部では温度境界の結合(連結した要素に対して)と合流部でのミキシング(瞬時混合:算術平均)を行う。

◎ 計算制御部(*WinPlant*)では上記の個々の計算手順を隠蔽した抽象化したメソッドを定義している。具体的には初期処理、整定計算処理、過渡計算処理など。

◎ 過渡計算時は内部的にはスレッドを使用している。また、こうした繰返し処理をタイマー要素を用いている。これらの機能を組み込んでいるので、ユーザーは数種類のパラメータのみ指定するだけでよい。

◎ 流動計算と伝熱計算は4対1のタイムステップで計算している。

◎ ジャンクションに接続するノードに対しての計算は、ジャンクションから見て下流側のノードを対象としている。この判定は、当該ジャンクションに接続しているノードが有する接続情報(上流側ジャンクション)を調べ、それが自身であればそのノードは下流側であると判断する。

A-2 コンポーネント仕様

以下のコンポーネントに関して、そのメンバー変数、メンバー関数･手続き、プロパティの一覧を、Table A-1~A-7 に示す。また、Fig. A-2 にWinCOMPのリスト(抜粋)を示す。

WinCOMP	基本要素	(VisualComponent, 流動計算)
+--(継承)--> *WinJunc*	Junction要素	(流動計算、伝熱計算)
+--(継承)--> *WinHXer*	熱交換器要素	(伝熱計算)
+--(継承)--> *WinPipe*	配管要素	(伝熱計算)
+--(継承)--> *WinPump*	ポンプ要素	(流動計算)
+--(継承)--> *WinValv*	弁要素	(流動計算)
WinPlant	計算制御要素	(非VisualComponent)

