マルチメディア通信と分散処理 78－8
（1996．9．27）

分散監視制御システム向けフレームワーク HolyGhost

野里 貴仁　　小島 泰三

三菱電機株式会社 産業システム研究所

筆者らはオブジェクト指向技術を用いて、分散型監視制御システム構築環境の設計・実装を行なっている。多種多様な監視制御システムに適用することとソフトウェアの再利用性を高めることを目的として、監視制御システム用のアプリケーションフレームワーク、HolyGhost を開発した。HolyGhost と GUI 構築用ライブラリである GhostHouse と共に用いることで、複雑な GUI を持つ監視制御ソフトウェアを容易に、かつ再利用性高く作成することが可能になる。本論文では、監視制御対象をモデル化するために用いたミラーワールド方式と呼ぶ手法と、分散監環境で通信量を低減するための機構について説明する。

HolyGhost: An Application Framework for Distributed Supervised Control System

Takahito Nozato and Taizo Kojima

Industrial Electronics & System Laboratory, Mitsubishi Electric Corporation
8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo, 661, JAPAN

We design and develop distributed supervised control systems with Object Oriented technology. For the purpose of applying various supervised control systems and raising reusability of the system software, we are developing an application framework called HolyGhost. Using HolyGhost and GhostHouse which is class library for building GUI, we can create the system software which has complex GUI easily, and make the software high-reusability. In this paper, we describe Mirror-World method for modeling the systems and the mechanism for reducing the traffic in the system.

1 はじめに

筆者らは、電力系統や公共プラントなどの監視制御システムのソフトウェア開発支援を目的として、オブジェクト指向フレームワークを開発している。既に筆者らの研究グループでは、C++ 言語によるクラスライブラリを開発し、実際の監視制御システムに適用してきた。このクラスライブラリを GhostHouse [1] と呼ぶ。

GhostHouse は優れたグラフィカルユーザインタフェース (GUI) 作成能力を有し、帳票ビルダやプラント監視画面ビルダの開発に大きな効果をあげてきた。しかし、これまでの適用分野は監視制御システムの中でもオフラインデータを扱うシステムばかりあった。これは、GUI の部品化やビルダの作成のために高度な機能を提供しているが、監視制御実行時のオンラインデータの処理等については十分な機能は提供されていなかったためである。

一方、オンラインデータを扱うソフトウェアでも複雑な GUI を扱っているので、GhostHouse を適用することで、その開発コスト低減することが期待できる。しかし、オンラインデータを扱うためには、個々のシステムごとにデータを扱うソフトウェアを作成する必要があるので、監視制御システムソフトウェアの開発効率を向上させるには不十分であった。

筆者らは、この点を改良し監視制御システムの開発効率を上げるために、分散監視制御システム向けのアプリケーションフレームワークを新たに開発した。以下、次節では、監視制御システムソフトウェアについて説明し、続いて筆者らが開発したアプリケーションフレームワークである HolyGhost の説明を行なう。

2 監視制御システム

2.1 監視制御システムソフトウェア

監視制御システムのハードウェアの構成例を図 1に示す。システム全体は、ネットワークを介した分散環境となっている。その中では、設備の状態を示すデータが定期的にデータゲートウェイを介して、LAN に流され、それらが一度制御計算機で処理されて、支援計算機や監視用 EWS にデータが渡されている。

監視制御システムソフトウェアは、対話処理ソフトウェアと制御ソフトウェアに分けることができる。対話処理ソフトウェアとは、監視画面を通して監視者にプラントの状態を伝え、逆に監視者が画面を操作

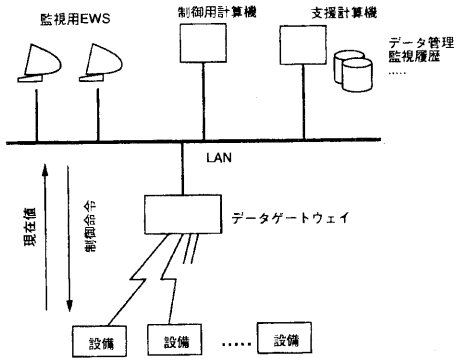

図 1: 監視制御システム概略

することでプラントの操作ができるようにするものである。このために、対話処理ソフトウェアは必要に応じて、プラントからのデータだけでなく、データベースなどとも情報のやりとりをする。対話処理ソフトウェアは図 1の監視用 EWS 上で動いている。制御ソフトウェアとは、プラントのデータを収集して、データベースに格納したり、対話ソフトウェアにデータを送る、また、対話ソフトウェアからの要求に応じて、実際にプラントを制御する、などの動作をするものである。制御ソフトウェアは図 1の制御用計算機上で動いている。

