技術報告

400. 運転監視システムのための分散コンピューティングの枠組み

TTY ベースプログラム結合とオブジェクト指向プログラミング

鈴木 知明^{1,*}, 鍋島 邦彦¹, 滝澤 寛²*A Framework of Distributed Computing for Monitoring Systems**Integration of TTY-Based Programs and Object-Oriented Programming*

Tomoaki SUZUDO, Kunihiro NABESHIMA and Hiroshi TAKIZAWA

A new methodology to construct distributed computing systems specially targeting nuclear power plant monitoring systems is proposed. In this framework, a monitoring system is composed of multiple modules and a client that administers them. Each module is designed as a TTY-based program, and therefore has a great flexibility when it is developed. The client holds virtual modules, each of which works as an interface to a module in the remote hosts. Because the virtual modules are defined as a class in the meaning of object-oriented programming, the whole system is easily structured. A prototype of neural-network-based monitoring system has been developed utilizing this methodology, and the expected advantages have been confirmed.

KEYWORDS: distributed computing, monitoring systems, TTY-based program, object-oriented programming, neural network, software reliability, advanced information processing

I. はじめに

現在, 計算機はますます高性能化し, かつ安価になってきている。このような資源を原子力プラント運転監視に適切に応用すれば, その経済性・安全性が向上できることを疑うものはいないだろう。実際, 高度な計算機技術を応用した運転監視システムの研究がさかに行われてきている^{1,2)}。例えば, これまで炉心監視³⁾, BWR 安定性監視⁴⁾等が実プラントの運転監視システムとして採用されてきた。このような実用化されたシステムにおいて共通していることは, 監視対象が限られており, その物理モデルが確立していることである。しかしながら, プラントで起きる可能性のあるすべての現象に関して物理モデルを作成することは現実的ではない。よって, 物理モデルのないまったく意外なまたは未知の異常を検知診断するようなシステムの開発は筆者らが知る範囲では, まだ十分とはいえない。原子力プラントの安全性を向上させるには, このような物理モデルなしでも可能な監視診断技術の発展が望まれる。

物理モデルが陽に存在しない現象に対しては, ウェーブレット, スペクトル解析, カオス解析などの統計的手法や, ファジー推論, ニューラルネットワーク, 遺伝的アル

ゴリズムなどのソフトコンピューティングの応用が有効であると考えられている。このような高度情報処理技術は家電製品の制御などの他分野ではすでにその有効性が証明され, 実際に利用されている。しかしながら, それらの高度情報処理の研究成果は実際のプラントの運転監視に積極的に応用されるまでには至っていない。この原因の一つとして考えられるのが, 原子力プラントが非常に複雑で異常の種類が多岐にわたり, 高度情報処理を利用してもすべての異常をカバーすることができないこと。または, 異常の検知はできるが人間のように異常原因の同定はできないなどの, 機能に制限があることなどが挙げられる。このようなことから, 今後は統計的手法やソフトコンピューティングのみに依存するのではなく, プラント設計情報データベースや, 大規模シミュレーションコード, 監視ロボットのアクティブセンシングなどの多角的な情報を運転監視システムに統合していくことが肝要である。

もう一つの原因は, 複雑なアルゴリズムに対する信頼性への危惧である。原子力産業界では安全性を優先するため, ソフトウェアの検証や健全性確認, いわゆる V & V が重要な位置を占める。また保守的な考え方がとられることが多く信頼性が証明されているハードウェアやソフトウェアを無理に交換することを嫌う。それに, それらのハードウェアやソフトウェアのコストは民生用のものに比べてかなり高価なものである。また, 運転員の教育も新しいシステムを導入するごとにやり直す必要があり, 混乱を招きやすい。我々はこのような環境下で, 様々な情報処理を統合し複雑な計算を行うシステムを実用化していかなければ

¹ 日本原子力研究所 (Japan At. Energy Res. Inst.)

² まどシステム (Madow Systems Inc.)

(2003年 5月12日 受理; 2003年 9月12日 査読通過)

* 著者連絡先: Tel. 029-282-6077, Fax. 029-282-6122,
E-mail: suzudo@clsu3a0.tokai.jaeri.go.jp

ならない。この「プログラム統合によって生じる複雑性」とソフトウェア信頼性の間のジレンマは原子力プラント運転監視ソフトウェアに限ったことではないが、安全が最優先される対象を扱うということから問題はより重要である。

著者らは、上記のジレンマをソフトウェア開発技法の観点から最小化したいと考えており、本報はそのような動機から始まった研究の成果をまとめたものである。本報は、しかしながら、詳細設計のための仕様記述法やソフトウェア試験技法、または開発現場のソフトウェアエンジニアが用いる Computer-Aided Software Engineering ツール(通称 CASE ツール)等について議論するものでもないし、デジタル式安全保護系に要求されるようなソフトウェア健全性を議論するものでもない。これらは実際に開発するアプリケーションが具体的に決定してから考慮すればよいことである。ここでは将来開発されるべき運転監視システム一般を想定し、原子力プラントという安全が最優先されると同時に、複数の高度な情報処理が実時間で要求される特殊な環境において、ソフトウェア工学的に優れた、すなわち信頼性の高いソフトウェア統合手法を議論することが目的である。

