組み込み制御システム向け時間駆動分散オブジェクト環境

石郷岡 祐[†] 横山孝典[†]

本論文では，組み込み制御システム向けの時間駆動分散オブジェクトモデルに基づく分散処理環境について述べる．時間駆動分散オブジェクトモデルは自分自身の属性値を周期的に更新するオブジェクト群からなり，制御ロジックを制御ブロック図で記述する自動車制御等の組み込み制御システムの記述に適している．本モデルに基づく実行環境として，レプリカオブジェクトにより効率的な位置透過性を実現する分散処理ミドルウェアを開発した．本ミドルウェアは，スタブを経由してオリジナルのオブジェクトから属性値を取得し，それを周期的な状態メッセージとしてレプリカに転送することで，両者の一貫性を保つ．また開発環境として，CORBA 準拠の IDL で記述されたインタフェース定義からスタブを生成する IDL コンパイラと，GUI により入力した対象システムの構成情報から，ミドルウェアの処理に必要なコンフィギュレーションデータを生成するコンフィギュレータを提供する．

A Time-triggered Distributed Object Computing Environment for Embedded Control Systems

Tasuku Ishigooka[†] and Takanori Yokoyama[†]

The paper presents a distributed computing environment based on a time-triggered distributed object model for embedded control systems. The time-triggered distributed object model consists of distributed objects that periodically update their own attribute values. The model is suitable for embedded control systems such as automotive control systems in which the control logics are designed with control block diagrams. We have developed a distributed computing middleware that provides efficient location transparency with replica objects. The middlleware maintains the replicas to be consistent with the original objects using periodic state messages. We have also developed a development environment with an IDL compiler and a configurator. The IDL compiler generates stubs referring to the interface definitions written in CORBA-compliant IDL. The configurator with GUI generates configuration data for the middleware.

1. はじめに

組み込み制御システムは，自動車制御，FA（ファクトリーオートメーション），ビル管理システム等の広い分野で用いられており，その多くはハードリアルタイムシステムである．リアルタイムシステムの構築方法には，入力イベントに応じて処理を行うイベント駆動アーキテクチャと，周期的に処理を行う時間駆動アーキテクチャがある．ハードリアルタイムシステムには前者よりも後者の方が適しているといわれている[1]．また，一般にディジタル制御は制御ブロック図で記述され，一定の制御周期（サンプル周期）に従った周期的処理を基本としており，時間駆動アーキテクチャに適している．そのような時間駆動アーキテクチャ向けのオブジェクト指向ソフトウェアを実現するため，我々は，周期的に処理を実行するオブジェクト群からなる時間駆動オブジェクトモデルと，そのモデルに基づくソフトウェア開発法を提案した[2)~4)]．

近年，組み込み制御システムの分散化が進んでいる．分散制御システムはネットワークに接続された組み込みコンピュータの集合で，たとえば，自動車分散制御システムは CAN（Controller Area Network）[5] のようなリアルタイムネットワークに接続された複数の ECU（Electronic Control Unit）と呼ばれる組み込みコンピュータからなる．また，時間駆動アーキテクチャ向けのネットワークとして，時間駆動プロトコルに基づいた TTP[6]，FlexRay[7] が発表されている．我々は，上記時間駆動オブジェクトモデルを分散環境に適用する場合，レプリカを用いて位置透過性を実現できることを示した[2]．すなわち，他のコンピュータ上のオブジェクトを参照する場合，参照先のオブジェ

† 武蔵工業大学
Musashi Institute of Technology

クトのレプリカを参照元のコンピュータ上に配置することで，位置透過性を実現する．しかし，レプリカの設置やレプリカの整合性を維持するためのパラメータの設計は開発者に任されており，また，プログラミング言語や CPU アーキテクチャに依存した設計が必要なため，効率的なシステム開発を可能とするには十分ではなかった．

情報処理分野では，クライアント・サーバ型モデルに基づく分散オブジェクト環境として CORBA [8] が広く用いられている．CORBA では，インタフェース定義言語（Interface Definition Language，IDL）を導入するとともに，スタブやスケルトンを生成する IDL コンパイラ等の開発環境を提供しており，プログラミング言語に依存しない，効率的なアプリケーション開発が可能になっている．また，リアルタイムシステムや組み込みシステム向けに Real-Time CORBA [9] や Minimum CORBA [10] も提案されているが，CORBA は RPC（Remote Procedure Call）に基づくメッセージパッシングを採用しており，ネストされたオブジェクト間通信による遅延やジッタを避けることは難しく，時間駆動アーキテクチャには適していない．

そこで本研究の目的は，時間駆動分散オブジェクトモデルに基づく組み込み制御システムを効率良く開発できる時間駆動分散オブジェクト環境を開発することである．そのため，リアルタイム性に優れた時間駆動分散オブジェクトの実行環境を開発するとともに，インタフェース定義言語の導入や IDL コンパイラ等の開発環境を提供する．

本研究では特に，制御ブロック図を用いて制御ロジックの設計がなされる組み込み制御システムを対象とする．自動車制御等の分野では，最近，MATLAB/Simulink [11] のような制御ブロック図ベースの CAD/CAE ツールを用いた，いわゆるモデルベース制御設計が広く行われるようになっており，多くの制御ロジックが制御ブロック図で記述されている．