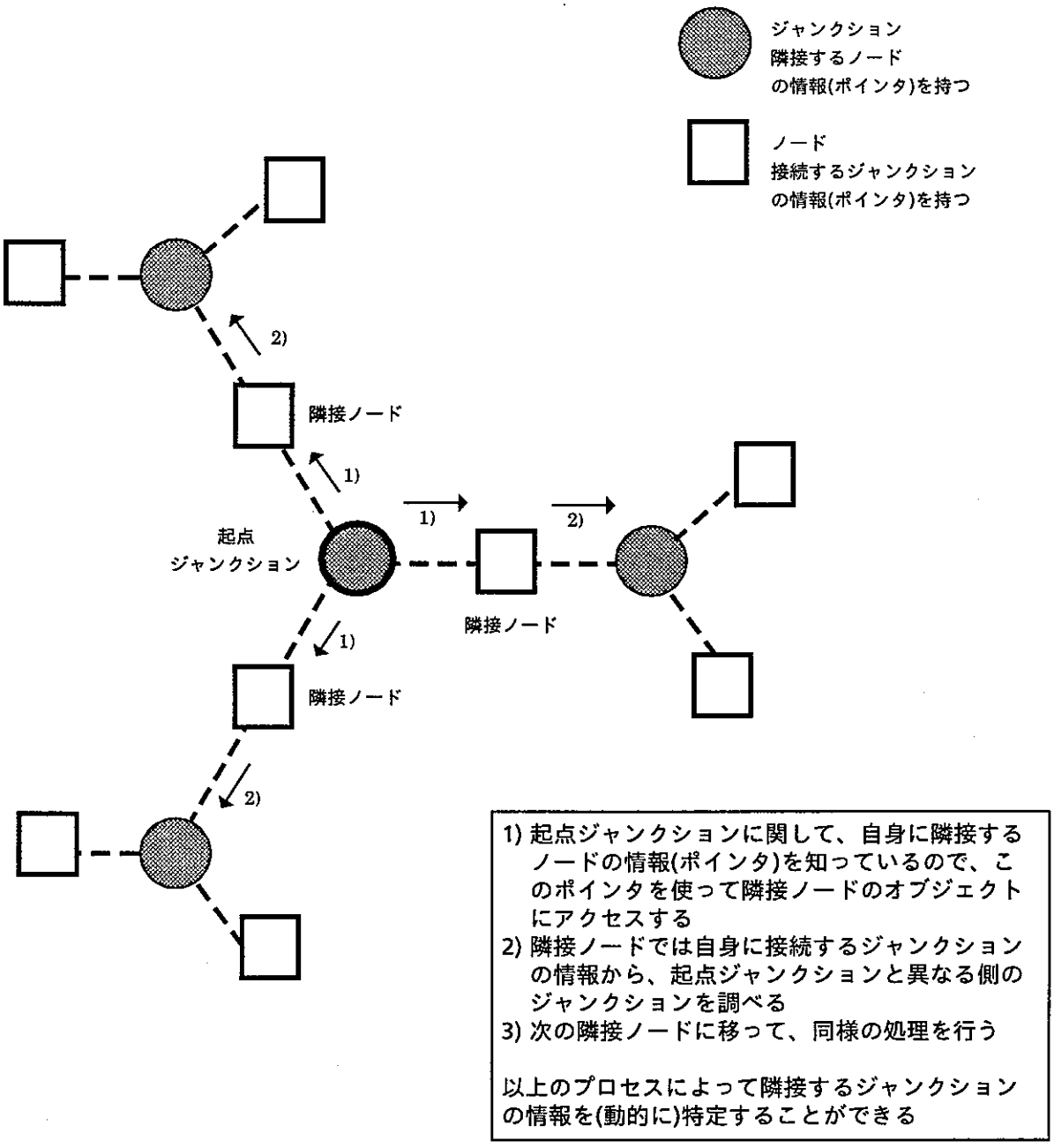

Fig. A-1 接続情報の同定機構

```
unit WinCOMP;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

const
 NumJuncEntry= 4;                    {Junc:最大接続数}
 NumProfile= 1000;                   {Pipe:最大Step数}
 NumDivHX= 20;                       {HX:最大分割数}
 NumSide= 2;                         {コンポーネント最大次数}
 clStatus1= clLime;                  {Init時カラー}
 clStatus2= clAqua;                  {Steady時カラー}
 clStatus3= clTeal;                  {Transient時カラー}
 clStatus4= clYellow;                {Connected時カラー}

var
 deltaT: Single;                     {時間刻み巾}
 RefFlow: Single;                    {基準流量(kg/sec)}
 Max_Err: array[1..3] of Single;     {流動,伝熱計算最大誤差:収束判定}
 ErrLimit: array[1..3] of Single;    {流動,伝熱計算許容誤差}
 clErrLevel: array[1..7] of TColor;  {流動計算誤差Color}
 DTempScale: array[1..2] of Single;  {表示用温度スケール:最大/最小値}
 ioFile: TFileStream;                {入出力用FileStream}
 MoveFlag,MoveX,MoveY: Integer;      {Component移動用}

type
 COMPType = (aHX, aPipe, aPump, aValve, aJunc);
 ShapeType = (adumy1, adumy2);
 ExeStatus = (sNone, sInit, sSteady, sTransient, sConnected, sDisconnect,sCaption, sProperty,
sMisc);
 COMPSide = (none, Prim, Secnd);
 DispType= (dpError, dpData);

 TWinCOMP = class(TGraphicControl)                    {TCustomControl}
 private
  {Private 宣言}
 protected
  {PrFPen: TPen;
  FBrush: TBrush;
  FShape: ShapeType;
  FGap,FGap2: Integer;
  FStatus: ExeStatus;                 //計算モード
  FNoSide: Integer;                   //コンポーネントの次数:1..2次
  FCn: array[1..NumSide] of TWinCOMP;                 //接続コンポーネント情報
  FInertia: array[1..NumSide] of Single;              //流体慣性
  FRefDP: array[1..NumSide] of Single;                //基準圧損(kg/m2)
  Fract: array[1..NumSide,1..2] of Single;            //誤差配分比,上流側/下流側
  dFlowdt: array[1..NumSide] of Single;               //流量変化
  D_Press: array[1..NumSide] of Single;               //圧力損失
  DP_Fixed: array[1..NumSide] of Boolean;             //圧力損失固定Flag