次章で詳しく述べるが、高度な実時間情報処理のために著者らが具体的に選択した統合形態は、遠隔ホスト上の複数のモジュールをネットワーク結合してオブジェクト指向的な分散コンピューティングを実現することである。オブジェクト指向的な考えの導入は、ソフトウェアシステム全体にわたる理解のしやすさ、すなわち透明性を高めてくれる。また、オブジェクト指向言語では下位クラスを作ることにより変更や拡張が容易に行え、カプセル化や抽象化にも適している。

また、本研究においてソフトウェア信頼性向上のために留意した主な点は、次の2点である。

- (1) 事前にバイナリーレベルでの V & V が容易に行えること。よって実行時にバイナリーコードにコンパイルされる Java や HTML などのインタープリタ言語を排除した。
- (2) バグなどが出尽くしたと考えられるほど長く用いられてきたネットワーク通信法を用いる。

本研究ではさらに、モジュールとして TTY (Tele Typewriter) ベースのプログラムを採用した。TTY ベースのプログラムを実時間の分散コンピューティングのモジュールとして用いるのは、筆者らが知る限りでは他に例がない。TTY ベースプログラムは移植性が高いため、分散コンピューティングの構成する際の自由度が高くなる。また、遠隔から容易に試験ができ、信頼性向上にも有利である。

第 III 章では、この方法論を使ったプロトタイプについて議論し、ニューラルネットワークの運転監視システムのソフトウェア開発に応用した例について紹介する。ただし、本報の目的は、上記のプログラム統合法の既存の方法

に対する優位性を実証することなので、監視機能など高度情報処理そのものの議論は行わない。最後に第 IV 章でまとめを述べる。

II. ソフトウェア開発の基本的枠組み

1. TTY ベースプログラムによる分散コンピューティング

従来、大規模コンピュータシステムはメインフレームとその端末で形成される形式が多かった。これは計算機が高価でその資源を有効利用するために考え出された方法である。しかし、今日、計算機が安価になり、かつネットワーク環境が飛躍的に向上した結果、機能ごとに計算機を導入する分散コンピューティングの考え方に移行してきている。この方式では、機能ごとに最適なコンピュータ環境が選べるので、上記の集中コンピューティング方式に対してシステム開発が容易である。また、モジュール化されるためシステムの更新や改良が容易になる。分散コンピューティングのもう一つのメリットは、1つのモジュールのクラッシュを他のモジュールが多重にバックアップでき、システム全体がクラッシュするリスクを軽減できることである。よって分散コンピューティングはシステムの信頼性に関してもメリットがある。

分散コンピューティングを実現するためには、各モジュールがネットワークを介して通信できるようになっていなければならない。問題はどのようなモジュール間の結合様式を採用するかであり、この議論は本研究の最も重要な部分である。

さて、本節の見出しに書かれている「TTY ベースプログラム」は、最近の PC やワークステーションでオペレーティングシステム(OS)のプロンプトが出る「コマンドウィンドウ」内でキーボード入力を用いて実行できるプログラムを指している。ファイルを転送するために使用する FTP や遠隔ログインのための telnet はその典型的な例である。すなわち、コマンドを入力し、その応答を見る、そして新たなコマンドを送る、の繰返しによってプログラムが実行されるのが、TTY ベースプログラムである。

本研究では、各モジュールを TTY ベースプログラムとして作成し、それらの複数のモジュールをある“オペレータ”を介して通信させるという方法を採用した⁵⁾。Fig. 1 にその概念を示す。“オペレータ”のタスクはすべての必要なモジュールが存在するホストコンピュータ(以下、簡単にホストと呼ぶことにする)に telnet を用いて遠隔ログインしそのモジュール、すなわち TTY ベースプログラムを起動し、それらの間の通信を行うことである。例えばデータ獲得モジュールに現在のプラント信号を表示させるコマンドを送り、それを読む。そして表示されたデータがあるデータ処理モジュールにコマンドとして入力し、そのデータ処理モジュールからの出力を読む。データ処理モジュールからの診断結果を何らかの方法でオペレータに知ら

Fig. 1 Modules and "Operator"

せる。このようなことを実時間で行うのが“オペレータ”の役目である。このような“オペレータ”としてのタスクを実行するプログラムを、本報では Administration プログラムと呼ぶことにする。

