社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－171 (15)
2007－EMB－ 3 (15)
2007／1／23

組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
武蔵工業大学大学院 電気工学専攻
E-mail : g0663004@sc.musashi-tech.ac.jp, yokoyama@cs.musashi-tech.ac.jp

本論文では，自動車制御等の組込み制御システムを対象とした時間駆動に基づく分散オブジェクト環境について述べる．まず，自分自身の属性値を周期的に更新する分散オブジェクト群から成る時間駆動分散オブジェクトを提案する．このモデルではオブジェクト間通信に遠隔メソッド呼び出しを用いず，レプリカオブジェクトを用いて効率的な位置透過性を実現する．そして，時間駆動分散オブジェクトミドルウェアと開発環境を開発した．ミドルウェアはスタブを経由してオリジナルオブジェクトから属性値を取得し，周期的に状態メッセージをレプリカに転送し，その一貫性を保つ．スタブは CORBA 準拠の IDL で記述されたインタフェース定義より生成する．

A Time-Triggered Distributed Object Computing Enviroment for Embedded Control Systems

Tasuku Ishigooka　　Takanori Yokoyama
Department of Electrical Engineering, Graduate School of Musashi Institute of Technology
E-mail : g0663004@sc.musashi-tech.ac.jp, yokoyama@cs.musashi-tech.ac.jp

The paper presents a distributed object computing environment based on a time-triggered architecture for embedded control systems such as automotive control systems. A time-triggered distributed object model consists of distributed objects that periodically update their own attribute values. Replica objects are used for efficient location transparency in the model. Inter-object communications are not remote method invocations. We have developed a middleware and a development environment based on the time-triggered distributed object model. The middleware maintains the replicas to be consistent with the original objects getting values of the original object through stubs using periodic state messages. The stubs are generated form the interface definitions written in CORBA-compliant IDL.

1. はじめに

組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いる．組込み制御システム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を構築する方法には，イベント駆動アーキテクチャと時間駆動アーキテクチャがあるが，組込み制御システムでは周期的な処理が基本であるため，前者よりも後者の方が適している[1]．時間駆動アーキテクチャは周期的に外部入力に関する処理を行なうプログラムモジュールから成る．

また近年，組込み制御システムの分散制御化が進んでいる．分散制御システムはネットワークに接続された組込みコンピュータの集合である．例えば，自動車分散制御システムは，リアルタイムネットワークに接続された多数の ECU（Electronic Control Unit）から成る．ECU は自動車制御を行うための組込みコンピュータである．自動車制御で用いられるリアルタイムネットワークの例として，CAN (Controller Area Network)[2]，TTP[3]，FlexRay[4] が挙げられる．TTP と FlexRay は時間駆動プロトコルに基づいている．分散組込み制御システムに対応するため，時間駆動に基づく分散オブジェクト環境が求められている．

情報処理分野での分散オブジェクト環境として CORBA[5] 等が広く用いられている．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いる．また，リアルタイムシステムや組込みシステム向けに Real-Time CORBA[6] や Minimum CORBA[7] が提案され，組込み制御システム向けに CAN 上の Real-Time CORBA も定義されている[8]．しかし，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いため，RPC に基づく環境は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，時間駆動アーキテクチャに基づく組込み制御システム向けの分散オブジェクト環境を開発することである．

本目的を達成するために，我々は時間駆動分散オブジェクトモデルを提案し，そのモデルに基づく時間駆動分散オブジェクト環境を開発する．本環境では RPC を用いず，レプリカを用いることで位置透過性とリアルタイム性の両者を実現する．

レプリカを用いた時間駆動分散オブジェクト環境は，組込み制御システムのようなハードリアルタイムシステ

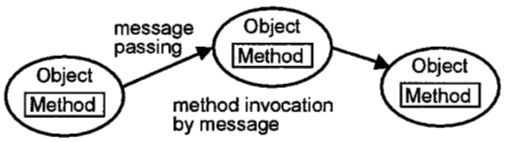
ムに適している．オブジェクト間通信を遠隔メソッド呼び出しではなく，レプリカオブジェクトのローカルメソッド呼び出しで実装できるため，特に粒度の小さいオブジェクト群で構成される組込み制御アプリケーションプログラムにも対応できる．