  function GetRefDP(Index: Integer): Single;
  function GetNoSide: Integer;
  procedure Paint; override;
  procedure DrawCaption;
  procedure SetPen(Value: TPen);
  procedure SetBrush(Value: TBrush);
  procedure SetShape(Value: ShapeType);
  procedure SetGap(Value: Integer);
  procedure SetGap2(Value: Integer);
  procedure SetFCn(Index: Integer; Value: TWinCOMP);
  procedure SetRefDP(Index: Integer; Value: Single);
 public
  {Public 宣言}
  Flow: array[1..NumSide,1..2] of Single;             //normarized流量,now/next
  TempIn: array[1..NumSide] of Single;                //温度;入口境界
  TempEx: array[1..NumSide] of Single;                //温度;出口境界
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  function  GetFCn(Index: Integer): TWinCOMP;         {接続情報取得}
  function  GetInertia(Index: Integer): Single;       {Inertia値取得}
  function  GetFract(Index,Index2: Integer): Single;{  配分比取得}
  function  GetdFlowdt(Index: Integer): Single;       {流量変化率取得}
  function  GetD_Press(Index: Integer): Single;       {圧力損失取得}
  procedure DisConnect;                               {接続解除}
  procedure SetFStatus(Value: ExeStatus);             {モード設定}
  procedure SetFract(Index,Index2: Integer;Value: Single);{配分比設定}
  procedure SetDP_Fixed(Index: Integer; Value: Boolean); {圧損係数設定}
  procedure Pressure_Drop; virtual;                   {圧力損失算出}
  procedure CdFlowdt(Side: COMPSide; DeltaPress: Single); virtual; {流量変化算出}
  procedure AdjustFlow(MEupper,MEdown: Single);       {流量変化補正}
  procedure SetNewFlow;                {算出流量更新}
  procedure Init;                      {初期化処理}
  procedure Steady_Pre;                {整定計算:前処理}
  procedure Transient_Pre;             {過渡計算:前処理}
  procedure HeatCalc;                  {伝熱計算}
  procedure MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  procedure MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
  procedure MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
  property  Shape: ShapeType read FShape write SetShape;
```