実際の Administration プログラムは、1つのモジュールのコマンドの実行とその結果の読み込みという繰返しによって処理が進行していくことになる。一般に、アプリケーションの機能を補完するために簡単なロジックを記述して連続した処理を実現できるようにしたプログラムをスクリプトと呼ぶ。すなわち Administration プログラムは一種のスクリプトである。実際の Administration プログラムでは、複数のアプリケーションを同時に扱い、かつまた条件分岐や配列変数の格納等の複雑なプログラミングが要求されるため、Unix のシェルスクリプトのような簡便なスクリプト言語を使うことはできないが基本的な考え方は同じである。

実際には、この目的のため我々は Expect⁶⁾と呼ばれるツールを利用する。このツールの詳しい説明は〔付録〕に譲るが、オペレーティングシステム(OS)に付随したスクリプト言語は、決まりきった手続きを簡便にプログラミングするのに適しているのに対して、〔付録〕のプログラム例のように、Expect を使用すればアプリケーションからの様々な応答に対応できる。また、Expect では1つのスクリプトで複数のアプリケーションとやりとりできるため、複数のホストにログインしてその中のアプリケーションを起動し操作できる。さらに、Expect は独立した言語ではなく、拡張ライブラリーとして C/C++ 言語から使用できるため、スクリプトの機能と C/C++ の有する多彩な機能を結合させることができる。

以上のような方法で、Fig. 1 で示した TTY ベースプログラムによる大規模な分散コンピューティングの構築が可能になる。この方法を用いれば、モジュールを開発するのにネットワーク通信のための高度なプログラミングテクニックを使用する必要がなく、ソースコードも理解しやすい。さらに非標準ライブラリーを用いないので移植性が高い、標準入出力ライブラリーを備えた言語なら何でも使えるので言語の選択の幅が増える、オフライン解析用等に開発した既存のプログラムを流用しやすい等のメリットがある。また、スタンドアローンで動作するプログラムなのでモジュールだけで動作確認ができる。さらに、スクリプトを使えばモジュールのソフトウェア試験を容易に自動化でき、信頼性向上に大きなメリットとなる。モジュールのスタンドアローン性はチームで開発作業を行う場合にはより大きな恩恵をもたらす。

2. オブジェクト指向(Object-Oriented : O-O)プログラミング

FORTRAN 等の手続き型(procedural)の言語では、計算機にこちらがやってほしい処理を直接順番に書いていくことによってプログラムを生成する。しかしながら、O-O プログラミング言語では、計算機内にオブジェクトと呼ばれる“部品”をいくつか定義することにより解こうしている問題の環境を整えておいた上で、部品同士の相互作用として処理あるいは機能を生成する。O-O 型言語を導入することによって得られる最も大きな恩恵の一つはプログラムを容易に階層化できることである。そのため、ソースコードは読み易くそのため保守も容易であり、複雑なソフトウェア開発におけるコーディングの信頼性を、すなわち設計仕様と実際のプログラムの同一性を向上させることができる。また、昨今 Unified Modeling Language (UML) と呼ばれる O-O 設計を対象にしたグラフィカルな仕様記述法が発案された⁷⁾。これはクライアントからの要求仕様から詳細な仕様へとアプリケーション開発を展開させていくときにソフトウェア開発者を支援するもので、すでにその有用性が実証され実際に様々な分野で使われている。このような技術の進歩も O-O プログラミングを選択する一つの理由となる。特に分散コンピューティングの場合にはもともと処理が分れているので、“部品”を定義しやすく O-O プログラミングに向いていると考えられ、本研究においてもその適用の可能性を考えた。

さて、本研究では、前述したように、各モジュールは TTY ベースプログラムになっているので、これをなんらかの方法でオブジェクトとして扱えるようにしなくてはならない。このため、Administration プログラムの内部でオブジェクトを作成し遠隔ホスト上のモジュールである TTY ベースプログラムと情報交換をさせる方法を考案した。この枠組みを表したのが Fig. 2 である。このような Administration プログラムのうちのオブジェクトは、仮想

Fig. 2 Virtual modules

的にモジュールの役割を果たすことから、仮想モジュールと呼ぶことにする。

各仮想モジュールの開発は、対応するクラスを Administration プログラム上に定義することで行われる。各クラスはそのインスタンス、すなわちオブジェクトが作成されると同時に、対応する遠隔ホストに telnet を使ってログインし目的のモジュールを起動する。よってこのような手続きはコンストラクタに組み込まれる。さて、Expect では各アプリケーションへの入出力はファイルと同等に扱われる。すなわちモジュール起動時にモジュールの入出力ストリームを表すファイルディスクリプタが得られる。それをオブジェクト内に変数として格納しておく。モジュールとの通信はこのファイルディスクリプタが示す仮想的なファイルへの入出力という形で実行され、高度なプログラミングは必要ない。例えば、通信を行うために send と receive という2つのメンバ関数を定義する。send はモジュールに文字列を送る関数で文字列を引数とし戻り値はない。また、receive は文字列をモジュールから受け取る関数で、引数はなく戻り値は文字列である。前述したように、これら2つの関数はプログラム内変数 fileDescriptor を参照する。