本論文の構成を以下に示す．**2** で時間駆動分散オブジェクトモデルを提案する．**3** で分散オブジェクトの環境について述べる．**4** で時間駆動分散オブジェクト環境の実装について述べる．**5** で本論文の結論を述べる．

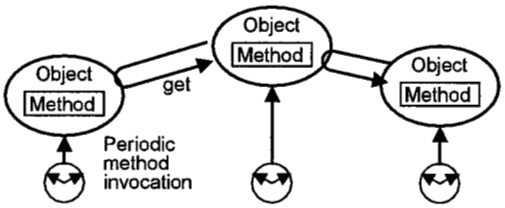
2. 時間駆動分散オブジェクト

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いる．図 1 (a) の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．



(a) Event-Triggered Object Model



(b) Time-Triggered Object Model

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

これに対し，すでに我々が提案している時間駆動オブジェクトモデル[9)10)11)] 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．

自動車制御等の組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ロック図が用いられる．図 2 に制御ブロック図の例を示す．この例は EngineTorque を演算するブロックと ThrottleOpenning を演算するブロックから成り，各々のブロックの演算は制御周期（サンプリング周期）に従って周期的に実行される．

図 3 に，図 2 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．ThrottleOpening の update メソッドは，EngineTorque と EngineTorque が持つ value の値を得て演算を行い，自身の value を更新する．よって各々のオブジェクトは，get メソッドを呼び出すためにオブジェクトの参照を持つ．オブジェクトの update メソッドは制御周期に沿って周期的に実行される．

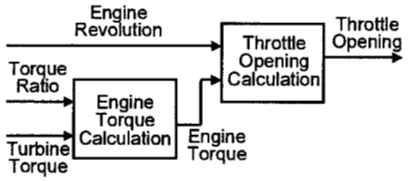


図 2 制御ブロック図の例

本モデルは制御ロジックを制御ブロック図で表す組込み制御システムに適している．制御ブロック図は，フィードバック制御やフィードフォーワード制御等の連続制御で広く使われている．

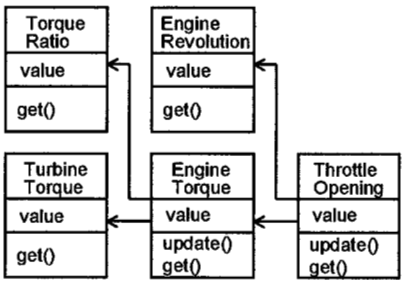


図 3 クラス図の例

2.2 時間駆動分散オブジェクトモデル

我々はレプリカオブジェクトを用いて位置透過性を実現する時間駆動分散オブジェクトを提案する．

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 4 に示す．2 つの ECU があり，図 3 で示したオブジェクトが分散配置されている．他の ECU 上のオブジェクトから参照されるオブジェクトのレプリカを，参照している ECU 上に配置し，それを参照することで位置透過性を実現する．図 4 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 get メソッドを用いて属性 value の値を得る．

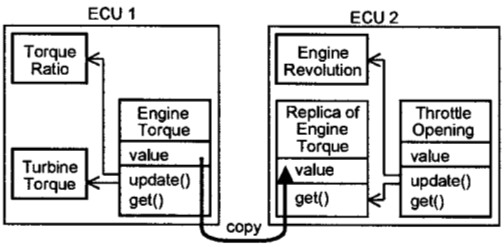


図 4 時間駆動分散オブジェクトモデルの例

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の一貫性を維持するため，オリジナルオブジェ

クトの属性値をレプリカへ同期的にコピーする．これを複製処理という．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，一方向の複製処理で一貫性を保持できる．

