視覚表現を活用した大規模制御システムの一設計法

正員 名井 健 (東京電力) 非会員 葛岡 英明 (東京大)
非会員 甘利 治雄 (東京電力) 非会員 廣瀬 通孝 (東京大)
非会員 稲村 浩平 (東京大) 非会員 石井 威望 (東京大)
非会員 小池 英樹 (東京大) 正員 林 達郎 (東京電力)

A Method of Large-Scale Control System Design Aided by System Visualization Technology

Takeshi Myoi, Member, Haruo Amari, Non-member (Tokyo Electric Power Company),
Kouhei Inamura, Non-member, Hideki Koike, Non-member, Hideaki Kuzuoka, Non-member,
Michitaka Hirose, Non-member, Takemochi Ishii, Non-member (The University of Tokyo),
Tatuo Hayashi (Tokyo Electric Power Company)

Progress in computer and communication technology has made it possible to construct a large scale control system. However, expansion of a control system will make it more difficult to design control software. In designing a large scale control system, it is hard to understand the relationship between computers such as flow and synchronization of tasks and messages. To solve this problem, we are developing a system to visualize control software in order to access information regarding the performance of total control system accurately and easily.

This paper describes characteristics of large scale control systems, the method of its visual representation, configuration and the role of the visualization tool, and efficiency of the visual design method.

キーワード：大規模制御システム，システムの全体設計，視覚表現，視覚化ツール

1. はじめに

近年，計算機技術とそれをネットワークする通信技術は急速に発達してきている。これにより，多数台の計算機を連係した大規模な制御システムの構築が可能となっている。電力系統制御システム⁽¹⁾，FA システム，交通管制システムなど，その具体例は非常に多く，社会基盤として重要な役割を果たしている。

今後，計算機技術・通信技術のより一層の発展と共に，制御システムの規模，機能も更に拡大すると考えられる。

システム構築を行う場合，一般的には要求仕様記述，ソフトウェアプログラム設計，テスト，デバッグという段階が踏まれる。大規模分散制御システムを設

計する場合，その各段階で計算機どうしあるいはタスクどうしの協調動作など，システム全体の挙動の記述，検証が必要となる。本論文では，システム全体の挙動の記述，検証にかかわる設計を全体設計と呼ぶ。これに対して，個々の計算機あるいはタスクのプログラム設計，作成を個別設計と呼ぶこととする。個別設計に関しては，個々のプログラムのみに集中すればよく，従来の開発環境でも対応可能である。これに対して全体設計では，計算機，タスクそれぞれの機能分担，イベントに対するメッセージの流れの決定，メッセージの同期処理・優先順位処理など，個別設計にはない困難さがある。現在，全体設計の重要性の認識が高まっているが，決め手となる手法は開発されていない。その一方，各種制御システムの規模の拡大，機能

の拡張に伴いその重要性が更にクローズアップされると予測される。

本論文の主目的は、今後重要性が増す大規模制御システムの全体設計を、正確かつ容易に実施できる手法を提言することである。この目的を達成する方法として、著者らはシステムを構成するソフトウェアの全体像を視覚表現する手法を採用した。ソフトウェアを視覚表現することにより、全体の挙動の表現が容易となり、全体設計を正確かつ効率的に実施できるようになる。この基本的な考え方のもと著者らは、

- (i) 大規模制御システムのソフトウェアの視覚表現手法の設定〔文献(2)で著者らは基本概念を提示した。本論文はその応用に力点を置く〕
- (ii) 視覚表現による設計を支援するための視覚設計支援ツールの試作
- (iii) 視覚表現と視覚設計ツールの具体的な活用方法とその有効性の考察

を行った。

以下、その内容について報告する。

2. 大規模制御システムの特徴

大規模制御システムとして、電力系統制御システムを具体例として取上げ、その特徴を分析する。電力系統制御システムはその設計において、時間軸を考慮した動的な分析が必要となる。つまり、計算機間・タスク間のメッセージの時間的な流れ、あるいはメッセージどうしの同期処理などのリアルタイム動作を記述する必要がある。これは、電力系統制御システムに限らず、大規模制御システム一般に共通する特徴である。