さてオブジェクト外部から send 関数および receive 関数を直接呼び出してモジュールと通信することも可能だが、モジュールに間違ったコマンドを送らないように、モジュールが有している TTY ベースのコマンドを各オブジェクトのメンバ関数として別に定義すべきである。例えば、あるモジュールにおいて“processData”というデー

タ処理を行うコマンドが用意されていたとする。このコマンドの実行はコマンドウィンドウ上では

```
module1> processData 34.5e-3
98.122e-2
```

というやりとりすると仮定する。ただし、下線部はキーボード入力であり、2行目の出力データ処理の結果である。これに対して、Administration プログラムのクラス側ではメンバ関数 processData が必要である。この関数は上記にコマンドに対応するように浮動小数点変数を引数とし、浮動小数点変数を戻り値とする。このメンバ関数は前出の send や receive などの関数を呼び出す。また、文字列と浮動小数点変数の間の型変換、およびその際の例外処理も必要である。これによって変数型のチェックが行える。

さて以上のような仕様をみたとすように C++ のクラスをコーディングすれば、以下ようになる。

```
class module1{
private:
    //ファイルディスクリプタの宣言
    int fd;
    //遠隔ホストのモジュール起動のためのメンバ関数の宣言
    int spawn( );
public:
    //コンストラクタの宣言
    module1( );
    //遠隔モジュールに文字列を送るメンバ関数の宣言
    void send(fd, string);
    //遠隔モジュールから文字列を読むメンバ関数の宣言
    string receive(fd);
    //コマンドを実行するメンバ関数の宣言
    double processData(double);
}
//コンストラクタの定義
module1::module1( ){
    //遠隔ホストのモジュールを起動
    fd = spawn( );
}
```

ここで、module1 はある遠隔ホスト上のモジュールに対応する C++ クラスである。このクラスのインスタンスを作成するとコンストラクタが呼び出され、結果として spawn 関数が呼び出される。spawn 関数ではモジュールとの通信が確立され、通信先を識別するファイルディスクリプタ fd が得られる。上記のコードでは spawn, send および receive の関数定義は省略した。これら3つの関数は Expect の C/C++ ライブラリを使って容易に作成できる。fd は private のメンバなので外部から参照できず、結果として send や receive など呼び出すことができない。このように外部からオブジェクト内の情報を隠蔽することをカプセル化と呼ぶ。これはモジュールをクライアント(この場合は Administration プログラム)の不正な操作から守

るために有効である。クライアントから呼び出せる関数はコンストラクタと processData 関数のみである。すなわち、Administration プログラムはモジュールを起動することと、モジュールが有するコマンドを実行する以外の操作はできず、システムの信頼性が保たれる。

Administration プログラム内で遠隔ホストの processData コマンドを使うには、main 関数(C/C++においてプログラム開始直後、最初に実行される関数)の中でクラス“module1”のインスタンスを作成した後、processData 関数を呼び出す。すなわち、以下のようなソースコードになる。

```
//標準入出力ライブラリのインクルード
#include <stdio.h>

//Expect ライブラリのインクルード
#include "expectlib.h"

//クラス module1 の定義がここに入るが省略する
//main 関数の定義
main() {
    //インスタンスを作成, モジュールと telnet 通信が確立
    module1 m1;
    //入力データの定義
    double input=0.1;
    //出力データの宣言
    double output;
    //データ処理
    output = m1.processData(input);
}
```

モジュールを追加する場合には、上記の module1 クラスを雛形として使い、processData 関数の代わりに、追加するモジュールのコマンドをメンバ関数として定義するだけである。これは O-O プログラミングの常套テクニックであるクラス継承を用いれば容易に実現できる。

以上のように、モジュールが TTY ベースプログラムのままでも O-O プログラミングを使った分散コンピューティングは可能であり、O-O 技術が有する様々な恩恵を受けることができる。

なお、本節で示したソースコードは、遠隔ホストの IP アドレスなど、説明上簡略化した部分があり、実際のプログラミングとは異なっていることに注意されたい。

3. 他の方ととの比較

これまで、本報で提案する分散コンピューティング法について説明してきた。本節においては、考えられる他の方法と比較することにより本方法のメリットおよびデメリットを議論する。

(1) TCP/IP ソケットプログラミングの利用

遠隔ホスト上のモジュールと通信する方法の一つとして、ソケットと呼ばれる TCP/IP を利用するアプリケーションのための Application Programming Interface (API) を

