分散オブジェクト指向システムにおけるメッセージのトレース

久保田 稔†

分散オブジェクト指向プログラムのトレースを自動的に行う方式について提案する．本方式をメッセージトレースと呼ぶ．これはオブジェクトとメッセージの両者にトレース状態を制御するデータを設け，メッセージ通信時に，オブジェクト間でメッセージのトレース状態を伝播させる機構に基づく．本方式は，多数の処理が同時に行われる分散システムにおいて，特定の処理のみのトレースを行うことを可能にする．これにより，分散オブジェクト指向プログラムの機能確認とデバッグを効率化することができる．また，メッセージ通信の状況を把握できるので，オブジェクトの分散配置の設計のための情報を得るためにも用いることができる．本稿では，本方式の基本概念，実現方式，実現例について述べる．

Message Trace in Distributed Object-oriented Systems

MINORU KUBOTA†

This paper proposes an automatic trace method for distributed object-oriented programs. This method is called *message trace*, and is based on the mechanism that trace control data assigned for both objects and messages are propagated from a sender object to a receiver object in message passing. The message trace is effective for verifying and debugging of distributed object-oriented programs because it allows tracing programs executed for an only specific task while a large number of tasks are executed simultaneously. Moreover, it is useful to gather information for designing the configuration of distributed objects because we can acquire statistical data on message communication by using it. This paper describes basic concepts, implementation, and application of the message trace.

1. はじめに

ネットワークの大規模化，高速化にともない分散処理がより一般的な技術となりつつある．通信分野においても分散処理を活用してネットワークワイドの高度なサービスを提供する方式が導入されている．

このような背景から，我々は並行オブジェクト指向モデルに基づく通信網システムのための分散処理プラットフォーム PLATINA[1]の研究・開発を行ってきた．この上で動作するアプリーケーションは並行動作するオブジェクトで実現され，それらが互いにメッセージを送受することにより処理を進める．

オブジェクトには，自律的に並行動作する能動オブジェクトと，能動オブジェクトの実行環境下で動作する受動オブジェクトの2種類があるが，本稿で単にオブジェクトという場合は能動オブジェクトを意味するものとする．オブジェクトは従来のOSの並列実行単位であるプロセスやタスクに相当する．PLATINAでは，スレッドを用いて実現している．PLATINAは，オブジェクトの実行とオブジェクト間のメッセージ通信を制御する実時間カーネル（以後，単にカーネルと呼ぶ）と，アプリーケーションに共通な拡張サービスを提供するサーバからなる．

各オブジェクトはネットワーク内で一意に定まるオブジェクト識別子を持ち，これを指定することにより，オブジェクトの存在するノードにかかわらずメッセージを送ることができる．メッセージは，オブジェクト間でやりとりされるデータの他に，送信側と受信側のオブジェクトの識別子等の制御データを含む．

分散した多数のオブジェクトがメッセージを送受しながら協調的に処理を進めることで，分散プログラムが実現できる．このような分散プログラムの機能確認やデバッグを行うには，以下の理由により，プログラムの構成要素であるオブジェクト間で送受されるメッセージ通信をトレースすることが有効である．

各オブジェクトは，単体として試験が完了し正常に動作したとしても，他オブジェクトと組み合わせて実

† NTT 光ネットワークシステム研究所
NTT Optical Network Systems Laboratories

行した場合に，オブジェクト間の実行のタイミングや，他オブジェクトの走行の影響により，所定の動作をしないことがある．プログラムが所定の動作をしない場合に不具合のある場所を特定するには，オブジェクトの状態の変化をトレースしなければならない．オブジェクトの状態は主としてメッセージ通信により変化するが，オブジェクトや送受されるメッセージ数が多いと，それらをすべてトレースすることは困難である．したがって，すべての処理ではなく特定の処理に関するメッセージ通信の流れのみを再現することが，分散プログラムの検証やデバッグに有効である．

一方，メッセージ通信をトレースすることにより，対象となるプログラムの実行に影響を与えないことが望ましい．これは，トレースすることが原因となってオブジェクト間の実行のタイミングが変化することにより，不具合の発生状況が変化する可能性があるためである．

本稿では，分散オブジェクト間でのメッセージ通信のトレースを行う方式について提案する．同時に複数の処理が実行される環境において，特定の処理のトレースのみをオブジェクトを変更することなく容易に行うことができるので，本方式は多数の分散オブジェクトからなるプログラムの機能確認とデバッグに有効である．また，メッセージ通信の状況を把握できるため，オブジェクトの分散配置の設計のための情報を得るためにも用いることができる．