また，属性のコピー処理はメッセージキューの必要がない状態メッセージ[12]により効率的に実装できる．以上のように，本モデルではオブジェクト間通信は遠隔メソッド呼び出しを用いず，ローカルメソッド呼び出しのみで実現され，分散オブジェクトとローカルオブジェクトに差がない．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，実行時間のは予測が容易である．複製処理のメッセージは周期的に転送されるため通信遅延時間は予測でき，ネットワークトラフィックの変動も少ない．

RPCに基づく分散処理モデルは，ネストするRPCの実行時間を予測することが困難なため，本モデルには適さない．またRPCは双方向性，同期通信で実装されているため，オーバヘッドが大きくなる．

3. 分散オブジェクト環境

3.1 概　　要

時間駆動分散オブジェクト環境を図5に示す．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，リアルタイムオペレーティングシステム(RTOS(Real-Time Operating System))，CANドライバで構成される．

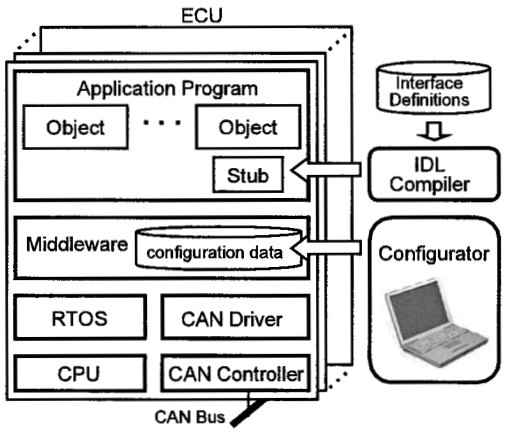


図5　分散オブジェクト環境

各ECUを繋げるためにCANバスを使用する．時間駆動プロトコルは時間駆動分散システムに適している．しかし我々は研究の最初のステップとしてCANを用いる．CANはコスト効率が良いため，自動車制御分野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に，ミドルウェアはすべてのECUの同期をとる機能を持つ．

アプリケーションプログラムは，時間駆動オブジェクトから成る．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．RTOSとしてμITRON[13]を用いる．

ミドルウェアはオリジナルオブジェクトの属性値を周期的にレプリカへコピーする．

スタブは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なう．スタブのインタフェース定義はIDL(Interface Definition Language)により記述される．それをIDLコンパイラが自動的に変換し，スタブのソースコードを生成する．

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．その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って生成される．

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環境を実現する．

複製処理の流れを図6に示す．この例では，EngineTorqueとThrottleOpeningという二つのオブジェクトがある．ECU1にEngineTorqueを配置し，ECU2にEngineTorqueのレプリカとThrottleOpeningを配置する．

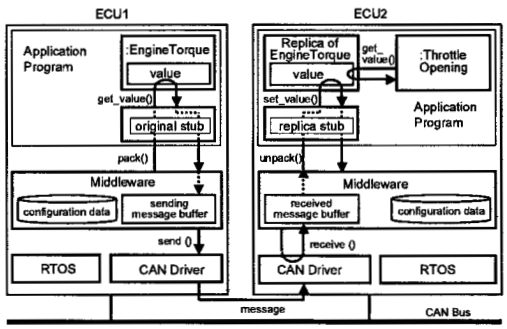


図6　複製処理の流れ

スタブにはオリジナルスタブとレプリカスタブの二種類があり，複製処理で用いられる．オリジナルスタブはオリジナルオブジェクトとミドルウェアを仲介する．そしてレプリカスタブはレプリカとミドルウェアを仲介する．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に配置される．

ミドルウェアは次のように複製処理を実行する．図6より，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．オリジナルスタブは取得した属性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．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（送信要求）を出す．メッセージは8バイト以下（CANメッセージの最大データサイズ）の単位で送信する．送信メッセージバッファのデータが

8バイトを超える場合は，数回に分けて送信する．

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（受信要求）を出す．その後，ミドルウェアはEngineTorqueのレプリカのレプリカスタブを呼ぶ．レプリカスタブは受信メッセージバッファの内容をレプリカへ格納する．