利用する方法が考えられる。現在、最もよく使われているモジュール間の通信手段と考えられ、C/C++でそのような通信を行うことは難しいことではない。しかしながら、このような方法を用いると、クライアント側だけでなくモジュール側にもソケットプログラミングが必要になる。ソケットを使えるプログラム言語は限られており、モジュールの開発言語が制限される。また、たとえソケットが使えたとしても非標準ライブラリーの場合が多く移植性が低くなってしまふ。

それに比べ前述の方法を用いれば、モジュール側でネットワーク通信のプログラミングは不要である。よってすでに開発されたアプリケーションを流用したい場合などに、ソケットプログラミングを用いた場合よりも修正が容易になる。また、通信に使われる telnet はこれまで長い間使われてきたため、バグが潜在する可能性は非常に低く信頼性が高い。なお、遠隔ホストのセキュリティ確保には telnet のユーザー照会をそのまま利用するため特別なプログラミングは不要である。

またソケットを用いたモジュールはスタンドアローンのプログラムではない。よってモジュールのみのソフトウェア試験できないので、例えば、仮のメインプログラムを作成しなくてはならない。このとき仮のメインプログラムの信頼性も問題になるので、ソフトウェア試験が提案された方法に比べ複雑になる。

(2) ファイルによる結合

モジュールと Administration プログラムをファイルを介して通信する方法も考えられる。ファイル入出力機能は標準入出力と同様に、ほとんどのプログラム言語で利用できるため、開発言語の自由度は TTY ベースプログラムを用いた場合に匹敵する。ただし異なるホスト間で通信するためには、モジュールか Administration プログラムの少なくともどちらかが遠隔ホスト上のファイル操作をする必要がある。これには記憶領域の共有やファイル転送などなんらかの TCP/IP を利用したアプリケーションが必要である。また、送り側受け側でファイルアクセスのタイミングを取る必要があり、そのための追加プログラミングも必要である。このように同じ結果を得るために、提案された方法に比べ、アルゴリズムがかなり複雑になると考えられる。

(3) 遠隔オブジェクト間の通信

現在すでに、異なったホストのオブジェクト間通信を行うソフトウェアの枠組みが徐々に整備されつつある。Common Object Request Broker Architecture (CORBA)⁸⁾はその代表的なものであり、デファクトスタンダードになりつつある。CORBA においては利用できる言語は関数型である必要があり、FORTRAN のような手続き型言語は使えない。また、モジュールで外部から呼び出されるインターフェースを定義しなくてはならないので、そのプログラミングが単なる TTY ベースプログラムよりもやや専

門的になる。よってモジュール作成の容易さや柔軟性においては本報で提案された方法に劣っている。しかし、様々な OS および言語が同時に使えるのは大きな魅力である。CORBA はすでにネットショッピングなどビジネス分野などで実用化されつつあるが、まだ開発途上であり、安全性が重視される分野への応用は時期早尚と考え本研究では考慮しなかった。しかし、今後、技術が成熟すれば、高信頼性が要求されるソフトウェアシステムにおいても魅力ある技術になると期待され、本報で提案された方法と共存させてモジュールの種類によって使い分ければシステム構築の自由度を高められる。

III. プロトタイプの開発

前章では、ソフトウェア統合型の運転監視システムの分散コンピューティングの枠組みについて述べてきた、本章では、この枠組みに準拠したプロトタイプを構築してその実現性を確認する。

1. オペレーティング・システム(OS)

本システムは Unix ワークステーションをネットワーク結合させたシステムを対象としている。現在、Unix を基本としている OS は、Solaris, Linux, MacOS 等である。これらのすべての OS で Administration プログラムおよびモジュールをインストールできる。その他の OS については、新たに Expect のライブラリーまたはそれに相当するものを作成する必要がある。また、他の OS 上のモジュール、すなわち TTY ベースプログラムは、そのままでは Unix 上の Administration プログラムに結合できず、少々追加のプログラミングが必要である。これについては III-3 節で実例を使って説明する。

2. ユーザーインターフェース

この機能は運転員が運転監視システムを起動したり、診断結果を監視したり、詳細な情報を要求するために、適当な入力装置と出力装置によって構成される。ユーザーインターフェースは運転監視システムにとって非常に重要な部分であるが、本報の研究対象ではないため、キーボードとマウスの入力および CRT の出力という常識的なハードウェア構成とする。

II-1 節で述べたように、システム全体を起動するのは Administration プログラムである。すなわち、Administration プログラムを実行するホストがシステム全体のエントリポイントになっている。セキュリティ上、他のホストの CRT に勝手に出力することはできないので、ユーザーインターフェースは Administration プログラムと同じホスト上に置くのが自然である。(ちなみに、複数のユーザーが存在する場合には、複数の Administration プログラムを起動すればよい。このようなマルチユーザーに関する詳細については次節に譲ることにする。)実際には、

Administration プログラムとユーザーインターフェースは 1 つのプログラムにまとめた。以下では、ことわりがない限り、「Administration プログラム」という用語はユーザーインターフェースを含んでいるとして使用する。