2. 分散プログラムのトレース

2.1 オブジェクト実行のトレース

試験担当者等によりあらかじめ指定されたプログラム（命令のアドレス）をオブジェクトが実行した際に，その時点であらかじめ指定されたデータを収集することをオブジェクト実行のトレースと呼ぶ．トレースを行う時点を**トレース契機**，トレースにより収集されるデータを**トレースデータ**，またトレース契機とそれに対応するトレースデータの種別の集合を**トレース制御情報**と呼ぶ．実行がトレースされるオブジェクト（処理）を**トレース対象オブジェクト（処理）**と呼ぶ．

トレース契機としてはプログラム中の任意の時点を指定することが可能であるが，本稿で提案する方式は，オブジェクトがメッセージを受信した時点をトレース契機とする．オブジェクトの単体の試験が完了していれば，メッセージ受信時にのみトレースデータを収集しても，分散プログラムの処理の流れが把握でき，有効な情報が得られる．

試験担当者等がトレース制御情報をカーネルに通知する．カーネルは，オブジェクトがメッセージを受信すると，3.2 節と 3.3 節で述べるようにトレースすべきかどうかを判断し，必要な場合にのみトレースデータを収集する．収集するトレースデータには，通信種別，送信/受信オブジェクトの識別子，メッセージの種別/内容等がある．

2.2 分散プログラムのトレース

逐次的なプログラムにおいてはプログラムのトレースは一般的な技術である．しかし多数の分散オブジェクトのトレースを行う場合には後述するような問題があり，分散プログラムに対応したトレース技術が必要となる．

まず分散プログラムにおけるトレースを説明する．**図 1** に示すように 3 つのノード，X，Y，Z において 3 つのアプリーケーションの処理 P1，P2，P3 が並行して処理されているものとする．各アプリーケーションは分散オブジェクトによって実現される．**図 2** は収集対象となるトレースデータを示したもので，`Tn.p` はオブジェクト o において，処理 p にかかわって収集の対象となるトレースデータを表す．下線がついた `Tn.p` が実際に収集されるトレースデータを表す．

すべてのノードにおいてすべての処理のトレースを行うことは，すべてのアプリケーションの処理に関するトレースデータが混在して収集され（図 2 (a)），トレースデータが膨大になり現実的でない．たとえば，本稿で提案する手法の適用先の 1 つである通信網システムでは，同時に数千の処理が実行される．また，特定のノード（ここでは X とする）に着目して，そのノードのすべてのプログラムの実行をトレースすると，他のノードのトレースデータが収集できず，またすべ

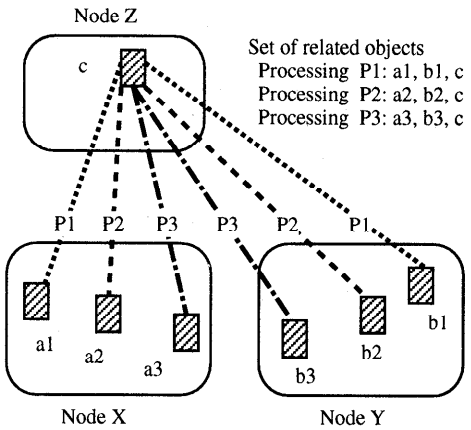


図 1 分散オブジェクト実行のトレース
Fig. 1 Trace of distributed object execution.

Tn.p represents trace data of an object n for a processing p.
Underlined Tn.p is actually gathered.

(a) Trace all processing.
Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta3.3 Ta1.1 Tb3.3 Tb2.2 Tc.2 Tc.3 Ta2.2 Tb2.2 Tc.1 Ta1.1

(b) Trace only in a node.
Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta3.3 Ta1.1 Tb3.3 Tb2.2 Tc.2 Tc.3 Ta2.2 Tb2.2 Tc.1 Ta1.1

(c) Trace target is an object a1.
Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta3.3 Ta1.1 Tb3.3 Tb2.2 Tc.2 Tc.3 Ta2.2 Tb2.2 Tc.1 Ta1.1
Tb1.1, Tc.1 are not gathered.

(d) Trace targets are objects a1, b1 and c, which provide processing P1.
Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta3.3 Ta1.1 Tb3.3 Tb2.2 Tc.2 Tc.3 Ta2.2 Tb2.2 Tc.1 Ta1.1
Tc.2 and Tc.3 are unnecessary information.

(e) Proposed method.
Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta3.3 Ta1.1 Tb3.3 Tb2.2 Tc.2 Tc.3 Ta2.2 Tb2.2 Tc.1 Ta1.1

図2 トレースデータ
Fig. 2 Trace data.