上記目的を達成するために，レプリカによる位置透過性を効率良く実現できる時間駆動分散オブジェクトの実行環境と開発環境を開発する．本実行環境は，RPC ベースの遠隔メソッド呼び出しを用いることなく，レプリカオブジェクトのローカルメソッド呼び出しにより遠隔オブジェクトを参照できる．これにより，位置透過性とリアルタイム性の両者を実現でき，組み込み制御システムのようなハードリアルタイムシステムにも適用可能な，時間駆動分散オブジェクト環境を提供できる．また，オブジェクトのインタフェース定義からレプリカの実現に必要なコードを自動生成する IDL コンパイラ等の開発環境も提供する．

本論文の構成は以下のとおりである．まず，2 章で時間駆動分散オブジェクトモデルについて述べる．次に，3 章で時間駆動分散オブジェクトの実行環境について，4 章で開発環境について述べる．そして，5 章で時間駆動分散オブジェクト環境の性能評価を行い，6 章で関連研究について論じる．最後に 7 章で本論文の結論を述べる．

2. 時間駆動分散オブジェクト

2.1 時間駆動オブジェクトモデル

従来のオブジェクトモデルはイベント駆動アーキテクチャに基づいており，**図 1** (a) に示すように，オブジェクトはメッセージを受け取り，メッセージを受信したときメソッドを実行する．これに対し，我々が提案した時間駆動オブジェクトモデル [2] は，図 1 (b) のようにオブジェクトのメソッドを周期的に実行する．

図 2 に時間駆動オブジェクトの基本クラスを示す．基本クラス ValueObject は，データ値を記憶する属性 value と，その値を算出する演算を行う update メソッドからなる．update メソッドは引数を持たず，データ値の算出に他のオブジェクトの属性値が必要な場合は，そのオブジェクトの get メソッドを呼び出して得る．update メソッドは，リアルタイム OS（Real-Time Operationg System，RTOS）の周期タスクによって実行される．

本論文が対象とする自動車制御等の組み込み制御システムでは，制御ロジックの設計工程において制御ブ

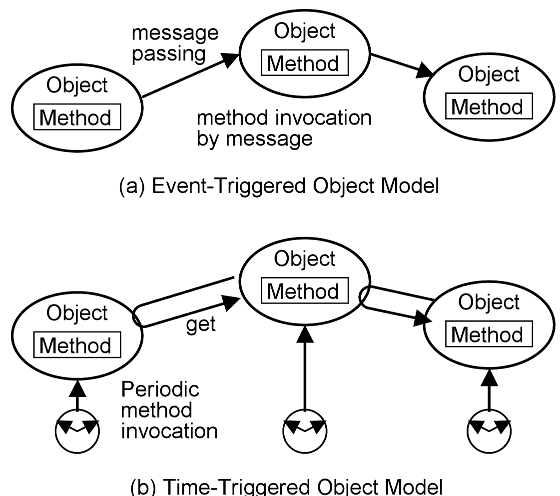

図 1　イベント駆動オブジェクトモデルと時間駆動オブジェクトモデル

Fig. 1　Event-triggered object model and time-triggered object model.

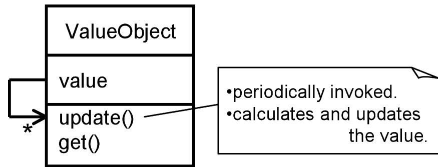

図 2 時間駆動オブジェクトの基本クラス
Fig. 2 Base class for time-triggered objects.

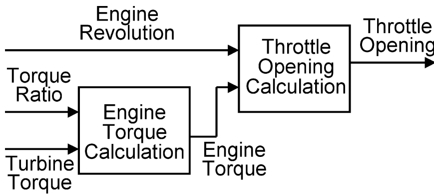

図 3 制御ブロック図の例
Fig. 3 Example block diagram.

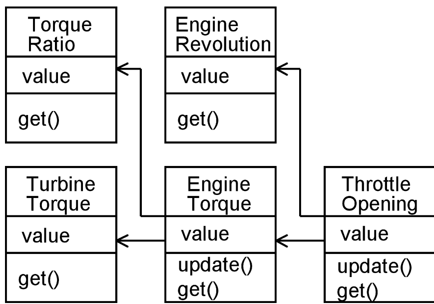

図 4 クラス図の例
Fig. 4 Example class diagram.

ロック図を用いることが多く，特に，フィードバック制御やフィードフォワード制御等の記述に広く使われる．**図 3** に制御ブロック図の例を示す．この例はエンジントルク（EngineTorque）を演算するブロックとスロットル開度（ThrottleOpenning）を演算するブロックからなり，データフローの上流から下流に向かって各ブロックの演算を行うのが，1 制御周期の処理である．それを制御周期に従って周期的に実行することで，制御処理が実現できる．

時間駆動オブジェクトモデルは，制御ロジックを制御ブロック図で表す組み込み制御システムに適している．前述のように，制御ロジックは制御に必要なデータ値を周期的に算出することで実現されるため，それらデータ値に対応させた時間駆動オブジェクトの集合により，制御処理を実現できる．

図 4 に，図 3 の制御ブロック図に対応する時間駆動オブジェクトのクラス図を示す．エンジントルクを表すクラス EngineTorque やスロットル開度を表すクラス ThrottleOpening は，基本クラス ValueObject のサブクラスである．たとえば ThrottleOpening は，エンジン回転数を表すクラス EngineRevolution と EngineTorque の参照を有し，それらの get メソッドを呼び出してエンジン回転数とエンジントルクの値を得，スロットル開度を算出し，自分自身の属性 value に記憶する．1 制御周期の処理では，制御ブロック図の表すデータフローの上流から下流に向かって，各オブジェクトの update メソッドを順に呼び出す．これを制御周期に従って周期的に実行することで制御処理を実現できる．

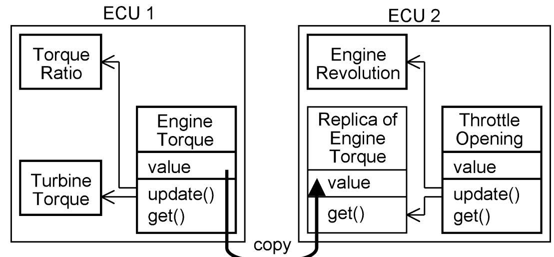

図 5 時間駆動分散オブジェクトモデルの例
Fig. 5 Example time-triggered distributed objects.

