分散制御システムの並列離散事象シミュレーションのための オブジェクト指向デザインパターンに関する研究*

戸村豊明** 金井 理***

Object-oriented Design Patterns for Parallel Discrete Event Simulation of Distributed Control Systems

Toyoaki TOMURA and Satoshi KANAI

Distributed control systems (DCS) consist of many sensors/actuators and a network interconnecting them, and are being introduced in various automation areas. For assuring the control performance of a DCS under heavy communication traffic, the precise simulation of the DCS is strongly needed. For this purpose, we propose a uniform, efficient and systematic method based on object-oriented design patterns for modeling and simulating DCSs. In this paper, two design patterns are newly proposed; Time-Warp pattern and Protocol pattern. Time-Warp pattern describes classes and interaction for executing the DCS simulation by communicating events having send/receive times, using Time Warp mechanism. Protocol pattern describes classes and interaction for uniformly structuring various communication protocol models used in DCSs, which are composed of the interactions among several layers and the state transition of each layer in a communication protocol. Finally, the effectiveness of the DCS simulator which is developed using these patterns was proved by comparing simulation results with the experiment results using the real DCS consisting of four CAN-based control nodes.

Key words: distributed control system, simulation, object-oriented, design pattern, controller area network

1. 緒 言

近年，集中管理方式の制御システムに代わり，通信ネットワークを介してセンサやアクチュエータ等の計装機器同士が直接ディジタル通信する分散制御システム（Distributed Control System, DCS）が，FA，BA（ビルオートメーション），車載 LAN 等の分野へ急速に普及している [1)2)].

一般的な DCS は，図 1 のようにセンサ・アクチュエータ等の「デバイスコンポーネント」，センサ信号の収集・アクチュエータの制御・ネットワーク経由の通信を行う「制御ノード」，1 台の制御ノードと複数のデバイスコンポーネントからなる「デバイス」，制御ノード間を接続するネットワークから構成される．

DCS は物理的通信路が 1 本のため，制御ノード間の通信遅れやデータ損失が生じやすくなる．通常この現象の有無を調べるのはシステムの設置後であり，原因究明と対策に多くの時間・費用が必要とされる．

そのため，システムの設置前に DCS の挙動を予測し，機能検証が行える DCS シミュレータが，システム設計者から強く望まれている．このシミュレータ開発を効率的に進めるには，1) DCS を構成するさまざまなデバイスの構造を表現できる DCS モデルを効率的に記述できる事，2) デバイス内部の挙動とデバイス間の通信挙動を 1) のモデルと関連付けて記述できる事，3) これらのモデルを特定のプログラミング言語上へ容易に実装できる事，という 3 条件を満たすシステムモデリング手法が必要となる．

オブジェクト指向モデリング手法は，DCS のような複雑な構造・挙動を持つシステムのモデリングに適した一手法である．本研究は，デザインパターン [3)] と呼ばれる再利用可能なオブジェクト指向ソフトウェア設計技法を用いて，上記の要求条件を満たすために，DCS の静的構造と動的挙動を高精度にモデリングし，このモデルをプログラミング言語上へ体系的に効率良く実装する手法の提案を目的としている．前々報 [4)] では，上記 1), 3) を満たすパターンとして，デバイス・コンポジットデバイスモデルの構造を規定する Device-Constructor パターン・Composite-Device-Constructor パターンをそれぞれ提案した．また前報 [5)] では，上記 2), 3) を満たすパターンとして，階層型有限状態機械として記述される制御ノード・デバイスコンポーネントモデルの挙動を規定する Statechart パターン，デバイスモデル内とデバイス間のイベント通信経路を規定する Event-Chain パターンを提案した．さらに前報では，これらのパターンを LonWorks [6)] を通信ネットワークとした DCS のシミュレータ開発に適用し，その通信特性を高精度に予測できる事を確認してきた．

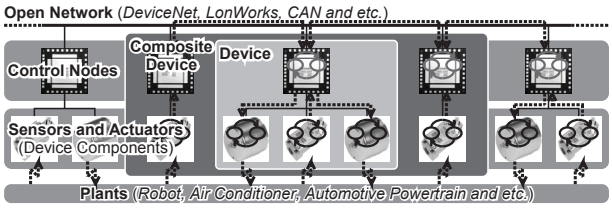

Fig.1 Typical Structure of DCS

実際には，DCS では応用分野別に，多様な通信プロトコルを持つネットワークが使われている．LonWorks 以外にも自動車内制御用の CAN [7)]，FA 制御用の ControlNet [8)] 等の規格が利用されている．しかし，これらの DCS は通信プロトコルの差こそあれ，基本的構造は図 1 に類似しており，かつ，構成機器の挙動は有限状態機械として記述できるものも多いため，モデリングとシミュレーションの共通化を図れる可能性がある．

しかし，前報までに報告した LonWorks 専用の DCS シミュレータの機能を，さらにこれら他規格の通信ネットワークにも対応できるように汎用性を高めるためには，以下の問題点を解決する必要がある．

* 原稿受付 平成 19 年 6 月 12 日
** 正 会 員 旭川工業高等専門学校 (北海道旭川市春光台 2 条 2 丁目)
*** 正 会 員 北海道大学大学院情報科学研究科 (北海道札幌市北区北 14 条西 9 丁目)