複製処理の周期とメッセージIDが同一の場合，複数のオリジナルオブジェクトの属性値すべてを一つの送信メッセージバッファに格納する．それ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．

時間駆動処理のタイムチャートの例を図7に示す．周期T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する．同じ周期T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．Task1とTask2はミドルウェアの同期機能により同期している．

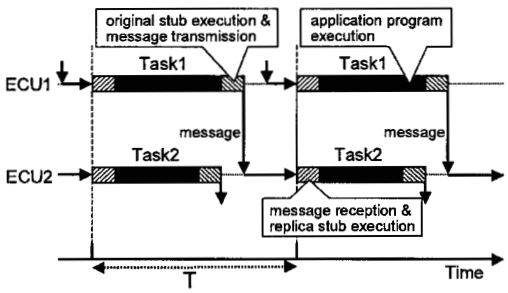


図7 時間駆動処理

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する．そしてCANドライバにメッセージの送信要求を出す．

Task2はアプリケーションプログラムを実行する前に，CANドライバにメッセージの受信要求を出す．その後，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，アプリケーションプログラムを実行する．タスクは周期毎に生成されるため，同じ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．

3.3 開発環境

我々は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コンパイラを開発した．また，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．

分散オブジェクトのインタフェースはIDLで定義し，IDLコンパイラによりスタブを生成する．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インタフェースに限る．

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する．IDLとスタブの詳細は**4.2**で述べる．

コンフィギュレータには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．構成情報には，システムを構築するECUの数，同期用のマスタECUの情報，複製処理の数，複製処理を行うオリジナルオブジェクトクラスの名前やインスタンスオブジェクトの名前，生成するレプリカクラスの名前，複製処理の周期，CANのメッセージのIDが含まれる．

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複製処理を行なう．コンフィギュレーションデータには，複製処理におけるオリジナルスタブやレプリカスタブの呼び出し，送信メッセージバッファや受信メッセージバッファの定義，CANドライバを用いた複製処理のメッセージの送受信，周期タスクの初期化処理等が記述されている．

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込み制御システムを構築できる．

4. 実　　装

4.1 対象言語

ワンチップマイクロコンピュータで構成された組込み制御システムでは，すべてのコードとデータはマイクロコンピュータ上の内部ROMとRAMに格納される．多くの組込み制御システムの実装においては必ずしもオブジェクト指向の機能のすべてが必要とは限らない[14]．メモリサイズの制限があるシステムには，オブジェクト指向プログラムをC言語を用いて実装することが多い．

これより我々は，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，最初のステップとしてC言語に対応したIDLコンパイラを開発した．

C言語で記述したEngineTorqueクラスのソースコードの例を図8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定義される．関数やマクロの第一引数は，インスタンスオブジェクトへのポインタである．

4.2 IDLとスタブ

分散オブジェクトのインタフェースをCORBA準拠のIDLで定義した．我々のIDLはCORBA IDLのサ

```
/* 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 8 C 言語のクラス定義の例

ブセットである．例えば “module” 宣言はサポートしていない．

図 8 で示した EngineTorque クラスのインタフェースの例を図 9 に示す．我々は複製処理の対象となる属性を定義するために，IDL の属性宣言 “attribute” を用いる．つまり，IDL で定義された属性だけがレプリカへコピーされる．図 9 の IDL では，属性 value のみが属性宣言されている．よって属性 limit はコピーされず，属性 value がレプリカへコピーされる．interface 名にはオリジナルオブジェクトのクラス名を用いる．

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

図 9 IDL のインタフェース定義の例

IDL コンパイラはオリジナルスタブ，レプリカスタブ，アクセスメソッドのソースコードファイルを生成する．また要求に応じて，レプリカクラスのソースコードファイルも生成する．これらはクラス単位で生成される．もし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．

レプリカクラスの例を図 10 に示す．EngineTorque クラスのアクセスメソッドの例を図 11 に示す．アクセスメソッドはスタブによって呼び出される．