2.2 時間駆動分散オブジェクトモデル

前節で述べた時間駆動オブジェクトを分散処理環境向けに拡張したのが時間駆動分散オブジェクトモデルである．本モデルでは，レプリカオブジェクトを用いて分散処理環境での位置透過性とリアルタイム性の両者を実現する．すなわち，他の ECU から参照されるオブジェクトについて，参照元のオブジェクトが存在する ECU 上に，参照先のオブジェクトのレプリカを配置することで位置透過性を実現する．

時間駆動分散オブジェクトを用いた分散制御システムの例を**図 5** に示す．2 つの ECU があり，図 4 で示したオブジェクトが分散配置されている．図 5 の例では，EngineTorque オブジェクトのレプリカが ECU2 に配置されている．ThrottleOpening オブジェクトは EngineTorque オブジェクトのレプリカを参照し，その get メソッドを呼び出してその属性値を得ることができる．

以下，参照先の本来のオブジェクトをオリジナルオブジェクト，参照元に配置されるレプリカをレプリカオブジェクトを呼ぶことにする．レプリカオブジェクトの状態とオリジナルオブジェクトの状態はつねに一致していなければならない．そこで，オリジナルオブジェクトの属性値をレプリカへ周期的にコピーする複製処理を行うことで，両者の一貫性を保持する．時間駆動オブジェクトモデルでは，属性の更新はそのオブジェクト自身によってのみ行われるため，1 方向のコピーで一貫性を保持できる．これは，制御ブロック図

におけるブロック間のデータの流れが 1 方向であることに対応している．複製処理は，状態メッセージ[12]により効率的に実装できる．状態メッセージとは，受信時にキューイングを行わず，上書きしてしまうメッセージである．

図 4 に示したオブジェクト群を，クライアント・サーバ型モデルに基づく CORBA のような分散オブジェクト環境により実装する場合，複数の ECU に分散配置したオブジェクトを，データフローの上流のオブジェクトから下流のオブジェクトのメソッドを遠隔メソッド呼び出しにより実装することが考えられる．しかし，その場合はネットワーク通信を含む遠隔メソッド呼び出しがネストして呼び出されることになり，実行時間の予測が困難になる．また，図 4 に示したオブジェクト群において，update の呼び出しを周期的に行い，get メソッドの呼び出しを遠隔メソッド呼び出しにより実現する方法も考えられる．しかし，get メソッドの処理にネットワーク通信が含まれるため実行時間の予測が難しくなるうえ，RPC ベースの遠隔メソッド呼び出しは双方向の同期通信で実装されるため，オーバヘッドも大きくなる．

本モデルでは，遠隔メソッド呼び出しを用いることなく，レプリカに対するローカルメソッド呼び出しにより他 ECU 上のオブジェクトの属性を得ることができる．get メソッドの処理はネットワーク通信を含まず，ローカルオブジェクトの場合と同じ処理のため，実行時間の予測が容易である．また，update メソッドはネストして呼び出されることはないため，処理全体の実行時間は各オブジェクトの update メソッドの処理時間の和になり，その予測も容易である．さらに，複製処理のメッセージは周期的に転送されるため，ネットワークトラフィックの変動も少なく，通信遅延時間の予測も容易である．

3. 実行環境

3.1 ソフトウェア構成

時間駆動分散オブジェクトに基づくソフトウェアの全体構成を**図 6** に示す．全体ソフトウェアは，アプリケーションプログラム，ミドルウェア，RTOS，CAN ドライバで構成される．

本研究では，自動車のパワートレイン系の制御を主な対象とし，各 ECU を接続するネットワークには CAN を使用する．本来時間駆動アーキテクチャには時間駆動プロトコルに基づくネットワークが適しているが，現時点ではそれほど普及はしていないため，本研究の最初のステップとしては CAN を用いることとした．CAN はコスト効率が良く，現在自動車のパワートレイン系の制御において広く用いられている．CAN を用いて時間駆動アーキテクチャを実現するため，ミドルウェアは ECU 間の同期をとる機能を提供することとする．

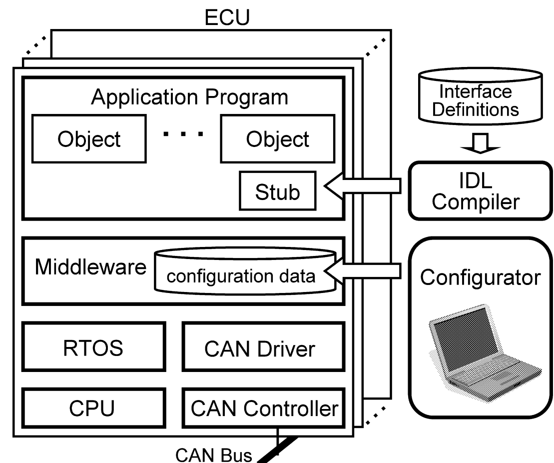

図 6 分散オブジェクト環境
Fig. 6 Time-triggered distributed object computing environment.