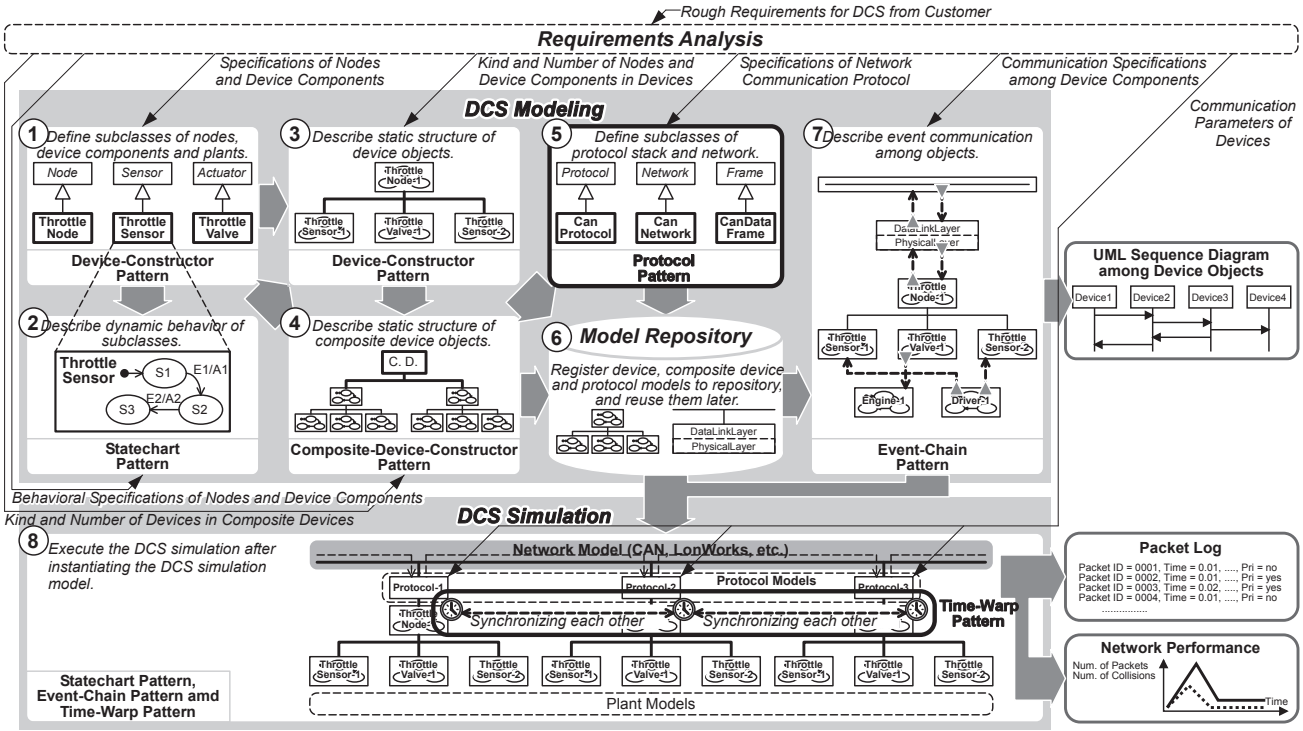

Fig.2 DCS modeling and simulation methodology using design patterns

(1) DCS のシミュレーションは, 基本的にネットワークの形態によらず，ネットワークに接続されたモデル構成要素単位の間でイベント通信を行いながら，これらの要素が時間的に同期を取り，状態を変化させてゆくシミュレーションモデルを持つ．このシミュレーションモデルは，ネットワーク形態によらず DCS 全体に共通であるにもかかわらず，前報までの開発方法では，プロトコルが変わると，シミュレーション機構も各プロトコルに合わせて新たに実装しなければならなかった．

(2) LonWorks 以外の通信プロトコルのシミュレーションを実現するためには，そのプロトコルで規定される再送信サービスやバス調停といったメカニズムや，ネットワーク上を伝送されるフレームの構造を新たにモデリングし，シミュレータへ新たに実装する必要がある．

これらの問題点を解決し, DCS のネットワーク通信プロトコル形式から独立した，汎用的な DCS シミュレータ開発を実現するための方法論の提案を行う事が，本研究の目的である．この目的実現のため，具体的に以下の事項を本論文で述べる．

(1) DCS モデルにおける制御ノード・センサ・アクチュエータ・制御対象モデルが，送信・受信時刻を持つイベントを通信する事によって論理上並列に動作し，かつ大域的に同期するシミュレーション機構を，ネットワーク通信プロトコルによらず統一的に規定するデザインパターンとして，Time-Warp パターンを新たに提案する．

(2) DCS におけるネットワークの通信プロトコルモデルの一般的な静的構造（プロトコルレイヤとそれらの間のデータフロー）および動的挙動（各レイヤにおける送信サービスやパケット作成機能等）を規定するためのデザインパターンとして, Protocol パターンを新たに提案する．

(3) (2) のパターンから，通信プロトコルモデルの実行可能 Java コードへ変換するための，システマチックな実装手続きを提案する．

(4) 提案するパターンと実装手続きを DCS シミュレータへ組み込んで，車載 LAN を想定した，CAN をネットワークとする DCS 用のシミュレータを開発し，シミュレーション結果と実機による通信実験結果とを比較する事により，上記 (1), (2), (3) の有効性を確認する．

2. デザインパターンに基づく DCS シミュレーション手法

本研究で提案するデザインパターンを利用した DCS モデラと DCS シミュレータの構造を図 2 に示す．ここで，本論文で提案する 2 つのデザインパターンの目的を以下に示す．