近年の監視制御システムでは、対話処理ソフトウェアの開発コストが監視制御システム全体を開発するコストの大きな部分を占めるようになってきている。これは、高度な対話処理が要求されるようになったからである。例えば、プラントデータを単なる数値で表示だけでなく、メータで表示するあるいは、トレンドグラフで表示するということが必要になっている。データを入力する際にも、キーボードから入力するだけでなく、マウスなどのポインティングデバイスを用い、メニューから値を選択するなどの対話的な入力手段を用意する必要がある。

2.2 GhostHouse

GhostHouse は、先に挙げた GUI ソフトウェアの開発コスト増大に対処するために、開発された C++ 言語によるクラスライブラリである。その狙いは、GUI を持つ対話処理ソフトウェアの開発自動化にあり、システムで利用するデータ定義から暫定的な GUI をも

つ対話処理ソフトウェアを自動生成し、その後で、画面上で修正をしながら最終的なソフトウェアを構築する。

GhostHouse では、モデル部品とビュー部品の 2 種類のオブジェクトを使って GUI を構築する。モデル部品はデータの管理を行なうオブジェクトであり、ビュー部品は描画とユーザからの操作を取り扱うオブジェクトである。モデル部品とビュー部品はリンクと呼ばれるオブジェクトを使って結合することができる。データ値の変更はモデル部品を介して行なわれる。そして、その副作用として結合しているビュー部品にデータ値の変更を通知する。通知を受けたビュー部品は、表示していた描画内容を変更する。ユーザの画面操作はビュー部品を介して行なわれる。この場合は、ビュー部品からモデル部品に通知され、モデル部品はデータ値の変更や関数の呼び出しを実行する。

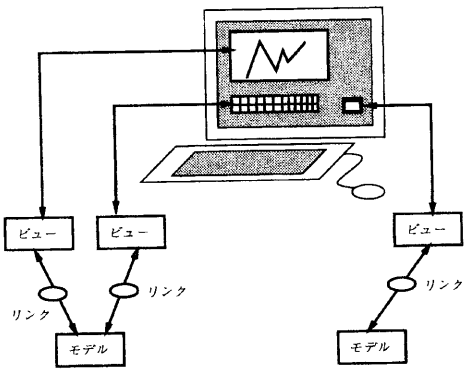

図 2: モデルとビュー

GhostHouse を使って監視制御システムの対話処理ソフトウェアを作成するためには、モデル部品が制御ソフトウェアとインタラクションを行なって、データ収集や制御命令の発行を行なわなければならない。このような実行をサポートするソフトウェア部品は既に存在しているが、それらはアプリケーションに大きく依存しており、GhostHouse と組み合わせるのは容易ではない。このために、GhostHouse を用いたアプリケーションは帳票ビルダなどのオフラインデータを扱うシステムに限られていた。

監視制御システムの開発効率をより高めるには、GUI ソフトウェアのサポートだけでなく、対話処理ソフトウェアと制御システムのインタフェースもサポートする必要がある。すなわち、制御ソフトウェアと対話処理プログラムのデータ処理に対して、統一的な枠組を提供することである。HolyGhost は、このような目的をもって開発された C++ 言語によるクラスライブラリである。

3 HolyGhost

筆者らは、HolyGhost を監視制御システム向けのアプリケーションフレームワークとして開発している。監視制御システムのソフトウェア開発コストを低減するためには、GhostHouse を補完するクラスライブラリでなく、より再利用性の高いフレームワークを用意する必要があるからである。我々が対象とする多くのシステムが分散環境で構築されているので、HolyGhost はクライアントサーバ方式に基づく分散システムを支援する機能も備えている。監視制御システムでは、実時間性が必要であるので、クライアントとサーバの間のプロセス間通信はすべて非同期に処理している。