アプリケーションプログラムは，制御ロジックを実現する時間駆動オブジェクトからなる．時間駆動オブジェクトの update メソッドは，RTOS の周期タスクにより実行される．

ミドルウェアはオリジナルオブジェクトの属性値をレプリカオブジェクトに転送する，複製処理の機能を提供する．複製処理により，オリジナルオブジェクトとレプリカの一貫性を維持できる．

オリジナルオブジェクトとミドルウェア，レプリカとミドルウェアの仲介を行うために，スタブを使用する．オリジナルオブジェクトとの仲介を行うスタブをオリジナルスタブ，レプリカとの仲介を行うスタブをレプリカスタブと呼ぶことにする．スタブは，IDL により記述されたオブジェクトのインタフェース定義から，IDL コンパイラにより生成される．スタブと IDL コンパイラの詳細は後述する．

ミドルウェアはメッセージ ID，データ型，データ長，通信周期等のコンフィギュレーションデータを参照し，複製処理を実行する．コンフィギュレーションデータはコンフィギュレータにより，静的に生成される．コンフィギュレーションデータとコンフィギュレータの詳細は後述する．

3.2 ミドルウェア

ミドルウェアはレプリカを用いて位置透過性のある環境を実現する．ミドルウェアは，オリジナルオブジェ

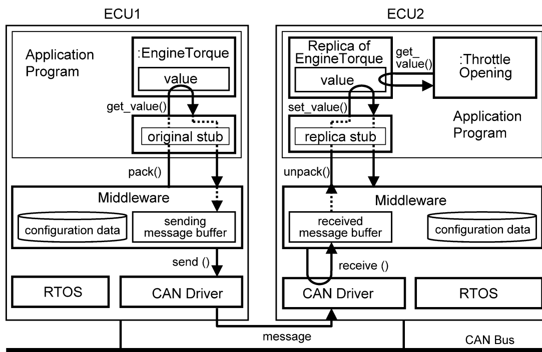

図 7 複製処理の流れ
Fig. 7 Processing flow of replication.

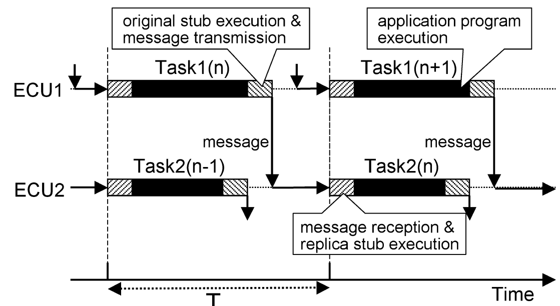

図 8 時間駆動処理
Fig. 8 Time-triggered processing.

クトの状態をレプリカへ周期的にコピーする複製処理により，それらの間の一貫性を保持する．

複製処理の流れを**図 7** に示す．この例では，EngineTorque と ThrottleOpening という 2 つのオブジェクトがあり，ECU1 に EngineTorque を，ECU2 に EngineTorque のレプリカと ThrottleOpening を配置する．また，EngineTorque のオリジナルスタブは ECU1 に配置され，EngineTorque のレプリカのレプリカスタブは ECU2 に配置されている．

ミドルウェアの複製処理を，図 7 を用いて説明する．まず，送信側について説明する．ECU1 のミドルウェアは EngineTorque の属性値を得るためにオリジナルスタブを呼ぶ（pack()）．オリジナルスタブはアクセスメソッド（get_value()）を呼び出して属性値を取得し，ミドルウェア中の送信メッセージバッファに格納する．そして，ミドルウェアは CAN ドライバに送信メッセージバッファのデータをメッセージとして送信するように要求を出す（send()）．

CAN の 1 メッセージのデータサイズは最大 8 バイトである．よって，1 オブジェクトの送信データが 8 バイトを超える場合は，メッセージ ID の異なる複数のメッセージとして送信する．また，1 メッセージあたりの送信データはできるだけ大きい方が効率は良い．そこで，1 オブジェクトのデータが小さい場合は，複数組み合わせて（パックして）送信する．このため，複製処理の周期とメッセージ ID が同一の場合には，複数のオリジナルオブジェクトの属性値を 1 つの送信メッセージバッファにパックして格納することとする．パックされたメッセージを受信したミドルウェアは，受信メッセージバッファの内容を各々のレプリカへ格納する．これにより効率的な複製処理が可能である．メッセージの分割やパックに関する情報は，コンフィギュレーションデータとして与えられる．

次に，受信側について説明する．ECU2 のミドルウェアは，CAN ドライバに他の ECU から受信したメッセージを受信メッセージバッファに格納するように要求を出す（receive()）．次に，ミドルウェアは EngineTorque のレプリカのレプリカスタブを呼ぶ（unpack()）．レプリカスタブは受信メッセージバッファの内容を読み出し，レプリカのアクセスメソッド（set_value()）を呼び出してその属性に格納する．

ECU 間のタスク起動の同期は，マスタ ECU が他の ECU に同期メッセージを周期的に送り，マスタ ECU に同期を合わせることで実現している．同期用のメッセージは最高優先度としている．また同期機能の実行周期は，すべてのタスク周期の最小公倍数とした．なお，本機能は CAN 向けに開発したものであり，将来時間駆動ネットワークを用いる場合には，本同期機能は不要となる．