(a) Time-Warp パターン：Statechart として記述された動的挙動を持つモデル（制御ノード・センサ・アクチュエータ・制御対象等のモデル）間で，送信・受信時刻を持つイベントを通信し，それぞれ固有の時計の時刻を進める事により，その結果大域的に同期するシミュレーション機構を規定する．

(b) Protocol パターン：デバイスモデルにおける通信プロトコルモデルとネットワークの構造と挙動を規定する．

前々報[4] では, DCS モデルの静的構造を規定するデザインパターンとして，Device-Constructor パターン，Composite-Device-Constructor パターンを既に提案している．これらは図 2 のモデリングプロセス①，③，④において利用される．また前報[5] では，DCS モデルの動的挙動を規定するデザインパターンとして，Statechart パターン，Event-Chain パターンを既に提案している．これらは図 2 のモデリングプロセス②，⑦，⑧において利用される．一方，本報で新たに提案する Time-Warp パターン，Protocol パターンは，図 2 のモデリングプロセス⑤，シミュレーション実行時⑧において以下のように利用される．

(1) モデリング対象となる通信プロトコルの仕様書を参照し，Protocol パターンを構成する各クラスの具象クラスを CASE ツール上で新たに定義し，これらの具象クラスとそのメソッドを Java 言語で実装する．これにより，

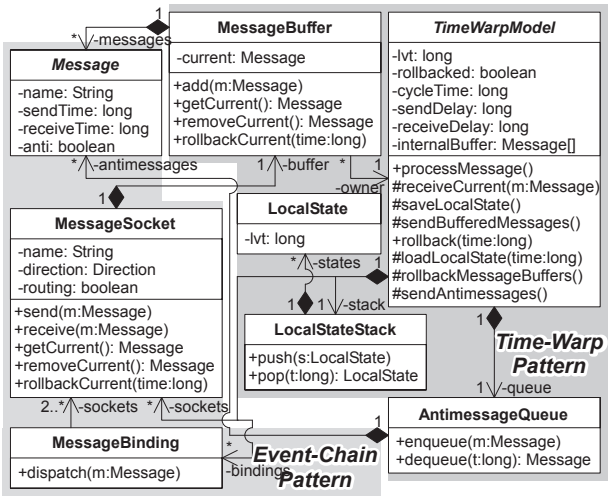

Fig.3 UML class diagram of Time-Warp pattern

ある特定のネットワーク規格に対応する通信プロトコルモデルとネットワークの動作シミュレーションモデルが定義される（図2 ⑤）.

(2) 各デバイスモデルに対し，(1) で定義・実装された通信プロトコルモデルのインスタンスを1つずつ生成・接続し，これらのインスタンスを1つのネットワークモデルへ接続する. 次に, 各インスタンスの通信パラメータ（例えば, 通信速度やメッセージ ID など）を設定し, DCS シミュレーションを開始する. その結果, パケットのログ, パケットの数，パケットの衝突回数等が出力され，DCS 設計者は，DCS の制御性能を評価できる（図2 ⑧）.

3. Time-Warp パターン

3.1 並列離散事象シミュレーション機構に対する要求条件

まず，本研究で求められている DCS シミュレーションを論理的に並列に実行するためのデザインパターンについての要求条件を明らかにする.

並列離散事象シミュレーション（Parallel Discrete Event Simulation, PDES）とは，離散的に生起するイベントによって状態遷移するシミュレーションモデルを複数のサブモデルへ分割して，それらを並列計算機の処理要素へ割り当てる事により，効率的にシミュレーションを実行する仕組みである [9]. PDES では，サブモデルごとに固有の仮想時刻（シミュレーション時刻）を管理する分散時刻管理機構を用いるのが一般的である. シミュレーションを進行させるには，これらの仮想時刻を大域的に同期させる必要があり，そのために保守的または楽観的な時刻同期手法を実装する必要がある.

各サブモデルでイベント間の発生時刻の順序関係に誤りが生じない事を確認しながら実行する保守的手法として，デッドロックを回避できる Null Message 法 [10]，同期法 [11] 等が提案されているが，モデリング対象によっては，処理要素におけるイベントの待ち行列でボトルネックが発生してしまい，サブモデルの仮想時刻を効率的に前進できないという問題点がある.

一方，サブモデルごとに局所仮想時刻（Local Virtual Time, LVT）を非同期に進めて，あるサブモデルにおいてイベント間の順序関係に誤りが生じたら，その仮想時刻と局所状態を巻き戻す事によって大域的に同期させる楽観的手法として，Time Warp 法 [12) 13)] が提案されている. Time Warp 法は，巻き戻し処理が雪崩的に発生しなければ保守的手法よりも効率的に PDES を実行できるという利点を持つ. さらに，Time Warp 法における大域的な同期処理は，各サブモデルにおけるイベント処理の呼出と不要なイベントと局所状態の調査・破棄（化石回収）のみで良いため，並列計算機から PC まで各種コンピュータ上へシミュレーション機構を容易に実装できるという利点を持つ.

この Time Warp 法を DCS シミュレーションへ適用するためには, 以下の条件を満たす PDES 実行用のパターンが必要となる.

(1) 制御ノード・センサ・アクチュエータ・制御対象モデルがイベント処理により状態遷移する時，各々が LVT を前進させた後，巻き戻しに備えて各々が現在の局所状態を保存できる事.