大規模制御システムの一例として、電力系統の総合自動化システム⁽³⁾を考えてみる。このシステムでは、変電所機器などの状態(変化)の監視、運転員による平常時あるいは緊急時の機器操作、定期的な記録・統計あるいは制御システムそのものの性能チェックなどの処理が実行される。これらの処理は一つの計算機中の一つのタスクのみで行われるのではなく、複数の計算機システム、サブシステム、タスクが相互に(メッセージを介して)連携することにより実現される。

例えば、送電線事故が発生した場合、監視サブシステム、情報送受信サブシステム、マン・マシンインタフェースサブシステム、操作サブシステム、更にサブシステムを構成する複数のタスク間をメッセージが連鎖的に流れる。この様子を図1に示した(サブシステム、タスクの構成は、設計の行い方で変化する。ただし、上述のサブシステムは設計指針によらず必要となるものである)。本論文では、あるイベントに対する

図1 総合制御所の処理の一例
Fig. 1. Typical activity of control center.

一連のメッセージの流れを「サクセッション」と呼ぶこととする⁽²⁾。

大規模制御システムでは、その機能・処理内容の多様性から様々なサクセッションが存在し、それぞれが異なった優先順位をもつ。前述のように優先順位の高いサクセッションのほか、平常時の操作、比較的優先順位の低い記録・統計処理など、優先順位レベルの数は非常に多くなる。

大規模制御システムを実際に運用していく場合、各種処理がふくそうすることも多く、様々なサクセッションが同時に存在し、複雑な優先順位処理が要求される場合も多い。更にそれらの処理はリアルタイムで実行される。数多くのサクセッション、複雑な優先順位処理、リアルタイム処理、これらが大規模制御システムの特徴である。このため、システム構築を行う際は、時間軸を含めた動的挙動を正確に表現しなければならない。

3. 制御システムの視覚表現

本章では、制御システムをどのように視覚表現するかを考える。前章までに、システムを設計する際は時間軸を含めた動的挙動を記述する必要があることを明らかにした。ここでは、前章で示した総合自動化システムの処理例をもとに、システムの視覚表現要素を抽出する。

必要となる視覚表現要素は以下のとおりである。

- (1) サブシステム・タスクといった機能の階層性の表現

図 2 制御システムの三次元表現
Fig. 2. 3D representation of control system.

- (2) サブシステム、タスクの関連性の表現
 - (3) イベントに対するメッセージの流れの表現
(サクセッションの表現)
 - (4) メッセージの同期、分岐の表現
- これらを同時に表現したのが図2である。

図2において、X軸方向からながめると、サブシステム-タスクなどの機能の階層性、およびサブシステム、タスクの関連性がわかる。一方、Y軸あるいはZ軸方向からながめると、時間の経過に伴うメッセージの流れとメッセージの同期、分岐のタイミングが把握できる⁽²⁾。

従来の手法でも、図形表現を活用した設計が行われている。例えば、サブシステム-タスクの階層性あるいは関連性を示すものとして、タスク間関連図があり、メッセージの流れや同期・分岐を表現するものとしてタイムチャートがある。しかしながら、これらは二次元表現が主体であり、タスク間関連図とタイムチャートを同時に表現することが困難である。このため設計者は、システム全体の挙動を考えると、頭の中でこれらに関連づけなくてはならず、実時間分散システムの設計を非常に難しいものとしている。

これに対して、図2のような三次元の表現方法を活用すれば、上述した視覚表現要素をすべて同時に表現でき、システムの全体設計を効率的に支援することができる。更に、以上の視覚表現基本要素以外にも視覚表現規定を定めておくことが望まれる。具体的には、

- (i) サクセッションの相互関係 (例えば優先順位処理)
- (ii) メッセージの不定回数繰返し、条件分岐、

待ループ

- (iii) タスクのファイルアクセスの有無の表示ルールである。

以上の視覚表現要素の具体的な実現方法と活用方法は、第5章で述べる。

4. 視覚化ツール

大規模制御システムの全体設計を行う際、通常、タスク間関連図とタイムチャートが利用される。これらはともに二次元平面上に表現することができ、従来の机上設計でも十分に対処することができる。