3.3 時間駆動分散処理

各 ECU 上の時間駆動処理のタイムチャートの例を**図 8** に示す．ECU1 にはタスク Task1，ECU2 にはタスク Task2 が存在する．両者とも周期は T であり，ミドルウェアの同期機能により同期して起動することが可能になっている．ECU1 上の Task1 は，EngineTorque の update メソッドを含むアプリケーションプログラムを実行し，ECU2 の Task2 は，ThrottleOpening の update メソッドを含むアプリケーションプログラムを実行する．

前述のように制御アプリケーションでは，制御ブロック図のデータフローの上流のオブジェクトから下流のオブジェクトに向けて，順に update メソッドを呼び出していく必要がある．図 8 では，Task1(n) において呼び出された EngineTorque オブジェクトの update メソッドで算出した値は，Task2(n) において呼び出される ThrottleOpening オブジェクトの update メソッドで参照される．このように，上流側のオブジェクトの update メソッドの呼び出しで算出した値が，下流側のオブジェクトの次の周期での update メソッドの

実行で参照されるように保つことで，データの時間的（周期的）な整合性を維持する．

データの整合性を保つには，各 ECU 上のタスクの処理時間が 1 周期内に納まるように設計する．ただし，実際には複製処理もネットワーク通信によるジッタが発生するので，1 制御周期からジッタ分を除いた時間内にタスクの処理が納まるように設計する必要がある．

図 8 に示すように，複製処理はアプリケーションプログラムの処理（update メソッドの処理）の前後に実行される．Task1 は，アプリケーションプログラムを実行した後，EngineTorque のオリジナルスタブを呼ぶミドルウェアの機能を実行し，CAN ドライバにメッセージの送信要求を出すことで，ECU2 へのメッセージが送信される．Task2 は，まず最初に CAN ドライバにメッセージの受信要求を出して受信データを受け取り，EngineTorque のレプリカスタブを呼ぶミドルウェアの機能を実行した後に，アプリケーションプログラムを実行する．ThrottleOpening オブジェクトが EngineTorque のレプリカオブジェクトの get メソッドを呼ぶ処理は，このアプリケーション中で実行される．以上のように，同一の周期のアプリケーションプログラムと複製処理は同じタスクによって実行されるため，両者の間で競合は生じない．

4. 開発環境

4.1 概 要

CORBA のような分散オブジェクト環境では，言語非依存の環境を実現するため，オブジェクトのインタフェースを IDL で記述する方法が用いられる．我々も，言語非依存の環境でスタブ等を生成可能とするため，IDL と IDL コンパイラを実装した．また，ミドルウェアの複製処理は，複製処理の数，周期，データサイズ，メッセージの ID 等のコンフィギュレーションデータを参照し，処理を行う．そこで，コンフィギュレーションデータを生成するコンフィギュレータを開発した．IDL コンパイラとコンフィギュレータで生成したソースコードと，アプリケーション，ミドルウェア，CAN ドライバのソースコードを合わせてコンパイルすることで，時間駆動分散オブジェクトに基づいた組み込み制御システムを構築できる．

開発環境の対象言語としては C 言語を選定した．これは，組み込み制御システムの場合，1 チップマイクロコントローラの内蔵 ROM・RAM を使用する等のメモリサイズの制約が厳しく，また実装においては必ずしもオブジェクト指向のすべての機能は必要とは限らず，オブジェクト指向に基づくソフトウェアであっても，C 言語を用いて実装することが多いためである[13]．そこで我々は，時間駆動分散オブジェクトで構成されたアプリケーションが C 言語で実装される場合を想定し，本研究の最初のステップとしての対象言語を C 言語とした．

```
/* EngineTorque.h */
/* Class Definition */
typedef struct {
    short value;
    short limit;
} EngineTorque;
/* prototype declaration */
void EngineTorque_update(EngineTorque*);
/* macros for access methods */
#define EngineTorque_get_value(this)\
                        (this->value)
#define EngineTorque_get_limit(this)\
                        (this->limit)
```

図 9 C 言語のクラス定義の例
Fig. 9 Example class definition in C.

```
interface EngineTorque {
    attribute short value;
}
```

図 10 IDL のインタフェース定義の例
Fig. 10 Example interface definition in IDL.

C 言語で記述した EngineTorque クラスのソースコードの例を**図 9** に示す．クラスは属性値 value，limit を要素とする構造体で表され，update メソッドは関数で表される．アクセスメソッドはマクロで効果的に定義される．関数やマクロの第 1 引数は，インスタンスオブジェクトへのポインタである．

4.2 IDL コンパイラ

他の ECU から参照される（レプリカを生成する）オブジェクトのインタフェースを CORBA 準拠の IDL で記述する．すなわち，定義するインタフェースは複製処理が必要なオブジェクトのみでよい．本 IDL は CORBA IDL のサブセットである．たとえば，CORBA IDL の “module” 宣言はサポートしていない．

図 9 で示した EngineTorque クラスのインタフェース定義の例を**図 10** に示す．interface 名にはオリジナルオブジェクトのクラス名を用いる．ここで，複製処理の対象となる属性を指定するために，IDL の属性宣言 “attribute” を用いることとした．すなわち，IDL で定義された属性のみをレプリカにコピーする．図 10 の例では，属性 value のみが属性宣言されている．し

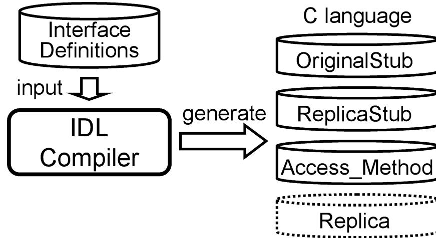

図 11 スタブ生成の流れ
Fig. 11 Stub generation by IDL compiler.

```
/* EngineTorque.h */
/* Replica Class Definition */
typedef struct {
    short value;
} EngineTorque;
```

図 12 レプリカクラスの例
Fig. 12 Example replica class code.

```
/* EngineTorque_access.h */
/* access method for EngineTorque */
#ifndef EngineTorque_get_value
#define EngineTorque_get_value(this)\
                        (this->value)
#endif
#ifndef EngineTorque_set_value
#define EngineTorque_set_value(this, v)\
                        (this->value = v)
#endif
```

図 13 アクセスメソッドの例
Fig. 13 Example access method code.