(2) (1) のモデルにおいて，到着したイベント間の順序関係に誤りが生じた場合，過去に送信したメッセージの効果を取り消し，かつ LVT と局所状態をその前時刻まで巻き戻せる事.

上述した条件を満たすデザインパターンとして，本研究では Time-Warp パターンを新たに提案する.

3.2 Time-Warp パターンの構造

Time-Warp パターンの UML クラス図を図3に示す. ここで, 図3における各クラスの機能を以下に示す.

- **TimeWarpModel**：外部からイベントを受信すると，LVT の時刻を前進させ，別のイベントを外部へ送信するモデルの抽象クラス．本研究では，これの具象クラスとして，Statechart として記述された挙動を持つ制御ノード・センサ・アクチュエータ等のモデルを表すクラスを定義する.
- **LocalState**：TimeWarpModel オブジェクトの局所状態を表す具象クラス．このクラスのオブジェクトの生成時点での LVT を属性として持つ.
- **AntimessageQueue**：過去に送信したイベントを取り消すためのイベントを，巻き戻しに備えて仮想送信時刻の順に格納する待ち行列の具象クラス.
- **LocalStateStack**：外部からのイベントを処理する時刻での, 制御ノード・センサ・アクチュエータなどの局所状態を, LVT の順に格納するスタックの具象クラス.
- **MessageBuffer**：TimeWarpModel オブジェクトに到着したイベントを仮想受信時刻順に格納するバッファの具象クラス.

Time-Warp パターンを構成しているこれらのクラスが持つメソッドを以下のように呼び出す事で，3.1 節で述べた条件を満たす並列離散事象シミュレーションを実行できるようになる.

(1) TimeWarpModel オブジェクトは, 到着したイベントを受信する時，自身の LVT をそのイベントの仮想受信時刻へ更新した後, LocalState オブジェクトを1つ生成して, それを LocalStateStack オブジェクトへプッシュする.

(2) MessageBuffer オブジェクトにおいて到着したイベント間の順序関係に誤りが生じると, TimeWarpModel オブジェクトは，誤り発生の原因となったイベントの仮想受信時刻よりも以前に保存された LocalState オブジェクトをポップし，そのオブジェクトが保持する LVT へ自身の LVT を更新する. その後，更新後の LVT 以降に送信したイベントの効果を取り消すイベントを AntimessageQueue オブジェクトから取り出して外部へ送信する.

3.3 Time-Warp パターンによる並列離散事象シミュレーション

TimeWarpModel クラスの processMessage() メソッドを呼び出すと, 並列離散事象モデルは下記のイベント処理を実行する.

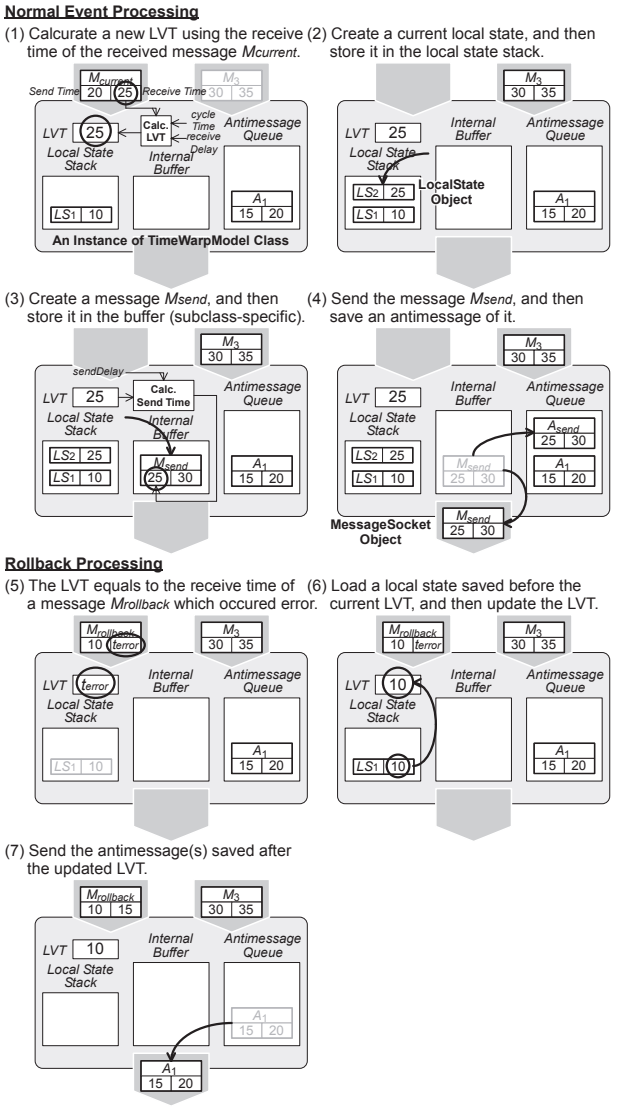

Fig.4 Event processing and rollback processing in Time-Warp pattern

(1) はじめに，rollbacked 属性が真ならば巻き戻し処理を実行するために (7) へ進み，一方，偽ならば (2) へ進む．

(2) 所有する全 MessageSocket オブジェクトから，最小仮想受信時刻を持つメッセージ $M_{current}$ を得，これをパラメータとし receiveCurrent()メソッドを呼び出し，(3) へ進む．