これに対して、前章の図2で示した三次元表現を紙上で実際に記述するのは困難となる。仮に記述したとしても、自由に視点 (タスク間関連図に着目する、タイムチャートに着目する、あるいは斜めから見て同時に把握するなど) を変えることによってシステムの全体像を把握するといったことができない。このため、従来の設計環境では、三次元表現のもつ特長を生かすことはできない。

そこで、著者らは三次元表現を容易に作成し、それを自在に編集できるようにする視覚化ツールを試作した⁽⁴⁾。その構成を図3に示す。

視覚化ツールは、基本的には三次元表現を視覚化するための三次元グラフィックワークステーション (以下、GWS と呼ぶ) と、三次元入力装置であるデータグローブ⁽⁵⁾ (以下 DG と呼ぶ) から構成されている。

更に、三次元表現の利点を十分に活用するため、GWS を立体視処理している。立体視システムには専用のコントローラを採用し、1秒間に60フレームの

図 3 視覚化ツールの構成
Fig. 3. Configuration of visualization tool.

立体映像を得ており、立体視によるちらつきはほとんどない。

三次元表現は、前記の GWS を中心とした視覚化ツール上に 3D グラフィックにより実現される。タスク、メッセージなどを表現するグラフィックデータはファイルとして登録されており、視覚化ツールの利用開始時にそのファイルを読み込み、初期グラフィックを作成する。タスク、メッセージの追加あるいは初期設計を行う場合は、新規作成のメニューにより新たなグラフィックを作成する。その結果はファイルに登録され、必要に応じて再利用される。

次に DG の活用方法について述べる。DG と GWS はシリアル回線によって接続され、DG からの情報が GWS に送信され処理される。DG から得られる情報は、(i) 空間的位置と回転角、(ii) 5 本の指の曲がり角、の二つである。前者は、立体グラフィックを扱うための必須の情報である。後者の指の曲がり角を利用すれば、手のジェスチャーをグラフィック操作のコマンドとして活用することができる（例えば、グラフィックの一部を移動させたいときは、その部分で手を握り、手を目的位置まで移動させるなど）。これにより、人間が自然界で行うような自然な動作感覚で立体グラフィックを扱うことができる。視覚化ツールの活用例を図 4 に示す。

以上の視覚化ツールを活用すれば、第 3 章で紹介した制御システムの全体設計に活用する三次元視覚表現

図 4 視覚化ツールの活用例
Fig. 4. Operation of visualization tool.

を、自在に作成・編集することが可能となる。

5. 視覚化ツールの活用

制御システムの三次元視覚表現と、それを実際の設計へ適用するための手段である視覚化ツールの活用方法を述べる。活用方法を具体的に述べる前に、もう一度、全体設計の役割を整理する。第 1 章でも記したように、要求仕様記述からテスト、デバッグまでの各段階で、全体設計は必要となる。具体的には、以下の点が必要となる。

- (i) 機能の分担、メッセージの流れ、同期処理、優先順位処理が的確に設定、記述できること。
- (ii) 作成中のシステムの全体的な駆動状況を容易かつ正確に認識できること。
- (iii) 全体設計で記述した情報が個別設計に反映できること。
- (iv) 作成したソースコードを全体構成という点から視覚化し、ソースコードの良否（特に、全体的な協調動作にかかわる部分の良否）を判断できること、更にシステム全体の動作結果の正当性をチェックできること。

この点を考慮して、

- (1) 制御システムの全体像の視覚表現の記述と編集
 - (2) 実行のシミュレート
 - (3) テンプレート(後述)の自動作成
 - (4) ソースコードと実行トレースの視覚化
- を視覚化ツールの具体的な活用方法として設定した。
- (1) 項により、時間軸を含めたシステムの全体像を明確に表現できるので (i) を、(2) 項により (ii) を、(3) 項により (iii) を、(4) 項により (iv) を満たすことができる。

以下、(1)～(4) 項それぞれについて詳細に述べる。

- (1) 視覚表現の記述と編集 GWS 上でサクセッションを記述することが基本となる。作成の手

順は、

- (i) タスクの配置
- (ii) メッセージの作成

である。タスク間をメッセージで結ぶことにより、サクセッションを作成する。タスクは通常、メッセージを受け起動状態となり、メッセージを送信して待状態となるが、起動状態を明確にするためその部分を直方体とする(図5)。更にメッセージ名称を記述してサクセッションが完成する。