3.1 アプリケーションフレームワーク

フレームワークとは、抽象クラスや具体的なクラスの集まりとそれらの間でのインタフェースであり、同時にサブシステムの設計そのものでもある [2]。すなわち、サブシステムを構成する上位クラスからなるクラスライブラリで、その中の各クラスのインタフェースはしっかりと決められているものである。フレームワークの有名な例として Smalltalk の MVC [3] モデルがある。フレームワークはサブシステムの抽象的な設計であるので、フレームワークを使った開発では、設計の再利用が可能となる。

あるアプリケーション分野に特定されて開発されたフレームワークをアプリケーションフレームワークと呼ぶ。アプリケーションフレームワークの例としては、金融アプリケーションを対象とした SwapManager[4] や鉄鋼プラントのアプリケーションを対象とした ProcessTalk [5] などがある。

アプリケーションフレームワークを用いると、適用先は限定されるが、個々の応用プログラムの開発では、より高度なソフトウェアの再利用を期待できる。このような特徴から、アプリケーションフレームワークは、類似したプログラムを多数開発する監視制御システムのソフトウェア開発に適している。

3.2 モデルサーバ

HolyGhost を使った監視制御システムは、サーバクライアント方式で構成されており、対話処理ソフトウェアは、モデルサーバと呼ばれるサーバを経由してデータ取得や制御操作を行う。モデルサーバは対話処理ソフトウェアと制御ソフトウェアの間に位置し、データを仲介する役割を持っている。

GhostHouse を用いた対話処理ソフトウェアでは、ビューとモデルがリンクで接続されていた(図 2)。一方 HolyGhost では、ビューはモデルエージェントと呼ばれるオブジェクトと接続されている。さらに、モデルサーバ内のモデルオブジェクトと呼ばれるオブジェクトとプロセス間通信を行なうことで接続されている(図 3)。モデルオブジェクトは、監視制御対象の状態(センサからの入力をコントローラや制御装置が変換した結果)を保持しており、監視制御対象を表現するオブジェクトである。モデルエージェントは対話処理ソフトウェアの中でモデルオブジェクトの代理として機能するオブジェクトである。ビューからみると、モデルエージェントは、モデルサーバと交信してデータを得る(あるいは逆に操作要求を送る)特別なモデルである。

GhostHouse においてモデル部品とビュー部品の関連が定義されていたように、HolyGhost でも、モデルエージェントとモデルオブジェクトのプロトコルはあらかじめ定義されている。すなわち、データ監視を始める場合、機器の制御要求を送る場合などのプロトコルは上位クラスの中で定義されており、システム開発時にプログラマがそれらを意識する必要はない。

制御ソフトウェアと対話処理ソフトウェアの間にモデルサーバを置くことには、次の利点がある。

- 対話処理ソフトウェアの再利用性向上
- システム内通信量の低減

制御ソフトウェアは対象となるハードウェアに強く依存しており、対話処理ソフトウェアにおいても、そのデータ処理部などはハードウェアに依存してしまう傾向があった。HolyGhost では、ハードウェア依存部をモデルサーバが吸収することで、対話処理ソフトウェアが特定の制御システムに依存することを避けている。このために、一度作成した対話処理ソフトウェアを他のシステムに利用することが容易になると考えられる。通信量の低減については 4節で説明する。

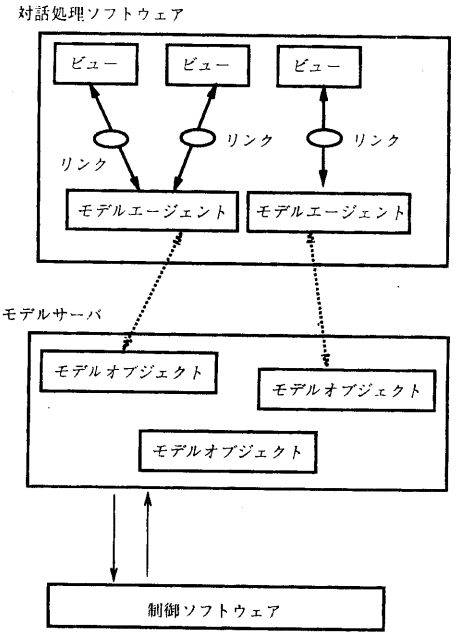

図 3: モデルサーバ

3.3 モデル化方式

監視制御システムには、開閉器やポンプなどの設備、コントローラのような制御用機器、データベースサーバのようなソフトウェアシステムなどの多数の構成要素が含まれている。そして、それらの間にはデータのやりとりを中心とした依存関係が存在している。フレームワークではこれらの構成要素をどのようにオブジェクトとして捉えるかの方針を持たなければならない。HolyGhost では、ミラーワールド方式と呼ぶ手法を導入した。ミラーワールド方式は、対話処理ソフトウェアからの視点を中心にしたモデル化を行ない、エンドユーザに対して理解の容易なデータ表現を提供することを目的としている。