(3) 図 4 (1) のように，自身の cycleTime 属性，receiveDelay 属性，$M_{current}$ の receiveTime 属性を基に $M_{current}$ 受信後の LVT を計算・更新し，自身の saveLocalState() メソッドを呼び出して，(4) へ進む．

(4) 図 4 (2) のように，LocalState オブジェクトを 1 つ作り，これをパラメータとして LocalStateStack オブジェクトの push() メソッドを呼び出す事により局所状態を保存した後，自身の sendBufferedMessages() メソッドを呼び出して，(5) へ進む．

(5) TimeWarpModel クラスのサブクラスは，図 4 (3) のように，自身の LVT と sendDelay 属性を基に仮想送信時刻を計算し，その時刻を持つ具体的メッセージ M_{send} を内部バッファへ格納してから，(6) へ進む．

(6) 図 4 (4) のように，M_{send} を MessageSocket オブジェクトへ送信すると共に，M_{send} のアンチメッセージ A_{send} をパラメータとして AntimessegeQueue オブジェクトの enqueue() メソッドを呼び出し，A_{send} を保存し終了する．

(7) 図 4 (5) のように，自身の LVT は誤りを発生させたイベントの仮想受信時刻 t_{error} に等しい．巻き戻し処理を開始するために，(8) へ進む．

(8) 図 4 (6) のように，t_{error} をパラメータとして自身の loadLocalState()メソッドを呼び出す事により，t_{error} 以前の LVT と局所状態へ巻き戻した後，(9) へ進む．

(9) 図 4 (7) のように，自身の sendAntimessages()メソッドを呼び出す事により，巻き戻し後の LVT 以降に送信された全イベントに対応するアンチメッセージを送信する．

以上により，制御ノード・センサ・アクチュエータなどからなる DCS の Time Warp 法を用いた PDES の実行が可能となる．

4. Protocol パターン

4.1 通信プロトコルモデリングに対する要求条件

本節では，通信プロトコルモデルの記述において，新たに提案すべきデザインパターンに対する要求条件を明らかにする．

通信プロトコルは，一般的に複数の階層（プロトコルレイヤ）から構成される．例えば，FA・BA で普及している LonWorks[6] は，OSI 参照モデルの 7 階層全てを規定している，一方，車載 LAN 分野で普及している CAN（Controller Area Network）[7] では，データリンク層と物理層のみを規定している．

従来の非商用ネットワークシミュレータ [14) 15) 16)] は, 通信プロトコルを TCP/IP またはその関連プロトコルへ限定するものが多く, その場合, これら以外のプロトコルを利用するためには, C/C++言語等によるハードコーディングが必要となる．一方，一部の商用ネットワークシミュレータ [17) 18)] は, モジュール化された多種の通信プロトコルを利用できるものの, DCS の制御ノードから下の階層のモデル（例えば，センサ・アクチュエータのモデルなど）の挙動記述に関して機能的に不充分である．

そこで本研究では，複数のプロトコルレイヤから構成される制御ノード内の通信プロトコルの挙動と，ネットワーク上でのフレーム通信の挙動を汎用的に記述し，模擬できるデザインパターンが必要となる．

4.2 Protocol パターンの構造

4.1 節で述べた要求条件を満たすデザインパターンとして，本研究では Protocol パターンを新たに提案する．このパターンの UML クラス図を図 5 に示す．ここで，図 5 における各クラスの機能を以下に記す．

- **Protocol** : 1 つのコントローラと 1 つのトランシーバからなる通信プロトコルの抽象クラス．
- **NetworkController** ： 複数のプロトコルレイヤから構成されるコントローラの抽象クラス．
- **NetworkTransceiver** ： ネットワークへフレームを送信するトランシーバの抽象クラス．
- **ProtocolLayer** ： ネットワークへフレームを送信または受信するプロトコルレイヤの抽象クラス．
- **Network** ： トランシーバから送信されたフレームを他のトランシーバへ伝送するネットワークの抽象クラス．
- **NetworkFrame** ： トランシーバから送信されるフレームの抽象クラス．Message クラスのサブクラスである．

この Protocol パターンは, 以下に示すクラス間の相互作用を実現するメカニズムを適用する事により，4.1 節で述べた要求条件を満たす事ができる．

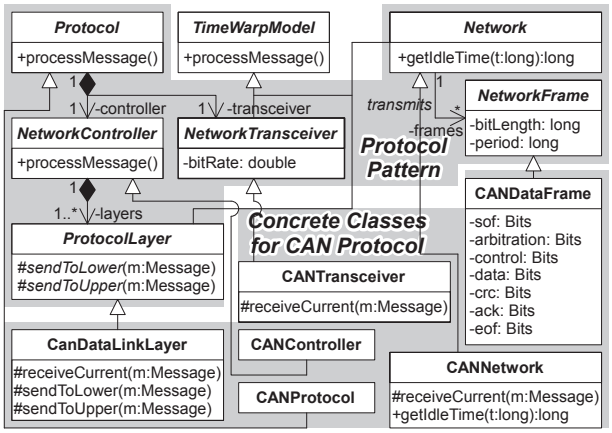

Fig.5 UML class diagram of Protocol pattern

(a) ProtocolLayer オブジェクトは MessageSocket オブジェクトを介して，他の ProtocolLayer オブジェクトや Device オブジェクトとのイベント通信が可能である．