監視、操作、マン・マシンなどのサブシステム(図1)は複数のタスクによって構成されている。システムの全体像をつかむうえで、サブシステム-タスクといった階層を表現する必要がある。これを実現するため、サブシステムを半透明表示し、包含するタスクを明確にした(図6)。更にサブシステムを通常に表示し、包含されているタスクを不可視とすることにより、システムの全体像を明確かつ単純に表示することができる。

以上により、視覚表現されるシステムの全体像を視点を変えて見ることにより、タスク間関連図に着目した表示(図7)、あるいはタイムチャートに着目した表示(図8)を自在に行うことができる。

ここまでが、制御システムの視覚表現の基本要素の実現と活用方法である。更に第3章で示した基本要素

以外の視覚表現を考えなくてはならない。まず第一に、サクセッションどうしの相互関係の記述が重要となる。

大規模制御システムでは、複数のサクセッションが同時に存在する。そのため、サクセッションどうしの干渉を記述する必要がある。具体的パターンとして、発生順序の決まっているサクセッション(シーケンス制御手順作成サクセッションとシーケンス制御実行サクセッションなど)、あるいは他のサクセッションに関係なく、任意の時刻に発生する優先順位の高いサクセッション(機器の状態変化処理サクセッションなど)がある。前者に対してはサクセッションの前後関係を動かさないようにし、後者に関してはサクセッションを動かしたとき影響を受ける(優先順位の低い)サクセッションを色変させる、といった機能を付加することで対処している。

更に、メッセージの繰返し、待ループは、それらを表すグラフィック記号を用意し、条件分岐については、分岐先のグラフィックの輝度を分岐数に応じて下げていくことで表現するようにした。また、ファイルアクセスの有無は、タスク(グラフィック表現された柱)をDGで触れたときに、アクセスのある場合のみ色変させることで認識できるようにした。

制御システムを設計するうえで、更に別の視覚表現

図5 サクセッションの作成
Fig. 5. Creation of succession.

図7 タスク間関連図
Fig. 7. Task diagram.

図6 タスクの階層表示
Fig. 6. Representantion of task hierarchy.

図8 タイムチャート
Fig. 8. Time chart.

規定が必要になることもある。その際は適宜、視覚表現規定を設定し、視覚化ツール上で実現すればよい。視覚化ツールは、前章で示したように高度に電子化されており、視覚表現規定の追加に対応することができる。

以上のシステム視覚表現作成、編集機能は、従来の紙上設計では実施することはできず、前章で紹介した視覚化ツールを活用してはじめて可能になる。

(2) 実行のシミュレート 視覚表現の記述と編集により、サクセッションが把握しやすくなり、システムの全体像がとらえやすくなる。ただし、メッセージの分岐などがあると、起動タスクの時間の前後関係がわかりにくくなる。そこで、サクセッションを時間の経過と共に色を変えて表示することによって実行の様子をアニメーションとしてシミュレートするようにしている。これによって、分岐などのある複雑な形態のサクセッションであっても、メッセージや起動タスクの時間の前後関係を明確にとらえられるようになる。

また、システム設計の途中の段階で、システム全体の協調動作を確認する必要性がたびたび生じる（いわゆるウォークスルー）。このシミュレート機能を利用すれば、設計段階でシステム全体の挙動の正当性を確認かつ容易に判断することができる。

(3) テンプレートの作成 本視覚表現と視覚化ツールは、制御システムの全体設計を支援するものであり、設計者はこれを利用して、サクセッションの記述とサクセッションどうしの相互関係などを記述する。この時点では、設計者は個々のタスクのプログラミング（個別設計）を意識しているわけではない。しかしながら、全体設計と個別設計の協調をとるには、サクセッションレベルの図形情報が下位のプログラムに反映されるべきである。このため、図形情報をもとに、個々のタスクのプログラミングの枠組み（以下、テンプレートと呼ぶ）を自動作成する機能を付加した。その一例を図9に示す。

タスク TASK2 は、mes2 というメッセージを受けて起動され、タスク TASK3 を起動すべくメッセージ mes3 を送信している。このサクセッションを視覚化ツールにより記述すると、図9に示すテンプレートが自動作成される〔メッセージ送受信のシステムコール（この例では activate と receive）は、制御用計算機の OS のシステムコールによって異なる。ただし、テンプレートの記述内容を変化させるのは容易なことであり、大きな課題とはならない〕。作成されたテンプレートには、全体設計で記述されたシステム全体の動作に関する情報が包含されており、この時点

図9 テンプレートの自動作成
Fig.9. Automatic creation of template.