図 4にミラーワールド方式の全体図を示す。ミラーワールド方式の特長は、現実の世界を反転させる形でモデルサーバにおいてオブジェクト化していることにある。例えば、図 4において、設備の現在状態は、設備 → コントローラ → 実時間データ収集サーバと伝播し、モデルサーバに到達する。そして、モデルサーバ内では、外界とは逆順で、それぞれに対応するモデル間をデータが伝播する。一方、対話処理ソフトウェ

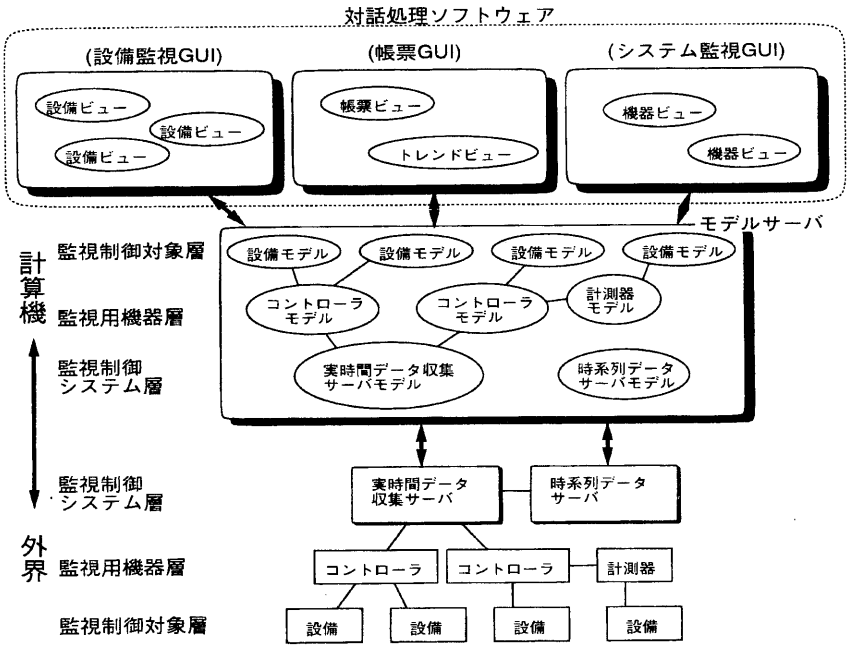

図 4: ミラーワールド方式

アからの制御要求については、現在状態の場合とは逆の方向で命令が伝播する。対話処理ソフトウェアにおいては、対象とする実設備に対応するオブジェクトのみに注目すれば良く、対象システム毎に異なる伝送系システムやコントローラの構成を隠蔽できる。エンドユーザに対し、監視対象である設備をモデル化したオブジェクトを示すことにより、理解が容易な GUI を構築することが可能となる。さらにまた、システム構成の変更においても、ソフトウェアモジュールの入れ換えにより対処できるので、個々のソフトウェアの再利用性が高まるという効果もある。

3.4 クラス構成

HolyGhost の上位クラスを図 5に示す。最上位クラスは HolyGhost である。HolyGhost のサブクラスには、GModelObject と GTask がある。GModelObject はモデルオブジェクトを表す上位クラスである。GTask はメッセージ処理の単位となるオブジェクトの上位クラスである。GTaskManager が Task をスケジューリングすることで、メッセージの非同期処理を実現している。GTask のサブクラスである GMessageAgent はプロセス間通信を実現するための上位クラスであり、GTimeoutTask は定期的な処理や例外処理を行なうための上位クラスである。GhostHouse にはモデルエージェントを表す上位クラスである ModelAgent を新しいサブクラスとして追加した。