(b) 4.3 節で述べる実装手続きを用いて ProtocolLayer クラスの具象クラスを定義し，そのメソッド内へ通信プロトコルで規定されている機能を実装するだけで，特定の通信プロトコル挙動を精密にシミュレーションできる．

4.3 Protocol パターンを用いた通信プロトコルモデル実装手続き

CAN 通信プロトコルを具体例として, Protocol パターンを用いて通信プロトコルモデルを実装する手続きについて述べる．この実装手続きは DCS 設計者によって下記のように行われる．

(1) 通信プロトコル仕様におけるデータ通信用フレームの構造を参照する事により，NetworkFrame の具象クラス（CANDataFrame）を定義する．この具象クラスの属性はフレームを構成する各種のビット列である．

(2) 通信プロトコル仕様におけるプロトコルレイヤの構造を参照する事により，ProtocolLayer の具象クラス（CAN DataLinkLayer）を定義する．この具象クラスでは，受信した Message オブジェクトの種類に応じて，別種の Message オブジェクトを送信する方向を決定するために receiveCurrent() メソッドをオーバーライドする．さらに，上位または下位のレイヤへ別種の Message オブジェクトを実際に送信する sendToLower() メソッドと sendToUpper() メソッドもオーバーライドする．

(3) NetworkController の具象クラス（CANController）を定義し，そのコンストラクタにおいて，上記 (2) で定義した具象クラスのオブジェクトの生成手続きを実装する．

(4) NetworkTransceiver の具象クラス（CANTransceiver）を定義し，上記 (1) で定義した具象クラスのオブジェクトを受信し，そのクローンをネットワーク方向へ送信するために receiveCurrent() メソッドをオーバーライドする．

(5) Protocol の processMessage() メソッドは NetworkController・NetworkTransceiver の順に processMessage() メソッドを呼び出すので，CANProtocol でのメソッドのオーバーライドは不要である．

(6) NetworkController の具象クラスとして CANController を定義する．NetworkController の processMessage() メソッドは ProtocolLayer の processMessage() メソッドを呼び出すので，CANController でのメソッドのオーバーライドは不要である．

(7) NetworkFrame の具象クラスとして CANDataFrame を定義する．

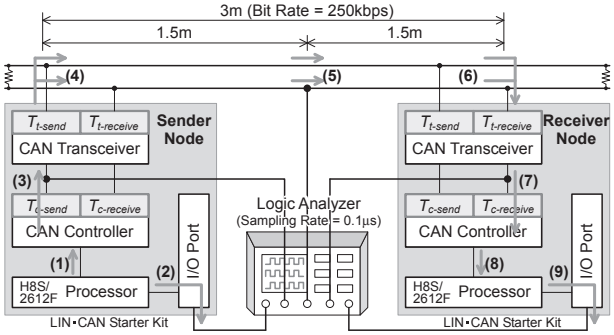

Fig.6 The measurement equipment of send and receive delays of CAN node

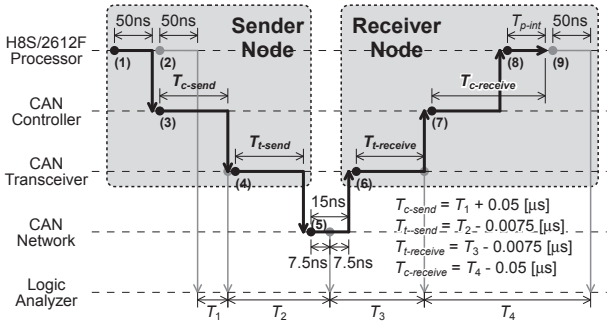

Fig.7 Communication timing between CAN nodes in Fig.5

5. 通信実験とシミュレーションの結果の相関性検証

以上の節で提案したデザインパターンの有効性を確認するため，本節では車載用制御システムのデファクトとなっている CAN バスで制御される DCS を対象に検証を行った．CAN バス制御の DCS では，一般に高速な応答が求められる事が多く，異なる制御ノード間での通信時における通信時間遅れの定量的予測が重要となる．通信時間遅れは，CAN 用制御ノード内部での送信・受信処理自体に要する時間，バス上をメッセージが通過する時間，さらに複数ノードの送信が衝突した場合，プロトコルにしたがってメッセージが送信待ちとなる時間の総和として表される．このうち，バス上をメッセージが通過する時間は微少であるため，制御ノード内部での送信・受信処理時間の一部に含めて考えることができる．本 DCS シミュレータで通信遅れ時間を定量予測するには，制御ノード内部で送・受信処理にかかる時間を定量予測するモデルが必要であり，一方，送信衝突による待ち時間については，これらの予測時間から提案した Protocol パターンを使って自動的に計算が可能である．

そこで本研究では，まず 2 台の CAN 制御ノード実機を接続した予備通信実験により，CAN 用制御ノード内部での実際の送受信処理時間と，バス上のメッセージ通過時間を精密に測定し, 制御ノード内部での処理時間の定量予測モデルを構築した．さらに, この予測モデルを本 DCS シミュレータ内に組み込み, 3 台以上のノード間で複雑なメッセージ通信を行なわせた際の通信時間遅れを予測させ, これを実機の時間遅れと比較検証し，デザインパターンの検証を行った．

5.1 予備通信実験

図 6 のように，2 台の CAN 制御ノード（北斗電子製 LIN・CAN スタータキット）を接続し，制御ノード内部の送信・受信処理自体に要する時間を，ロジックアナライザを用いて実測する予備通信実験を行った．本実験では，図 7 のように，CPU が測定開始用ビット信号を出力してから CAN コントローラが