で、設計者は個々のテンプレートの内容の記述のみに集中すればよいようになる。つまり、テンプレート作成機能は、全体設計と個別設計の橋渡し役として有効に作用する。

更に、テンプレートの内容を最低限記述し、「設計の初期段階でシステム全体を動作させてみる」といったラピッドプロトタイピングの手段としても活用することができる。

(4) ソースコードと実行トレースの視覚化

作成したソースコードを視覚化すれば、その良否の判断がしやすくなる。特に、ソースコードのうち、計算機間あるいはタスク間の連係に関係する部分に注目し視覚化すれば、ソースコードの全体構成のチェックが行いやすくなる。大規模な分散実時間システムの設計の正確化に、ソースコード視覚化は有効に作用すると考えられる。

具体的な視覚化の実施方法は以下のとおりである。まず、ソースコードのうちタスク間通信に関係するシステムコールを抽出し、タスク間のつながりを求め、それを視覚化し、全体構成を表現する。

更に、実行結果の履歴（トレース）を制御用計算機の OS から獲得し、その結果を上記タスク間関連にオーバーラップさせることにより、実行結果を動的に表現することを試みた。具体的には、実行結果のうち、タスクスイッチングとメッセージフローに関する情報を選出し、時間経過と共にタスク間関連図の関連部分を色変えて、タスク、メッセージの協調的な稼働結果を視覚的に表現した。

以上の実施方法を図10に、実施結果（ソースコードの視覚化）を図11に示す。

ソースコードの視覚表現手法は図2に示した三次元

図 10 ソースコードの視覚化手法
Fig.10. Visualization method of source-code.

図 11 ソースコードの視覚化結果
Fig.11. Visualization of source-code.

表現手法に準拠させた。このソースコード視覚化により、ソースコードあるいは仕様書などの文書を読むより、容易かつ確実にソフトウェアの全体像のチェックを行うことができる。更に実行トレースの動的視覚化により、実行状況の把握、実行性能のチェック、システムの協調動作に関するテストやデバッグを効率的に行うことが可能となる。

6. 有効性の検証

本論文で提言した大規模制御システムの視覚表現手法と視覚化ツールの有効性を確認するために、変電機器制御システムのミニモデルを作成し（5サブシステム、41タスク、約10kStep）、各種の定性的評価を行った。サブシステム作成にあたっては前章までに記した視覚表現手法と視覚化ツールの機能を活用した。その結果、

- (i) サブシステム、タスクの広がり、サクセッション錯綜度の認識
- (ii) メッセージの集中箇所の把握、同期処理の複雑性、システム構成の特徴（マン・マシンサブシステムにメッセージが集中するシステムであ

るなど）の把握

- (iii) シミュレート機能によるシステムの稼働状態の予測
- (iv) 優先順位関係など、サクセッションの相互関係の把握
- (v) テンプレート作成機能を利用した全体設計から個別設計への展開
- (vi) ソースコードの視覚化によるソフトウェアの全体構成のチェック

に大きく寄与した。更に、システムの各種情報をわかりやすく視覚表現することにより、システム設計が良好に実施された（例えば、メッセージの過剰の集中を避けることができた）。

視覚表現を活用したソフトウェア作成の研究は数多くなされておられ、著者らも三次元視覚表現を活用したソフトウェア作成の有効性について、基礎的な定量評価を行ってきた。そこでは、視覚表現と視覚化ツールを活用すると最高70%ソフト作成時間が削減することを報告した⁽²⁾。しかしながら、大規模システムの開発を本格的に支援するための、ソフトウェア視覚化の研究はほとんど実施されていない。