4 通信量制御

我々が対象としているシステムは図 1の構成の分散システムである。一般に分散システムでは、システム内の通信量を低くすることが大きな性能向上につながる。監視制御システムのような実時間性を問われるシステムではなおさらである。このために、HolyGhost では、上位クラスの中でメッセージフロー制御などの通信量を低減させる枠組を実装している。

今、定期更新しているプログラムにおいて監視データの表示が短期間停止した場合を考えてみる。データが受け取られないので、モデルサーバの中には既に最新ではなくなったデータがキューに貯められていく。したがって、送信相手が何らかのエラーなどで読み出しを行なわなければ、キューが溢れてしまう。また、

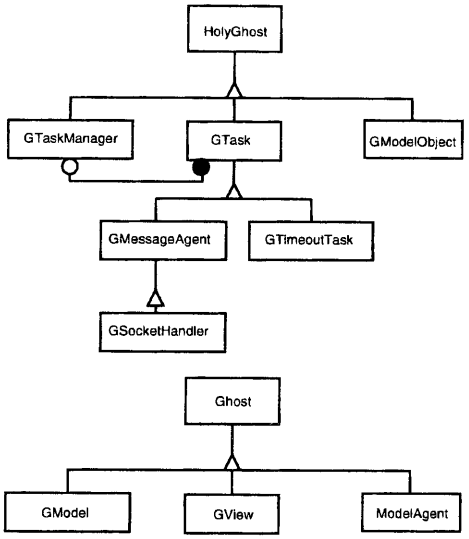

図 5: クラス構成図

キューが溢れる前にデータ表示を再開したとしても、最新ではないデータ (多くの場合不要なデータである) を送り出すことで、更新表示に遅れが生じる可能性がある。

このような場合に対処するために、データを 2 種類に分類している。一つは、全てのデータを通知しなければならないもの (時系列の変化や履歴の表示に使う) であり、もう一つは、最新データのみが重要であるもの (画面表示に使う) である。モデルサーバからのメッセージには、どちらの種類のデータを扱うかを示す印を付けておくメッセージ送信時には、キュー内のメッセージの総量がある閾値以上であり、かつ、このメッセージが最新データのみを要求しているのなら、キュー内の対応する古いメッセージを削除し、新しいメッセージを追加する。

また、非同期にメッセージの処理をするために、画面切替時に前画面で参照していたデータが対話処理ソフトウェア側に届くことがしばしばある。この場合にメッセージを送出しないようにするために、画面を構成するために必要なデータ要求の固まりをグループ化している。このグループのことをセッションと呼んでいる。このセッションは制御ソフトウェアに通信パケットを送る時にも、制御ソフトウェアからデータを受信した時に、通信パケットを調べ、該当するセッションがなければ、そのデータを不用と判断する。

HolyGhost におけるフロー制御の仕組みは簡単であるが、実際のシステムに対して、効果的に作用している。

5 まとめ

分散監視制御システム向けフレームワーク HolyGhost について説明した。HolyGhost は、監視制御のためのデータ処理に重点を当てたクラスライブラリである。HolyGhost ではクライアント・サーバ方式の分散システムの構築をサポートしており、GUI 構築用クラスライブラリ GhostHouse と共に用いることで、複雑な GUI を持った、オンライン系の監視制御ソフトウェアを容易に作成することができる。

現在 HolyGhost を適用して、トンネル監視システム、下水処理監視システムなど開発が行なわれている。

参考文献

[1] 杉本明 ,北村操代. “生成・カスタマイズ方式による GUI 構築手段の提案とクラスライブラリ GhostHouse による実現”, 情報処理学会論文誌, 36(4):944 – 958, Apr. 1995.

[2] Ralph E. Johnson and Rebecca J. Wirfs-Brock. “Surveying Current Research in Object-Oriented Design”, *Communication of ACM*, 33(9):104–124, September 1990.

[3] A. Goldberg. “Information Models, Views and Controllers”, *Dr. Dobb's Journal*, pages 54–61,106,107, Jul. 1990.

[4] Erich Gamma and Thomas Eggenschwiler. “Et++ SwapsManager:Using Object Technology in the Financial Engineering Domain”, In *OOPSLA '92*, pages 166–177, 1992.

[5] R.Weinrich, K. Pirklbauer and R. Plosch. “Object-Oriented Process Control Software”, *Journal of Object-Oriented Programming*, 7(2):30–35. May 1994.