Fig. A-2 WinCOMPソースリスト(抜粋-1/4)

```
 published
  {Published 宣言}
  ----< 略 >---

constructor TWinCOMP.Create(AOwner: TComponent);
begin
 inherited Create(AOwner);
 Width:= 32; Height:= 24; FGap:= 4; FGap2:= 16;
 FPen:= TPen.Create;
 FPen.OnChange:= StyleChanged;
 FBrush:= TBrush.Create;
 FBrush.OnChange:= StyleChanged;
 ShowHint:= True;
 FNoSide:= 1;
 OnMouseMove:= MouseMove;
 OnMouseUp := MouseUp;
 OnMouseDown:= MouseDown;
end;

destructor TWinCOMP.Destroy;
begin
 FPen.Free;
 FBrush.Free;
 inherited Destroy;
end;

  ----< 略 >---

function TWinCOMP.GetFCn(Index: Integer): TWinCOMP;
begin
 Result:= FCn[Index];
end;

procedure TWinCOMP.SetFCn(Index: Integer; Value: TWinCOMP);
var
 indx,i: Integer;
 xJunc: TWinJunc;
 aMsg: string;
begin
 if (csLoading in ComponentState) then
  FCn[Index]:= Value
 else //設計or実行モード
  begin;
   if ((Fcn[Index]<>Value)and(Value is TWinJunc))or(Value=nil) then
    begin   //有効接続先はTWinJuncのみ
     //変更前の接続先とのLink解除(if有)
     if FCn[Index]<>nil then
      begin
       xJunc:= FCn[Index] as TWinJunc;
       for i:= 1 to NumJuncEntry do
        if xJunc.GetJnNode(i)<>nil then
         if xJunc.GetJnNode(i).Name=Self.Name then indx:= i;
       aMsg:= 'Disconnect to '+xjunc.name+'.Jn['+IntToStr(indx)+']'#13#13;
       xJunc.SetJnNode2(indx,nil);
       xJunc.SetJnSide(indx,none);
      end;
     //新接続先とのLink設定
     if Value<>nil then //接続先とのLink設定
      begin
       xJunc:= Value as TWinJunc;
       indx := 0;
       for i:=NumJuncEntry downto 1 do
        if xJunc.GetJnNode(i)=nil then indx:= i;
       if indx<>0 then
       begin
        ShowMessage(aMsg+'Connect to '+xjunc.name+'.Jn['+IntToStr(indx)+']');
        xJunc.SetJnNode2(indx,Self);    //接続先とのLink設定
        if Index=1 then xJunc.SetJnSide(indx,Prim);
        if Index=2 then xJunc.SetJnSide(indx,Secnd);
        FCn[Index]:= Value;          //接続先と接続
        SetFStatus(sConnected);       //接続状態へ変更
       end
       else ShowMessage('Connect Full!!'); //接続先Juncの接続端がfull!!
      end
      else          //接続先とのLink解除 Value=nil
       begin
        FCn[Index]:= nil;
        if (FCn[1]=nil)and(FCn[2]=nil) then
         begin
          SetFStatus(sNone);  //接続状態解除
          ShowMessage(aMsg+'Now No Connection');
         end;
       end;
     end;
    end;
  end;

procedure TWinCOMP.DisConnect;
begin
 SetFCn(1,nil); SetFCn(2,nil);
end;
```

Fig. A-2 WinCOMPソースリスト(抜粋-2/4)

```
function TWinCOMP.GetInertia(Index: Integer): Single;
begin
 Result:= FInertia[Index];
end;

function TWinCOMP.GetRefDP(Index: Integer): Single;
begin
 Result:= FRefDP[Index]*1.0e-4;
end;

function TWinCOMP.GetNoSide: Integer;
begin
 Result:= FNoSide;
end;

procedure TWinCOMP.SetRefDP(Index: Integer; Value: Single);
begin
 FRefDP[Index]:= Value*1.0e+4;
end;

function TWinCOMP.GetFract(Index,Index2: Integer): Single;
begin
 Result:= Fract[Index,Index2];
end;

procedure TWinCOMP.SetFract(Index,Index2: Integer;Value: Single);
begin
 Fract[Index,Index2]:= Value;
end;

function TWinCOMP.GetdFlowdt(Index: Integer): Single;
begin
 Result:= dFlowdt[Index];
end;

function TWinCOMP.GetD_Press(Index: Integer): Single;
begin
 Result:= D_Press[Index];
end;

procedure TWinCOMP.Pressure_Drop;
var
 i: Integer;
begin
 for i:=1 to GetNoSide do
  D_Press[i]:= FRefDP[i]*(Flow[i,1]+Flow[i,2])
              *Abs(Flow[i,1]+Flow[i,2])/4
              + dFlowdt[i]*FInertia[i]*RefFlow;
end;

procedure TWinCOMP.SetDP_Fixed(Index: Integer; Value: Boolean);
begin
 DP_Fixed[Index]:= Value;
end;

procedure TWinCOMP.CdFlowdt(Side: COMPSide; DeltaPress: Single);
var
 i: Integer;
begin   //圧力差より流量変化率算出
 i:= Ord(Side);
 if (not DP_Fixed[i])and(FInertia[i]<>0) then
  dFlowdt[i]:= ( DeltaPress - FRefDP[i]*(Flow[i,1]+Flow[i,2])
     *Abs(Flow[i,1]+Flow[i,2])/4 )/(FInertia[i]*RefFlow);
end;

procedure TWinCOMP.AdjustFlow(MEupper,MEdown: Single);
var
 i: Integer;
begin
 for i:=1 to GetNoSide do                    //流量変化率を質量保存誤差から補正
 begin
  dFlowdt[i]:= dFlowdt[i] + Fract[i,1]*MEupper/2 - Fract[i,2]*MEdown/2;
  //次ステップ の流量算出
  Flow[i,2]:= Flow[i,1]+dFlowdt[i]*deltaT;
  //最大流量変化率の取得
  if Max_Err[2]<Abs(dFlowdt[i]) then Max_Err[2]:= Abs(dFlowdt[i]);
 end;
end;

procedure TWinCOMP.SetNewFlow;
var
 i: Integer;
begin
 for i:=1 to GetNoSide do
  Flow[i,1]:= Flow[i,2];                     //算出流量をCurrent流量
end;

procedure TWinCOMP.Init;
var
 i: Integer;
begin                                       //初期化処理:共通部分
 SetFStatus(sInit);
```