Unix をインストール環境としたため、ユーザーインターフェースは X-Windows 上で構築した。このため Xlib を使うことになるがこれを直接使ってグラフィック機能のプログラミングを行うのは不便であり、通常はより使い勝手のよい X ツールキットが使われている。このようなツールキットはすでに多く存在しているが、我々はその中から GTK+⁹⁾を選択した。これはなるべく Unix の C/C++ で一般的に使い、多く流通していて、将来にわたって長く保守されると期待されるものを選択した結果である。

3. データ獲得モジュール(daq モジュール)

アナログ信号を実時間でデジタル値に変換し、それをさらに物理値に変換し、Administration プログラムに送信するのがこのモジュールの役目であり、運転監視システムでは必要不可欠なモジュールであるので、特に節を設けてその構築について説明する。

このモジュールは、単にデータ解析を行うモジュールとは違い、ハードウェアに強く依存しているため、Unix を選択できない場合が考えられる。また、制御室では多数のユーザーが Administration プログラムを同時に起動することを念頭に置かねばならないが、daq モジュールを前章で提案した通常のモジュールとして設計すると、プラントデータが 1 ユーザーに占有されてしまい不都合である。我々は上のような制限から、このモジュールに特別なプログラミングが必要であると考えた。

Fig. 3 では我々が Windows 上に構築した daq モジュールと Unix 上の Administration プログラムとの接続を示している。ここで、daq モジュールは基本的には通常のモジュールと同様に、コマンドベースのインタラクティブなプログラムとして設計される。すべてのコマンドについて詳細に述べないが、データの要求などがコマンドベースで操作できるようにする。例えば“get data”と入力すると、下の例のように時刻とすべてのチャンネルのデータを出力するようにしておく。

```
daq> get data
```

```
2002-DEC-12 23:22:10 23.334e-2 84.763e-2 ...
```

Fig. 3 にある“遠隔ターミナル”は Unix 上の TTY ベースプログラムであり、Administration プログラムと daq モジュールを結合させるインターフェースの役割をもっている。具体的には、遠隔ターミナルは Administration プログラムによって起動され、遠隔ホスト上の daq モジュールに対して TCP/IP コネクションを確立する。よって、daq モジュールでは“遠隔ターミナル”との TCP/IP コネクションを確立するため、TCP/IP ソケットを使ってサーバ機能を作成しておく必要がある。コネクションが確

Fig. 3 Connection between the daq-module and the administration program

立されると, daq モジュール側ではコマンドの入出力ストリームが標準入出力ではなく“遠隔ターミナル”との間で確立された TCP/IP のストリームになるようにしておく必要がある。このようなしかけによって, daq モジュールを Unix 上の TTY ベースプログラムと見かけ上全く同じように利用できる。(なお, ソケットプログラミングに関しては文献10)を参考にした。)この手法には一般性があり, 原理的には daq モジュールをどのような OS にも構築できるはずである。daq モジュールは既存の計装制御システムとのインターフェースであり, daq モジュールの適応性が高いことはシステム全体の移植性が高いことを意味する。

また, ソケットを使ったプログラミングでは1つのサーバに対して複数のコネクションを確立することができるため, 1つの daq モジュールを複数の Administration プログラムから利用できる。よって, 複数の運転員がプラントデータ等の情報を共有するとともに, 運転監視システムを別々に実行でき独自の解析を実行できる。

なお, 遠隔ターミナルから daq モジュールへの接続の際 telnet を通さないで, そのままでは遠隔ホスト側にセキュリティの問題が生じてしまう。そこで, 本システムでは daq モジュール独自のユーザー管理プログラムを別途作成し, daq モジュール接続時にユーザー名とパスワードを照合するようにした。なお, その際, ユーザーはデータ収集の定義・開始終了が行える管理者と, データ収集するだけの一般ユーザーの2レベルに分けている。

4. 応用例: ニューラルネットワークによる異常検知システム

(1) 用いられたニューラルネットワーク

ここで用いられたニューラルネットワークは3層の階層型のオートアソシアティブネットワークである。このタイプを選んだ理由は, 計算時間が短く実時間システムへの応用に適しているからである。また, 特にそのニューラルネットワークにおいて入力ユニット数と出力ユニット数を

同じにし, 各ユニット(すなわち各ニューロン)の入出力関数にはシグモイド関数を採用した。入力ユニットには原子力プラントから計測された信号が入力される。すなわち, 入出力のユニット数は異常検知に用いられる信号の数に等しい。出力層から出力される値は入力層に入力された信号の1サンプリング周期(本件では4s)後の予測値である。