たがって，属性 limit はコピーされず，属性 value のみがレプリカへコピーされる．

IDL により記述したインタフェース定義を IDL コンパイラに入力することで，スタブを生成する．**図 11** に示すように，IDL コンパイラはオリジナルスタブ，レプリカスタブ，オブジェクトやレプリカのアクセスメソッドのソースコードファイルを生成する．また，IDL コンパイラは必要に応じてレプリカクラスを生成できる．レプリカクラスとオリジナルオブジェクトクラスが同一ソースコードで良い場合には，レプリカクラスを IDL コンパイラで生成する必要はない．各ファイルはクラス単位で生成される．

レプリカクラスの例を**図 12** に示す．また，EngineTorque のアクセスメソッドの例を**図 13** に示す．アクセスメソッドは属性の読み書き用のメソッドであり，スタブによって呼び出される．

オリジナルスタブの簡単な例を**図 14** に示す．_EngineTorque_pack() は図 7 の pack() に対応する．

```
/* EngineTorque_stub.c */
#include "EngineTorque.h"
#include "EngineTorque_access.h"
/* Original Stub for EngineTorque */
void _EngineTorque_pack(EngineTorque* this,
                    Byte* addr, int offset) {
    short tmp;
    tmp = EngineTorque_get_value(this);
    *(addr + offset) = (byte)(tmp >> 8)
    *(addr + offset +1) = (byte)(tmp)
}
```

図 14 オリジナルスタブの例
Fig. 14 Example original stub code.

```
/* EngineTorque_rstub.c */
#include "EngineTorque.h"
#include "EngineTorque_access.h"
/* Replica Stub for EngineTorque */
void _EngineTorque_unpack(EngineTorque* this,
                    byte* addr, int offset) {
  short tmp;
  tmp = *(addr + offset) << 8;
  tmp += *(addr + offset +1);
  EngineTorque_set_value(this, tmp)
}
```

図 15 レプリカスタブの例
Fig. 15 Example replica stub code.

_EngineTorque_pack() の引数は，順番にオリジナルオブジェクトのポインタ，送信メッセージバッファのポインタ，送信メッセージバッファのオフセットである．関数_EngineTorque_pack() は，EngineTorque の値をミドルウェアの送信メッセージバッファに格納する．CPU 依存であるアライメントの影響を受けずに格納することができるため，無駄のない複製処理が可能である．

レプリカスタブの簡単な例を**図 15** に示す．_EngineTorque_unpack() は図 7 の unpack() に対応する．_EngineTorque_unpack() の引数は，順番にレプリカのポインタ，受信メッセージバッファのポインタ，受信メッセージバッファのオフセットである．関数_EngineTorque_unpack() は，ミドルウェアの受信メッセージバッファの内容を EngineTorque のレプリカに格納する．レプリカスタブもオリジナルスタブと同様にアライメントの影響を受けず，効率良い複製処理が可能である．

また必要に応じて，ビッグエンディアンとリトルエンディアン間の互換性を保つ機能を備えたスタブを生成できる．ただし，図 14 と図 15 では，その部分を省略し簡略化している．

レプリカの有無は，オブジェクトを ECU に配置するときに決定する．そこで，レプリカのインスタン

ス宣言を含むソースコードファイルは，コンフィギュレータによって生成することとした．

4.3 コンフィギュレータ

GUI（Graphical User Interface）を用いて対象の分散システムの構成情報を入力することで，コンフィギュレータが自動的にコンフィギュレーションデータを生成する．GUI で入力する構成情報の項目とその詳細を**表 1** に示す．

コンフィギュレータは，表 1 の入力項目に従って各々の ECU に対するコンフィギュレーションデータを生成する．コンフィギュレーションデータは C 言語で記述されている．

ミドルウェアはこのコンフィギュレーションデータを参照し処理を行う．コンフィギュレーションデータの内容とその詳細を**表 2** に示す．

前節で述べたように，レプリカのインスタンスの宣言もコンフィギュレータにより生成する．EngineTorque のレプリカのインスタンス宣言の例を**図 16** に示す．レプリカのインスタンスオブジェクトの名前は，オリジナルオブジェクトのインスタンスオブジェクトの名前と同一である．

表 1 構成情報
Table 1 Information for configuration.

入力項目	詳細
ECU の台数	対象システムの ECU の台数
マスタ ECU	選択した ECU に同期を合わせる
複製処理の数	すべての複製処理の数
クラス	オリジナルオブジェクトのクラス名
インスタンス	インスタンスオブジェクト名
実行周期	複製処理を実行する周期
データサイズ	複製処理の転送データサイズ
メッセージ ID	複製処理を転送する CAN メッセージ ID
配置情報	オブジェクトとレプリカの配置情報

表 2 コンフィギュレーションデータ
Table 2 Configuration data.