Fig. A-2 WinCOMPソースリスト(抜粋-3/4)

```
 for i:=1 to GetNoSide do
 begin
  Flow[i,1]:= 0; Flow[i,2]:= 0;
  dFlowdt[i]:= 0;
  SetDP_Fixed(i, False);
 end;
end;

procedure TWinCOMP.Steady_Pre;       //整定計算:前処理
var
 i: Integer;
begin
 SetFStatus(sSteady);
 for i:=1 to GetNoSide do SetDP_Fixed(i, False);
end;

procedure TWinCOMP.Transient_pre; //過渡計算:前処理
var
 i: Integer;
begin
 SetFStatus(sTransient);
 for i:=1 to GetNoSide do SetDP_Fixed(i, False);
end;

procedure TWinCOMP.HeatCalc;         //境界条件入力
var
 i: Integer;
begin
 for i:=1 to GetNoSide do
 begin
  if FCn[i]<>nil then
  begin
   TempIn[i]:= FCn[i].TempEx[1];
   TempEx[i]:= TempIn[i];
  end;
 end;
end;

procedure TWinCOMP.MouseMove(Sender: TObject;
     Shift: TShiftState; X, Y: Integer); //コンポーネント移動中
begin
 if (ssShift in Shift)and(MoveFlag=1) then
 begin
  Self.Top := Y + MoveY;
  Self.Left:= X + MoveX;
  MoveY := Self.Top;
  MoveX := Self.Left;
 end;
end;

procedure TWinCOMP.MouseUp(Sender: TObject; Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer); //コンポーネント移動終了
begin
 if ssShift in Shift then
 begin
  MoveFlag:= 0;
  Screen.Cursor := crDefault;
 end;
end;

procedure TWinCOMP.MouseDown(Sender: TObject; Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer); //コンポーネント移動開始
begin
 if ssShift in Shift then
 begin
  MoveY := Self.Top;
  MoveX := Self.Left;
  MoveFlag:= 1;
  Screen.Cursor:= crDrag;
 end;
end;

end.
```

Fig. A-2 WinCOMPソースリスト(抜粋-4/4)