異常検知を行うにはまず, プラントが正常な運転状態で計測されたプラント変数の時系列を用いて, ニューラルネットワークが, 上記のように, 正しい予測値を出力するように学習させておく。ここで正常な運転状態として, 定常運転時だけでなく, 起動時やシャットダウン時のデータに対しても学習させておくと, 学習の精度が良くなる。学習アルゴリズムには最も広く用いられているバックプロパゲーションアルゴリズムを採用した。十分学習されたニューラルネットワークに今度は実時間のプラント変数を入力し, その出力は同じ信号の次にサンプリングされた値と比較される。ここですべての変数がニューラルネットワークの予測値とほぼ同じであればプラントは正常とみなされ, そうでない場合には異常とみなされる。これが異常検知の原理である。

ただし, 一般に原子炉は燃焼が進むに従ってプラント特性が徐々に変化する。このため, 運転サイクル初期用に学習されたニューラルネットワークは運転サイクル末期の異常検知に適用できない。なぜなら, プラント特性が変化してしまっているため, 正常でも異常と判断してしまうからである。そのため, 本研究では特に, ニューラルネットワークが正常なプラント特性変化に追従するように適応学習付ニューラルネットワークを適用した。これは異常監視時にも, 計測されたプラント変数が正常と判断できるほど予測値に近ければそのデータを用いてニューラルネットワークを実時間で更新する方法であり, これによりニューラルネットワークは常に最新のプラント特性を考慮したモデルになる。詳しくは文献11)を参照されたい。

(2) 異常検知システムの構築

さて, これを本システムのモジュールにするには, 入力

Fig. 4 Schematic picture of the neural-network-based fault detection system

データ(すなわちプラント変数データ)のセットを標準入力からプログラムに入力したら、出力データ(すなわちプラント変数の予測値データ)のセットの計算結果を標準出力に出力されるようにプログラムしておけばよい。すでに我々は前項のような適応学習型のニューラルネットワークのプログラムを FORTRAN で開発しており、長期にわたって試験してきたため、その信頼性は高いと考えそれを利用することにした。このプログラムはオフラインで異常検知機能を研究するためのもので、プラント変数データの入力先、および予測値の出力先はいずれもファイルである。よってモジュール化の作業は FORTRAN コード数行の変更ですんだ。このモジュールを neuro と呼ぶことにする。このように長年使われてきた、いわゆるレガシーコードを簡単に運転監視のモジュールにできるのは、ここで採用された分散コンピューティングの枠組みのメリットである。

この例において、モジュールは daq と neuro の2つである。それぞれに対して Administration プログラムの中に C++ のクラスを作成した。Fig. 4 にシステム全体を表す図を示す。

また、Fig. 5 にユーザーインターフェースの画面表示を示す。このプロトタイプのはこまで説明してきた手法の成立性を確認するためであり、ユーザーインターフ

Fig. 5 Graphical user interface of the neural-network-based fault detection system

ェースはデータ処理の健全性が確認できる程度のごく簡単なものとどめた。具体的には、画面上部に監視システムの終了と起動のために“stop”ボタンと“start”ボタンを置いた。また、その下のスクロールバーを備えた大きなウィンドウには監視システムが正常に動作していることを検証するための数値データを表示させた。具体的には、1行目は現在の日付と時刻である。2行目以降は列ごとにグループ化されたデータで、左から1列目が観測されたプラントデータセット、そして2列目がそれに対するニューラルネットワークの予測値、そして3列目がそれらの差である。

ハードウェア構成には同じルータ下に置かれた3つの計算機を用いた。Administration プログラムと daq 用の遠隔ターミナルを第1のホストに、neuro を第2のホストに、daq を第3のホストにそれぞれ配置した。図では daq がプラントに接続されているように描かれているが、実際にはまだ実プラントに接続されていないため、実際の原子炉からすでに得られているデータや、プラントシミュレータで作成したデータを、実時間と同じ間隔で読み込ませた。これらを用いて、実時間データ処理の健全性が確かめられた。

IV. まとめ

以上、第I章で述べた高機能と高信頼性を併せ持った運転監視システムのための分散コンピューティングの枠組みを考え、そのプロトタイプを構築した。

TTY ベースプログラムを高度情報処理のモジュールに

採用したことは、モジュールのプログラミングに自由度を与えることができたと考える。第 III 章で紹介したニューラルネットワークのアルゴリズムはもともオフラインデータ解析用に FORTRAN で書いてあったが、数行のソースコードの変更によって本研究で提案した枠組みのモジュールに修正することができた。このように信頼性が確認されている既存のコードを容易にモジュールとして追加することができるのは、この手法の大きなメリットである。今後、計算速度が向上すれば、高度情報処理の手法だけでなくプラント設計データベースや安全解析用の熱水力シミュレーションコードなどをモジュールにして、より詳細な監視診断が可能になると考える。