内容	詳細
バッファ生成	送信，受信メッセージバッファの定義
レプリカ生成	レプリカのインスタンス宣言
初期化処理	複製処理を行う周期タスクの設定
仲介処理	複製処理でスタブを呼び出す
送受信処理	CAN ドライバを用いた複製処理の送受信
同期処理	マスタ ECU に同期を合わせる

```
/* EngineTorque_replica.c */
#include "EngineTorque.h"
#include "EngineTorque_access.h"
/* Replica Object */
EngineTorque engineTorque;
```

図 16 レプリカのインスタンス宣言の例
Fig. 16 Example replica instance code.

5. 性能評価

開発した時間駆動分散オブジェクト環境の評価実験を行った．実験では，ハードウェアとして CAN コントローラを内蔵したマイクロコントローラ（H8S/2638）を，RTOS には μITRON[14] を用いた．

H8S/2638 を用いて測定した，複製処理における 2 つの ECU 間通信時間と，その最小値と最大値の差（ジッタ）を**表 3** に示す．同一のハードウェアタイマを用いて測定するため，2 つの ECU 間での往復の複製処理の実行時間を測定した．H8S/2638 のクロック周波数は 20 MHz であり，CAN の伝送速度は 1 Mbps である．表中の時間の単位はすべて μsec である．

表 3 には往復のジッタが示してあるが，片道の場合のジッタも同程度か，あるいはそれより小さいと予測できる．この結果によると，複製処理の対象バイト数の増加に従って複製処理時間も増加するが，ジッタの値はほぼ一定である．この結果から，CAN 通信のブロックが発生しない場合のジッタの値は 100 μsec 以下と予測できる．ここでブロックと呼んでいるのは，CAN ではいったんメッセージ送信が開始されると，それが完了するまで，より高い優先度のメッセージであっても送信が待たされる現象のことである．ブロックが起こる場合は，1 メッセージの通信時間（伝送速度 1 Mbps，送信データ 8 バイトのとき少なくとも 108 μsec）と同程度のジッタが発生するため，合計で 200 μsec 程度のジッタが発生する可能性がある．

3.3 節で述べたように，ジッタを考慮して制御周期内に処理が完了するように設計する必要がある．本研究で主な対象としているパワートレイン系の制御では，10 msec をベースに，その整数倍の制御周期がよく用いられる．上記ジッタの値は，10 msec 周期の場合の 2%程度であり，設計上大きな問題とはならないと考える．また，ジッタを考慮することはその分 CPU 利用率の低下を招くが，この程度の値であれば許容できる範囲と考える．

従来のクライアント・サーバ形式に基づく分散オブジェクト環境の場合は，遠隔メソッド呼び出しがネストするため，通信によるジッタは 1 回の通信のジッタ

表 3 往復の複製処理
Table 3 Time of round trip replication.

バイト	平均時間	最小値	最大値	ジッタ
1	259.2	252.8	345.6	92.8
2	316.8	297.6	390.4	92.8
4	387.2	371.2	464.0	92.8
8	544.0	531.2	627.2	96.0

を最大ネスト数倍した値となる．このため，制御周期 10 msec に対して設計上問題になる可能性がある．また，複雑なプログラムの場合には，最大ネスト数を明確に把握すること自体が容易ではない．ジッタ値の増大は CPU 利用率も低下させる．これに対し，時間駆動分散オブジェクト環境では，通信とは独立に，周期的に update メソッドが実行されるため，ジッタ値は蓄積されない．

また，レプリカオブジェクトを使用した時間駆動分散オブジェクト環境は，CORBA のような RPC ベースの分散オブジェクト環境より，処理のオーバヘッドも少ない．本方式の複製処理は一方向の通信で実現されるが，CORBA のような RPC ベースの分散オブジェクト環境では 1 回の RPC で双方向の通信を行うため，通信負荷は倍になる．また CORBA では，CPU アーキテクチャによるデータ表現の違い等を吸収するためのヘッダ情報が必要であり，CAN を用いた組み込みシステム向け CORBA [19] ではヘッダを圧縮しているものの，8 バイト中の 2 バイトをヘッダにあてている．これに対し本方式は，データ表現に関する情報をコンフィギュレーション情報として静的に与えることで，ヘッダ情報を不要とし，通信量を削減できる．

一方，本方式の問題点として，レプリカを設けることによるメモリ消費量の増大が懸念される．しかし前述のように，レプリカオブジェクトは複製が必要な属性の記憶領域とレプリカスタブのみを持てばよく，レプリカオブジェクト 1 つあたりのメモリ消費量は数十バイト程度である．たとえば自動車車間距離制御システムの試作例では，3 つの ECU 上の合計 137 のオブジェクトに対し，レプリカ数は 17 であり[2]，レプリカによるメモリ消費量の増大は実用上問題にならないと考える．

以上のように，自動車のパワートレイン系や，より大きなジッタが許容されるボディ系の制御システムについては，本環境は十分適用可能と考える．一方，X-by-wire アプリケーションのように，より小さなジッタが要求される分野に対応するには，時間駆動プロトコルに基づく時間駆動ネットワークが必要と考えている．本環境におけるジッタの主原因は CAN プロトコルにあり，今後，FlexRay 等の時間駆動ネットワークや OSEKtime [15] 等の時間駆動オペレーティングシステムの導入を検討している．

6. 関連研究との比較

これまでに周期的動作が可能な，いくつかのリアルタイムオブジェクトモデルが提案されている．たとえば，TSO（Time-Sensitive Object）モデル[16] はリアルタイムシステムの設計と分析に有効である．また TMO（Time-Triggered Message-Triggered Object）[17] モデルには周期的に実行される自発的メソッドがある．しかし，これらモデルのオブジェクト間通信はメッセージパッシングが基本であり，システム全体の動作を時間駆動で行うものではない．