Table A-1 WinCOMP仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
FPen	TPen	Canvas用ペン色	GetRefDP	基準圧力損失取得	Paint	コンポーネント描画	Pen	Canvas用ペン
FBrush	TBrush	Canvas用ブラシ色	GetNoSide	コンポーネント接続数取得	DrawCaption	キャプション	Brush	Canvas用ブラシ色
FShape	ShapeType*	コンポーネント外観	GetFCn	コンポーネント接続情報取得	SetPen	Canvasペン設定	Gap	コンポーネント形状寸法
FGap	Integer	コンポーネント外観寸法	GetInertia	流体慣性取得	SetBrush	Canvasブラシ設定	Gap2	コンポーネント形状寸法
FGap2	Integer	コンポーネント外観寸法	GetFtract	質量誤差分配比率取得	SetShape	コンポーネント形状設定	Caption	コンポーネントキャプション
FStatus	EXEStatus	コンポーネントの状態	GetD_Press	圧力損失取得	SetGap	コンポーネント形状寸法	Color	コンポーネント色
FNoSide	Integer	コンポーネント接続端数			SetGap2	コンポーネント形状寸法	Visible	コンポーネント表示on/off
FCn[]	TWinCOMP*	コンポーネント接続情報			SetFCn	接続情報設定	ShowHint	Hint表示機能on/off
FInertia[]	Single	流体慣性			SetRefDP	基準圧力損失設定	Hint	Hint表示文字列
FRefDP[]	SIngle	基準圧力損失(100%時)			DisConnect	接続情報解除	OnMouseMove	MouseMoveイベント
Fract[]	Single	質量誤差分配比率			SetFStatus	状態設定	OnMouseUp	MouseUpイベント
dFlowdt[]	Single	流量変化率			SetFract	質量誤差分配比率設定	OnMouseDown	MouseDownイベント
DP_Fixed[]	Single	圧力損失固定Flag			SetDP_Fixed	圧力損失固定Flag設定	Staus	コンポーネント状態
Flow[,2]	Single	流量[Normarized]			Pressure_Drop	圧力損失計算	cn1Node	1次側端接続先
TempIn[]	Single	入口温度			CdFlowdt	流量変化率計算		
TempEx[]	Single	出口温度			AdjustFlow	流量変化率補正計算		
					SetNewFlow	流量計算		
					Init	初期化処理		
					Steady_Pre	整定計算前処理		
					Transient_Pre	過渡計算前処理		
					HeatCalc	伝熱計算		
					MouseMove	マウスMoveイベント処理		
					MouseUp	マウスUpイベント処理		
					MouseDown	マウスDownイベント処理		
					StyleChanged	コンポーネント形状変更時処理		

*ユーザー定義型、[]は配列を表す

Table A-2 WinHXer仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
Temp[,,]	Single	内部温度	GetmLength	伝熱長さ取得	SetmLength	伝熱長さ設定	hx1_SpecHeat	1次側比熱
Coef[,]	Single	行列式係数	GetMark	流体位置取得	CoefSet_S	整定計算用係数設定	hx1_Area	1次側断面積
Rside[]	Single	行列式係数			CoefSet_T	過渡計算用係数設定	hx1_Density	1次側比重量
Triag[]	Single	行列式係数			Former_Half	行列式処理	hx1_Inertia	1次側流体慣性
Area[2]	Single	断面積			Latter_Half	行列式処理	hx1_RefDP	1次側基準圧力損失
Alpha[2]	Single	熱伝達係数			Revice_Matrix	行列式処理	hx2_SpecHeat	2次側比熱
Cp[2]	Single	比熱			Move_Mark	流体位置算出	hx2_Area	2次側断面積
Rho[2]	Single	比重量			Steady	整定計算	hx2_Density	2次側比重量
Htr	Single	熱貫流率			Transient	過渡計算	hx2_Inertia	2次側流体慣性
mLength	Single	伝熱長さ			Paint	コンポーネント描画	hx2_RefDP	2次側基準圧力損失
dTempdt	Single	温度変化率			Display	データ表示	hx_Length	伝熱長さ
TempGapStep	Single				HeatCalc	伝熱計算	hx_HeatTransfer	熱貫流率
dTempdtStep	Single				TurnDrawMode	データ表示モード変更	cn2Node	2次側端接続先
FMark[2]	Single	流体位置			OnDblClickProc	ダブルクリック時処理		
FDrawMode	DrawMode*	データ表示モード			DialogOpen	ダイアログ表示処理		
					SaveFile	コンポーネントデータの保存		
					LoadFile	コンポーネントデータの読込み		