Table 1 Send and receive delays of CAN controller

データ長 [byte]	0	1	2	3	4	5	6	7	8
T_1の平均値 [μs]	47.68	47.68	47.68	47.66	47.68	47.66	47.66	47.70	47.66
T_4の平均値 [μs]	188.8	232.2	264.8	300.6	336.4	372.8	408.6	452.8	488.4

フレーム送信を開始するまでの時間 T_1，フレームが CAN トランシーバによって送信開始されてから CAN バスの中間点に達するまでの時間 T_2，フレームが CAN バスの中間点から CAN トランシーバによって受信受信開始されるまでの時間 T_3，CAN コントローラがフレーム受信を開始してから CPU がビット信号を出力するまでの時間を T_4 測定した．これらの測定値から，制御ノードにおける CAN コントローラでの送信・受信所要時間 $T_{c\text{-}send}$・$T_{c\text{-}receive}$，CAN トランシーバでの送信・受信所要時間 $T_{t\text{-}send}$・$T_{t\text{-}receive}$ へは, (1)式～(4)式により変換が行える．

$$T_{c\text{-}send} = T_1 + 0.05\ [\mu s] \quad (1)$$

$$T_{t\text{-}send} = T_2 - 0.0075\ [\mu s] \quad (2)$$

$$T_{t\text{-}receive} = T_3 - 0.0075\ [\mu s] \quad (3)$$

$$T_{c\text{-}receive} = T_4 - 0.05\ [\mu s] \quad (4)$$

ここで，0.05μs は CPU がビット信号出力関数を呼び出すのに要する時間, 0.0075 μs は CAN バス上でフレームを 1.5m 伝送するのに要する時間である．

5.2 CAN 制御ノード内部での送信・受信処理時間の予測モデル構築

$T_{t\text{-}send}$, $T_{t\text{-}receive}$ は通信データ長に依存しないという仮定の下で, 0.01μs の測定分解能で CAN トランシーバの T_2，T_3 を各々 10 回ずつロジックアナライザで測定した結果，分解能と同程度しか変化しない事が明らかとなった．そこで各測定結果の平均値を T_2，T_3 として上述の関係に代入する事により，$T_{t\text{-}send}$, $T_{t\text{-}receive}$ は以下の値で近似できる事が分かった．

$$T_{t\text{-}send} \approx 0.038 - 0.0075 = 0.031\ [\mu s] \quad (5)$$

$$T_{t\text{-}receive} \approx 0.353 - 0.0075 = 0.346\ [\mu s] \quad (6)$$

一方，図 6 の装置において 0, 1, …, 8 バイトのデータを通信する時，0.01μs の測定分解能で CAN コントローラの T_1，T_4 を各々 5 回ずつ測定した結果，分解能と同程度しか変化しない事が明らかとなった．そこで各測定結果の平均値を計算する事により, 表 1 が得られた. 表 1 より, T_1 はデータ長に依存せず, ほぼ一定である事が分かる. ゆえに, 表 1 の T_1 の平均値を T_1 として (1) 式に適用する事により，$T_{c\text{-}send}$ は (7) 式で近似できる．

$$T_{c\text{-}send} \approx 47.7 + 0.05 = 47.75\ [\mu s] \quad (7)$$

一方, 表 1 より, CAN データフレームの全ビット長 N_{frame} を横軸，T_4 を縦軸としてプロットすれば，T_4 は N_{frame} の一次関数となる．ゆえに，このプロットから回帰直線を計算する事により，T_4 は (8) 式のような一次関数として近似できる．

$$T_4 \approx 3.997 \times N_{frame} + 0.737\ [\mu s] \quad (8)$$

表 1 の T_4 と (8) 式の T_4 の誤差を計算すると，その最大値は約 0.1 μs となる．この値は CAN の最大通信速度（1 Mbps）における 1 ビットあたりの伝送時間のたかだか 0.1 倍なので，充分小さいものとして無視できる．(8) 式を (4) 式に適用する事により，$T_{c\text{-}receive}$ は (9) 式で近似できる．

$$T_{c\text{-}receive} \approx 3.997 \times N_{frame} + 0.687\ [\mu s] \quad (9)$$

5.3 通信実験と DCS シミュレーションの相関性の評価

5.2 節で構築した CAN 制御ノード内部での送信・受信処理時間の定量予測モデル式 (5), (6), (7), (9) を DCS シミュレータの通信プロトコルモデル内に組み込み，より複雑な制御ノード構造と通信シナリオを持つケースにおいて，実機とシミュレータでの通信遅れ時間の結果を比較検証した．

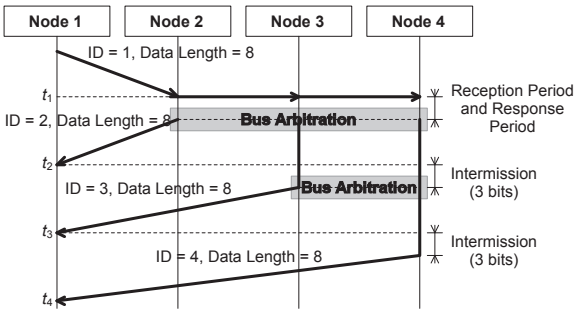

Fig.8 Communication scenario among four CAN control nodes

Table 2 Comparison of experiment and simulation results

Rcv. Time	Experimental Results [μs]					Simulation Results[μs]	Maximum Error [μs]
	Result 1	Result 2	Result 3	Result 4	Result 5		
t_1	48	48	48	48	47	47	1
t_2	587	587	586	586	586	585	1
t_3	1090	1090	1090	1090	1090	1087	3
t_4	1594	1594	1954	1954	1953	1593	1