```
/* EngineTorque.h */
/* Replica Class Definition */
typedef struct {
    short value;
} EngineTorque;
/* macro for access method */
#define EngineTorque_get_value(this)\
                          (this->value)
```

図 10 レプリカクラスの例

```
/* 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 11 アクセスメソッドの例

簡単なオリジナルスタブの例を図 12 に示す．_EngineTorque_pack() は図 6 の pack() と同一である．_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．

```
/* 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 12 オリジナルスタブのコードの例

簡単なレプリカスタブの例を図 13 に示す．_EngineTorque_unpack() は図 6 の unpack() と同一である．_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．

```
/* 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 13 レプリカスタブのコードの例

必要に応じて，ビッグエンディアンアーキテクチャとリトルエンディアンアーキテクチャ間の互換性を保つ機能を備えたスタブを生成できるが，図 12 と図 13 では，

その部分を省略している．

レプリカの有無は，すべてのオブジェクトをECUに配置するときに決定される．よって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される．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．

4.3 実験評価

時間駆動分散オブジェクト環境の実装及び実験を行った．実験では，ハードウェアとしてCANコントローラを内臓したマイクロコントローラ（H8S/2612）を使用した．マイクロコントローラのクロック周波数は20MHz，CANの伝送速度は1Mbpsである．RTOSは使用せず，複製処理のみの時間を測定した．また，ミドルウェアの機能によりCANバスに接続されたECUの同期をとる．

8バイトのデータを複製処理したときの平均時間は176 μ secであった．同一のハードウェアタイマにより測定するため，2つのECU間の往復の複製処理の実行時間を測定し，その1/2を複製処理の時間とした．往復の時間における変動値（最小と最大の値の差）は，2.4 μ secであり，片道の時間の変動値もほぼ同じか少なくなると予測できる．これは衝突が起こらない時の変動値であり，衝突が起こると，変動値はCANにおける1メッセージの通信時間となる．

5. 結　　論

我々は組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を提案し，そのモデルに基づく分散オブジェクト環境を開発した．本環境では効率的な位置透過性を実現するためにレプリカオブジェクトを利用する．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開発した．

今後，時間駆動分散オブジェクト環境の拡張として，時間駆動ネットワークと時間駆動オペレーティングシステム[15]の導入や，C++言語に対応したIDLコンパイラの開発を計画している．

参 考 文 献

1) Kopetz, H. : Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Transaction on Information & Systems*, Vol. E76-D, No. 11, pp. 1325–1332, 1993.
2) Kiencke, U. : Controller Area Network - from Concept to Reality, *Proceedings of 1st International CAN Conference*, pp. 0-11–0-20, 1994.
3) Kopetz, H.and Grunsteidl, G. : TTP-A Protocol for Fault-Tolerant Real-Time Systems, *IEEE Computer* Vol. 27, No. 1, pp.14–23, 1994.
4) Makowitz, R. and Temple, C. : FlexRay - A Communication Network for Automotive Control Systems, *Proceedings of 2006 IEEE International Workshop on Factory Communication Systems*, pp. 207–212, 2006.
5) OMG Technical Document formal/02-06-01 : *The Common Object Request Broker: Architecture and Specification, Version 3.0*, 2002.
6) OMG Technical Document formal/02-08-02 : *Real-Time CORBA Specification, Version 1.1*, 2002.
7) OMG Technical Document formal/02-08-01 : *Minimum CORBA Specification, Version 1.0*, 2002.
8) Lankes, S., Jabs, A. and Bemmerl, T. : Integration of a CAN-based Connection-oriented Communication Model into Real-Time CORBA, *Proceedings of the International Parallel and Distributed Processing Symposium(IPDPS'03)*, p.121a, 2003.
9) 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.
10) 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.
11) 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.
12) Kopetz, H. and Merker, W. : The Architecture of MARS, *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, pp.274-279, 1985.
13) トロン協会　ITRON仕様検討グループ：μITRON4.0仕様，Ver.4.02.00, 2004.
14) 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.
15) OSEK/VDX : Time-Triggered Operating System, Version 1.0, 2001.