*ユーザー定義型

Table A-3 WinPipe仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
Prof[,]	Single	温度履歴	GetStep	温度履歴データ点数取得	Steady	整定計算	pp_Vol	配管体積
Step	Integer	温度履歴データ点数	GetMark	流体位置取得	Transient	過渡計算	pp_Inertia	配管流体慣性
Mark	Single	流体位置			Paint	コンポーネント描画	pp_RefDP	配管基準圧力損失
FVol	Single	配管体積			Display	データ表示	cnTrend	配管流れ方向
FTrend	TrendType*	流れの方向			HeatCalc	伝熱計算		
					OnDblClickProc	ダブルクリック時処理		
					DialogOpen	ダイアログ表示処理		
					SaveFile	コンポーネントデータの保存		
					LoadFile	コンポーネントデータの読込み		

*ユーザー定義型

Table A-4 WinPump仕様

メンバ変数	Type変数型	内容	function	内容	procedure	内容	property	内容
Rev	Single	回転数	PumpHead	吐出圧力特性	Paint	コンポーネント描画	pm_HeadMax	最大吐出圧力
HeadMax	Single	最大吐出圧力	GetValue	回転数取得	Pressure_Drop	圧力損失計算	pm_HeadRef	基準吐出圧力
HeadRef	Single	基準吐出圧力			SetValue	回転数設定	pm_RefQ	基準流量
RefQ	Single	基準流量			Display	データ表示	pm_Rev	ポンプ回転数
					Init	初期化処理		
					CdFlowdt	流量変化率算出		
					OnDblClickProc	ダブルクリック時処理		
					DialogOpen	ダイアログ表示処理		
					SaveFile	コンポーネントデータの保存		
					LoadFile	コンポーネントデータの読込み		

*ユーザー定義型

Table A-5 WinValv仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
FOpen	Single	弁開度	Qfunc	弁圧損特性	SetRefDPV	基準圧力損失設定	Shape	弁形状
FRefDPV	Single	基準圧力損失	GetRefDPV	基準圧力損失取得	Paint	コンポーネント描画	vv_RefDP	弁基準圧力損失
FProf	ValveProf*	弁圧損特性Flag	GetValue	弁開度取得	SetShape	弁形状設定	vv_Open	弁開度
FShape	ShapeType	コンポーネント外観			SetValue	弁開度設定	vv_Prof	弁圧力損失特性
					Display	データ表示		
					OnDblClickProc	ダブルクリック時処理		
					DialogOpen	ダイアログ表示処理		
					SaveFile	コンポーネントデータの保存		
					LoadFile	コンポーネントデータの読込み		

*ユーザー定義型

Table A-6 WinJunc仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
FBTYpe	BoundaryType*	境界型	GetJnNode	接続ノード取得	Distiribute	配分比率算出	JType	Junction型
JCn[]	ConnType*	接続情報	GetJnSide	接続ノード端点取得	Paint	コンポーネント描画	bdType	境界型
Pressure	Single	圧力	GetFJType	Junction型取得	SetShape	コンポーネント形状設定	bdPres	圧力境界値
P_Bound	Single	圧力境界値	GetFBType	境界型取得	Disconnect	接続情報解除	bdTemp	温度境界値
T_Bound	Single	温度境界値	GetPressure	圧力値取得	SetJnNode	接続ノード設定	cn1Node	接続端1の接続先
P_Fixed	Boolean	圧力固定Flag	GetP_Bound	圧力境界値取得	SetJnNode2	接続ノード設定(間接)	cn2Node	接続端2の接続先
M_Error	Single	質量誤差	GetT_Bound	温度境界値取得	SetJnSide	接続ノード端点設定	cn3Node	接続端3の接続先
FShape	ShapeType	コンポーネント外観	GetMError	質量誤差取得	ResetJnNode	接続ノード設定解除	cn4Node	接続端4の接続先
					SetFJType	Junction型設定	cn1Side	接続端1の接続先端
					SetFBType	境界型設定	cn2Side	接続端2の接続先端
					SetP_Bound	圧力境界値設定	cn3Side	接続端3の接続先端
					SetT_Bound	温度境界値設定	cn4Side	接続端4の接続先端
					ShowBoundary	境界型Junc表示		
					ShowUnConnect	未接続Junc表示		
					SetPumpHead	ポンプ吐出圧力設定		
					Propagate	圧力伝播処理		
					ResetP_Fixed	圧力固定解除		
					Mass_Balance	質量誤差計算		
					NPressure_Drop	接続ノードの圧力損失計算		
					NCdFlowdt	接続ノードの流量変化率計算		
					NAdjustFlow	接続ノードの流量変化率補正計算		
					NSetNewFlow	接続ノードの流量計算		
					Display	データ表示		
					Init	初期化処理		
					Steady_Ini	整定計算初期処理		
					Transient_Ini	過渡計算初期処理		