まず，先程と同様の 4 台の CAN 制御ノードを CAN バスで接続したモデルを定義し，ノード 1 がノード 2，3，4 へメッセージを送信し，これら 3 台のノードがほぼ同時に応答する図 8 に示す通信シナリオに基づいて, 通信速度を 250kbps に設定し, 受信時刻 t_1～t_4 を測定する通信実験を行う．図 8 において，制御ノード 2，3，4 はほぼ同時に応答を試みるため，バス調停が 2 回発生し，その結果 ID の小さいものから順に送信される．図 8 に基づいて実機による通信実験を 5 回実施した結果と，DCS シミュレーションの実行結果を比較したものを表 2 に示す．Pentium 4（2.66GHz）搭載の PC によるシミュレーション実行時間は約 3 秒であった. 表 2 により, 両者の最大誤差は 3μs であり，CAN ネットワークにおける 1 ビットあたりの伝送時間である 4μs 未満である事が分かる．ゆえに, 本研究で開発した DCS シミュレータは, CAN バス制御の DCS における通信時間遅れなどの見積もり・予測評価に充分な時間的精度を持つと言える．

6. 結　　言

本研究では，分散制御システムの並列離散事象シミュレーション機構の実現と，通信プロトコルモデルの体系的実装を目的とした研究を行い，以下の結論を得た．

(1) DCS モデルにおける制御ノード・センサ・アクチュエータ・制御対象モデルが送信・受信時刻を持つイベントを通信する事によって論理上並列に動作し，かつ大域的に同期するシミュレーション機構を実現できるデザインパターンとして，Time-Warp パターンを新たに提案した．

(2) DCS におけるネットワークの通信プロトコルの静的構造・動的挙動を，ネットワークの形式から独立して汎用的にモデル化可能なデザインパターンとして，Protocol パターンを新たに提案した．

(3) (2) のパターンから，通信プロトコルモデルの実行可能 Java コードへ変換するための, システマチックな実装手続きを提案した．

(4) 提案するパターンと実装手続きを，提案した DCS シミュレータへ組み込んで，車載 LAN 用の CAN バス制御 DCS の挙動を予測する機能を開発し, その実行結果と実機上での実行結果を比較した．その結果，DCS の制御性

能に重要となるネットワーク通信時間遅れの見積もりが最大誤差 3μs と実用上問題のない範囲である事を示し，提案するパターンと実装方法に基づく DCS 向けの並列離散事象シミュレーション開発方法論の妥当性を確認できた．

今後の課題として，LonWorks ネットワークに関する通信特性の相関性検証が挙げられる．これに関しては続報で報告する．

謝　　辞

本研究は，文部科学省科学研究費補助金・基盤研究 (C)（課題番号 15560089）の一環として進められた．記して感謝する．

参 考 文 献

1) 元吉 伸一: フィールドバス入門, 日刊工業新聞社, (2000).
2) R. Piggin, *et al.*: The Current Fieldbus Standards Situation – A European View, Assembly Automation, **19**, 4 (1999) 286.
3) E. Gamma, R. Helm, R. Johnson and J. Vlissides : オブジェクト指向における再利用のためのデザインパターン，ソフトバンクパブリッシング, (1999).
4) 戸村 豊明, 金井 理, 岸浪 建史, 上広 清, 山元 進: 分散制御システムの静的構造モデリングのためのオブジェクト指向デザインパターンに関する研究, 精密工学会誌, **69**, 6 (2003) 815.
5) 戸村 豊明, 金井 理, 岸浪 建史, 上広 清, 山元 進: 分散制御システムの動的挙動モデリングのためのオブジェクト指向デザインパターンに関する研究, 精密工学会誌, **71**, 3 (2005) 379.
6) EIA/CEA-709.1-B: Control Network Protocol Specification, (2002).
7) ISO 11898-1: Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling, (2003).
8) Welcome to ControlNet International (http://www.controlnet.org/).
9) R. M. Fujimoto : Parallel and Distributed Simulation Systems, John Wiley & Sons Inc., (1999).
10) J. Misra : Distributed Discrete-Event Simulation, ACM Computing Surveys, **18**, 1 (1986) 39.
11) D. M. Nicol : The Cost of Conservative Synchronization in Parallel Discrete Event Simulations, J. ACM, **40**, 2 (1993) 304.
12) D. R. Jefferson : Virtual Time, ACM Transactions on Programming Languages and Systems, **7**, 3 (1985) 404.
13) B. P. Zeigler, H. Praehofer and T. G. Kim : Theory of Modeling and Simulation ― Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, (2000).
14) The Network Simulator - ns-2 (http://www.isi.edu/nsnam/ns/).
15) About GloMoSim (http://pcl.cs.ucla.edu/projects/glomosim/).
16) MIRAI-SF Simulator (http://www2.nict.go.jp/w/w121/mirai-sf/index.html).
17) OPNET Technologies, Inc. (http://www.opnet.com/).
18) Scalable Network Technologies: Creators of QualNet Network Simulator Software (http://www.scalable-networks.com/).