また、II-2 節で紹介した仮想モジュールの考え方は、遠隔ホスト上にある TTY ベースプログラムをあたかもローカルホスト内のオブジェクトのように扱えるようにするものである。通常、分散コンピューティングにおいてネットワークプログラミングは非常に大きな部分を占めるが、この手法を用いればアプリケーション開発の労力を有意に軽減できると考える。また、II-2 節で示したようなカプセル化を行うことにより、遠隔モジュールを誤って不正に操作する可能性を低くすることができる。

分散コンピューティングの技術は、携帯端末の普及や電子商取引等の新たなビジネスの登場に引っ張られて大きく発展しつつある。このような技術を原子力分野、特に運転制御における高度情報処理に応用していくことが重要である。しかしながら、高信頼性が要求される原子力分野において最先端のソフトウェア技術を盲目的に受け入れるわけにはいかない。本報で示された分散コンピューティングの枠組みがこのジレンマに対する 1 つの答えとなることを期待したい。

— 参考文献 —

- 1) 北村正晴, 兼本 茂, 佐伯 昭, 他, “進化する診断技術—人間機械協調の新しいパラダイムを目指して,” 原子力誌, **40** [9], 652 (1998).
- 2) 若林二郎, 大賀幸治, 永井哲郎他, “原子力発電プラント・セーフティサポートシステムの開発,” 原子力誌, **43** [4], 331 (2001).
- 3) N. Fujitsuka, H. Tanouchi, Y. Imamura, *et al.*, “Experience and evaluation of Advanced On-Line Core Monitoring System “BEACON” at Ikata Site,” *Proc. In-Core Instrumentation and Reactor Core Assessment*, Mito, Japan, 103 (1996).
- 4) T. Anegawa, M. Ishikawa, “Study of BWR stability at Kashiwazaki BWR Unit 4 using the On-Line Stability Monitor SIMON-64,” *Proc. SMORN-VII*, Avignon, France, Vol. 1, 137 (1995).
- 5) T. Suzudo, K. Nabeshima, H. Takizawa, “Software Integration for Monitoring Systems with High Flexibility,” *Prog. Nucl. Energy.*, **43** [1-4], 405 (2003).
- 6) D. Libes, *Exploring Expect—A Tcl-based Toolkit for Automating Interactive Programs*, O'Reilly, (1995).
- 7) G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling*

Language User Guide, Addison Wesley, (1999).

- 8) CORBA ホームページ (<http://www.corba.org>).
- 9) たなかひろゆき, GTK+入門—基礎からはじめる X プログラミング, ソフトバンク, (2002).
- 10) 澤川 渡, 他, TCP/IP 解析とソケットプログラミング, オーム社, (2000).
- 11) 鍋島邦彦, ニューラルネットワークを用いた原子炉異常監視診断技術の研究, JAERI 1342, 日本原子力研究所, (2001).

〔付 録〕

Expect についての紹介

Expect はインタラクティブなプログラムを自動化するために、米国 NIST の Don Libes によって開発されたツールキットである。フリーソフトウェアになっていて、Don Libes が管理するホームページ <http://expect.nist.gov> からダウンロード可能である。Expect は独立した言語ではなくて Tcl (インタープリタ言語の 1 つでティクルと呼ばれる), または C/C++ の拡張ライブラリーとして利用できる。詳細は文献 6) に譲るが、使用方法は極めて単純である。例えば, “telnetServer” というホスト名のコンピュータに “myName” というユーザー名で自動的にログインする場合を考える。Tcl ではそのスクリプトを、以下のように書ける。

```
spawn telnet telnetServer
set timeout 10
expect{
    "connection refused" exit
    "unknown host" exit
    timeout exit
    "username:"
}
```

```
send "myName\r"
```

まず、1 行目で telnet を起動する。2 行目では telnet からの応答を何秒待つか、すなわちタイムアウトまでの秒数が設定される。この場合は 10 s に設定されている。その下の { } の中は telnet 側からどのような出力が返されるかに依存した条件分岐になっている。

最初の条件は “connection refused” が返された場合で、この場合はこのスクリプトを終了する。また同様に “unknown host” が返された場合にもこのスクリプトを終了する。3 番目の条件はタイムアウトの場合で、10 s たっても応答がない場合にスクリプトを終了させる。4 番目の条件は正常にユーザー名の入力を要求された場合で、この場合スクリプトは継続され最後の行で、ユーザー名が telnet に送られる。実際の場合にはこの後、パスワードの要求および入力、遠隔ホスト上のアプリケーションの起動および操作そしてログアウトというように処理が進行していくことになる。

説明を簡単にするため、上記の例では Tcl を使ったが、この言語は原子力プラントなどの信頼性の高いソフト

ウェアが要求される分野のプログラミングには向いておらず、そのような応用に対してはライセンスも取得できない。本文でも述べたように、実際のプログラミングにおいてはC/C++を使用している。なおコーディングの際、文

献6)の中のC/C++からの呼出しについての説明と、Expectのダウンロード時に添付されてくるサンプルプログラムを参考にした。