また組み込みシステム向けに，Real-Time CORBA に基づく分散オブジェクト環境が提案されており，時間駆動イーサネットを用いた Real-Time CORBA [18] や CAN を用いた組み込みシステム向け CORBA [19] が報告されている．しかし，これらの分散オブジェクト環境におけるオブジェクト間通信は基本的にクライアント・サーバ型に基づく遠隔手続き呼び出しであり，時間駆動アーキテクチャに適したものではない．

これに対し我々は，制御ブロック図を用いて制御ロジックを表現可能な組み込み制御システムを対象として，レプリカを用いた時間駆動分散オブジェクト環境を開発した．本環境は，状態メッセージを用いた複製処理により位置透過性を効率良く実装できるとともに，ジッタが少なく処理時間の予測も容易である．このため，時間駆動アーキテクチャに基づく分散組み込み制御システムに適している．

7. 結　　論

制御ロジックを制御ブロック図で記述する組み込み制御システムを対象に，時間駆動分散オブジェクトモデルに基づく分散オブジェクト環境を開発した．本環境ではレプリカオブジェクトを利用することで，時間駆動アーキテクチャに適した位置透過性を実現している．また，開発環境として IDL コンパイラとコンフィギュレータを提供することで，効率の良いシステム開発を可能にした．

今後の計画として，時間駆動ネットワークや時間駆動オペレーティングシステムの導入や，C++言語に対応した IDL コンパイラの開発を考えている．

参 考 文 献

1) Kopetz, H.: Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Trans. Information & Systems*, Vol.E76-D, No.11, pp.1325–1332 (1993).
2) 横山孝典，納谷英光，成沢文雄，倉垣　智，永浦　渉，今井崇明，鈴木昭二：組込み制御システムのための時間駆動オブジェクト指向ソフトウェア開発法，電子情報通信学会論文誌，Vol.J84-D-I, No.4, pp.338–349 (2001).

3) 横山孝典：組み込み制御ソフトウェアのアスペクト指向に基づく開発法，情報処理学会論文誌，Vol.47, No.4, pp.1185–1194 (2006).
4) 吉村健太郎，宮崎泰三，横山孝典：オブジェクト指向組み込み制御システムのモデルベース開発法，情報処理学会論文誌，Vol.46, No.6, pp.1436–1446 (2005).
5) Kiencke, U.: Controller Area Network — from Concept to Reality, *Proc. 1st International CAN Conference*, pp.0-11–0-20 (1994).
6) Kopetz, H. and Grunsteidl, G.: TTP-A Protocol for Fault-Tolerant Real-Time Systems, *IEEE Computer*, Vol.27, No.1, pp.14–23 (1994).
7) Makowitz, R. and Temple, C.: FlexRay — A Communication Network for Automotive Control Systems, *Proc. 2006 IEEE International Workshop on Factory Communication Systems*, pp.207–212 (2006).
8) OMG Technical Document formal/02-06-01: The Common Object Request Broker: Architecture and Specification, Version 3.0 (2002).
9) OMG Technical Document formal/02-08-02: Real-Time CORBA Specification, Version 1.1 (2002).
10) OMG Technical Document formal/02-08-01: Minimum CORBA Specification, Version 1.0 (2002).
11) Moscinski, J.: *Advanced Control with MATLAB and Simulink*, Ellis Horwood, Ltd. (1995).
12) Kopetz, H. and Merker, W.: The Architecture of MARS, *Proc. 15th International Symposium on Fault-Tolerant Computing*, pp.274–279 (1985).
13) 成沢文雄，納谷英光，横山孝典：組み込み制御システムのためのオブジェクト指向コード生成ツール，情報処理学会論文誌，Vol.46, No.5, pp.1306–1317 (2005).
14) トロン協会 ITRON 仕様検討グループ：μITRON4.0 仕様，Ver.4.02.00 (2004).
15) OSEK/VDX: Time-Triggered Operating System, Version 1.0 (2001).
16) Callison, H.R.: A Time-Sensitive Object-Model for Real-Time Systems, *ACM Trans. Software Engineering and Methodology*, Vol.4, No.3, pp.287–317 (1995).
17) Kim, K.H.: Object Structures for Real-Time Systems and Simulators, *IEEE Computer*, Vol.30, No.8, pp.62–70 (1997).
18) Lankes, S., Jabs, A. and Reke, M.: A Time-Triggered Ethernet Protocol for Real-Time CORBA, *Proc. 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp.215–222 (2002).
19) Lankes, S., Jabs, A. and Bemmerl, T.: Integration of a CAN-based Connection-oriented Communication Model into Real-Time CORBA, *Proc. International Parallel and Distributed Processing Symposium* (*IPDPS'03*), p.121a (2003).

(平成 18 年 12 月 28 日受付)
(平成 19 年 6 月 5 日採録)

石郷岡 祐（学生会員）
1983 年生．2006 年武蔵工業大学工学部電子情報工学科卒業．同年同大学大学院工学研究科電気工学専攻修士課程入学．現在，同課程在学中．リアルタイム分散処理の研究に従事．

横山 孝典（正会員）
1959 年生．1981 年東北大学工学部通信工学科卒業．1983 年同大学大学院工学研究科電気及通信工学専攻修士課程修了．同年（株）日立製作所入社．1987 年から 1990 年まで（財）新世代コンピュータ技術開発機構出向．2004 年より武蔵工業大学．分散システム，組み込みシステム，ソフトウェア工学等の研究に従事．博士（情報科学）．電子情報通信学会，IEEE，ACM 各会員．