今回の視覚表現と視覚化ツールは、本格的な大規模分散システムの作成支援を目指したところに特長がある。今回の評価の材料とした変電制御システムのミニモデルは、ソフトウェアのステップ数は小さいが、サブシステム、タスクの広がり、複数の優先順位をもったサクセッションなど、大規模分散実時間システムの要素はかなりの部分網羅されており、検証モデルとして妥当であると考えられる。

7 まとめと将来展望

本論文では、大規模制御システムの全体設計を支援する手法を提言し、その有効性を定性的に明らかにした。しかしながら、実際の設計に活用するには課題も多い。特に、要求仕様の分析、ソフトウェア再利用の促進については、その手法、ツールとも今回の提言だけでは不十分である。このため今後は、視覚化ツールと要求仕様作成管理ツール、ソフトウェア再利用のためのデータベースとの関係を図る予定である。

計算機技術の発達とソフトウェア視覚化は密接な関係をもってきた。1台の計算機内のプログラムの構成を把握するものとして、フローチャートが開発された。マルチタスク処理、計算機の連係が可能となると、タスク間関連図、タイムチャートを活用するようになった。そして現在、計算機の大規模ネットワーク化、機能分散が進められ、更にはニューロの適用も可

能となっている。これら計算機技術の発達は大規模かつ高機能の制御システムの開発を促進するものである。その一方で、ソフトウェア作成には多くの課題を投げかけることになった。サクセッションの優先順位処理、計算機・タスクの機能分担の設定などシステム全体の協調動作の記述の複雑化である。著者らはこれらに対応した新たなソフトウェア視覚化手法と視覚化ツールを提言した。

本論文の作成にあたり、御指導をいただいた東京電力システム研究所 埼玉主任研究員に感謝の意を表します。

(平成2年6月27日受付, 同3年2月7日再受付)

文 献

- (1) 「特集：電力分野における自動化技術の最近の動向」, 電学誌 (昭63-9)
- (2) 名井・甘利, 他: 「制御用言語の図形表現に関する研究」, 第4回ヒューマンインターフェースシンポジウム論文集, p. 45 (昭63)
- (3) 橋本・林: 「設備総合自動化システムの開発について」, 昭62 電気学会全大, p. 1576
- (4) M. Hirose & H. Amari: "A Study on Visualization of Control Software Design", MIT-JSME Workshop on Cooperative Product Development
- (5) 日経エレクトロニクス, No. 11, p. 106 (昭63)

名 井 健 (正員)

昭和60年3月慶応義塾大学大学院工学研究科修士課程修了。同年4月東京電力(株)入社。62年7月同社システム研究所にて、制御用ソフトウェアの開発手法の研究に従事。

甘 利 治 雄 (非会員)

昭和63年3月東京大学工学部卒業。同年4月東京電力(株)入社。平成元年4月同社システム研究所にて、制御用ソフトウェアの開発手法の研究に従事。

稲 村 浩 平 (非会員)

平成2年3月東京大学大学院工学系研究科修士課程修了。同年4月キヤノン(株)入社。在学中は主として、三次元グラフィック技術の研究に従事。

小 池 英 樹 (非会員)

昭和63年3月東京大学大学院工学系研究科修士課程修了。現在、同博士課程在学中。主として、大規模ソフトウェアの視覚化技術の研究に従事。情報処理学会, AI学会会員。

葛 岡 英 明 (非会員)

昭和63年3月東京大学大学院工学系研究科修士課程修了。現在、同博士課程在学中。主として、ネットワーク上の共同作業の研究に従事。情報処理学会会員。

広 瀬 通 孝 (非会員)

昭和57年東京大学大学院工学系研究科産業機械工学専門課程博士課程修了。同年同大学工学部講師, 58年同助教授, 現在に至る。工学博士。システム工学, ヒューマン・インタフェースに関する研究に従事。

石 井 威 望 (非会員)

昭和29年東京大学医学部医学科卒業。32年同大学工学部機械工学科卒業。38年同大学大学院博士課程修了。同年同大学工学部専任講師, 39年同助教授, 48年同教授, 現在に至る。工学博士。システム工学, 医用工学の研究に従事。

林 達 郎 (正員)

昭和36年3月慶応義塾大学工学部卒業。同年4月東京電力(株)入社。現在、技術開発本部システム研究所主席研究員。電力系統・設備自動化システムの開発に従事。