Table A-6 WinJunc仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
					HeatCalc	伝熱計算		
					OnDblClickProc	ダブルクリック時処理		
					DialogOpen	ダイアログ表示処理		
					SaveFile	コンポーネントデータの保存		
					LoadFile	コンポーネントデータの読込み		

*ユーザー定義型

Table A-7 WinPlant仕様

メンバ変数	変数型	内容	function	内容	procedure	内容	property	内容
FTime	Single	計算時間	GetnCOMP	計算要素数取得	SetErrLimit	最大誤差設定値設定	Interval	タイマインターバル
FdeltaT	Single	タイムステップ	GetErrLimit	最大誤差設定値取得	SetInterval	タイマインターバル設定	DTimeStep	タイムステップ
FCount	Integer	計算回数	GetInterval	タイマインターバル取得	SetFdelataT	タイムステップ設定	Ref_Flow	基準流量
FRefFlow	Single	基準流量	GetFCount	計算回数取得	TimerProc	タイマ処理	ErrMax1	マスバランス許容誤差
nBJ	Integer	境界Junc数	GetFdeltaT	タイムステップ取得	Init	初期処理	ErrMax2	流動計算許容誤差
iBJ	Integer	境界Junc番号			Steady_Pre	整定計算前処理	ErrMax3	伝熱計算許容誤差
nCOMP	Interger	計算要素数			Transient_Pre	過渡計算前処理	TScaleMax	温度スケール最大値
xCOMP	TWinCOMP*	要素集合			Steady	整定計算	TScaleMin	温度スケール最小値
xJunc	TWinJunc*	Junc要素集合			Transient	過渡計算	LblErrF	流動計算誤差表示Label
xPipe	TWinPipe*	配管要素集合			Suspend	計算中断	LblErrT	伝熱計算誤差表示Label
xHXer	TWinHXer*	熱交換器要素集合			Resume	計算再開	LblStts	状態表示Label
xValv	TWinValv*	弁要素集合			Display	データ表示		
xPump	TWinPump*	ポンプ要素集合			ShowBoundary	境界Junc表示		
DispMode	DispType*	表示モード			ShowUnConnect	未接続Junc表示		
FTempScale[]	Single	温度スケール			SaveFile	データ保存		
FTimer	TTimer	タイマ			LoadFile	データ読込み		
FLabel[]	TLabel	表示用Label			HXerCreate	熱交換器要素生成		
FStatus	EXEStatus*	計算状態			PipeCreate	配管要素生成		
ICnt1	Integer	反復回数			PumpCreate	ポンプ要素生成		
bCancel	Boolean	キャンセルFlag			ValvCreate	弁要素生成		
					JuncCreate	Junc要素生成		
					ShowSummary	Summary表示		
					About	AboutBox表示		

*ユーザー定義型