てのアプリケーションに関するトレースデータが混在して出力される（図2(b)）．

同時に実行される処理の数が多い環境で特定の処理の機能確認やデバッグを行う場合，不要なトレースデータもあわせて収集されると，必要なトレースデータのみを取り出すことが困難になる．したがって機能確認やデバッグの対象となる特定の処理（トレース対象処理）が実行された場合にのみトレースデータを収集する方式が有効である．

2.2.1 オブジェクトに着目したトレース

トレース対象処理のみに着目したプログラムのトレース方法として，オブジェクト（タスク）ごとにトレースを行う方法がある．これを**オブジェクトトレース**あるいは**タスクトレース**[2]と呼ぶ．たとえば図1においてオブジェクトa1をトレース対象とすると，a1が実行した処理に関するトレースデータのみが出力される．しかし処理P1に関して，オブジェクトb1，cはトレース対象となっていないので，これらに関するトレースデータは収集されない（図2(c)）．

2.2.2 関連オブジェクト集合

トレース対象処理が複数のオブジェクトにより行われる場合，これらのオブジェクトの集合を**関連オブジェクト集合（related object set）**と呼ぶ．たとえば処理P1の関連オブジェクト集合は，a1，b1，cである．

オブジェクトトレースを用いてトレース対象処理をすべてトレースする場合，関連オブジェクト集合に属するすべてのオブジェクトをトレース対象としなければならない．分散システムにおいては，オブジェクト数が多く，またそれらが複数のノードに分散していることから，関連オブジェクト集合を決定すること，およびトレースを行う前にこれらをトレース対象として指定することは困難である．

2.2.3 サーバオブジェクトのトレース

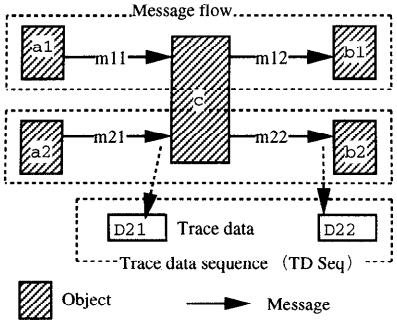
オブジェクトトレースでは，関連オブジェクト集合をすべてトレース対象とすることができても以下のような問題が残る．すなわち複数のオブジェクトからの処理要求を受けるオブジェクト（サーバオブジェクトと呼ぶ）では，トレース対象でない処理のトレースデータも収集されてしまう．たとえば図1においてトレース対象処理がP1でも，オブジェクトcは，P1，P2，P3すべての処理を行うため，オブジェクトcの実行にともなって，P2，P3に関するトレースデータも収集される（図2(d)）．

サーバオブジェクトにおいてトレース対象処理のみトレースすることは，実行中の処理がトレース対象かどうかの判定処理が必要となるが，オブジェクトトレースだけでこれを行うことはできない．アプリケーションにこれらの判定処理を組み込めば可能となるが，運用時には不要となる試験やデバッグのための処理をアプリケーションに記述することは望ましくない．

2.2.4 メッセージトレース

オブジェクトトレースは，分散プログラムのトレースに適用するには前述のような問題点がある．分散システムにおけるトレースでは，図2(e)に示すように，トレース対象処理に関するトレースデータのみが収集できることが望ましい．

本稿では以下の要求条件を満たすトレース方法（これを**メッセージトレース**と呼ぶ）について提案する．



Message sequence: (m11,m12), (m21,m22)
Object sequence: (a1, c, b1), (a2, c, b2)

図3 メッセージフローとトレースデータシーケンス
Fig. 3 Message flow and trace data sequence.

(1) 複数のノードからなる分散システムにおいて，複数の処理に関し，それぞれの処理を行うために実行される複数のオブジェクトの中で，特定の処理（トレース対象処理）を行うもののみトレースを行う．

(2) トレース対象処理の関連オブジェクト集合に属するオブジェクトが自動的にトレース対象となり，複数の処理にかかわる（複数の関連オブジェクト集合に属する）オブジェクトにおいては，トレース対象処理にかかわる場合にのみ，トレースを行う．

(3) (1)，(2)の実現に際して，アプリケーションを変更せずにカーネルだけで対処する．

3. 分散オブジェクト間メッセージトレース

3.1 メッセージフロー

ある処理を実現するために実行されるオブジェクトとそれらが送受するメッセージの集合を，その処理に関する**メッセージフロー**と呼ぶ（**図3**）．メッセージフローの中のオブジェクトの集合が前述の関連オブジェクト集合となる．またトレース対象処理に対応するメッセージフローの最初のオブジェクトを特に**トレース開始オブジェクト**と呼ぶ．メッセージフローの中のメッセージの並びを**メッセージシーケンス**，オブジェクトの並びを**オブジェクトシーケンス**と呼ぶ．

3.2 オブジェクトのトレース制御

2.2.4 項の要求条件を満たすトレースを行うため，カーネルのオブジェクトの実行制御に以下の機能を追加する．まずトレースを行うかどうかを示すデータ（**トレース状態表示**と呼ぶ）をオブジェクトに付加する．これは各オブジェクトの実行制御用データに格納されるのでアプリケーションには影響がない．トレース状態表示が有効なオブジェクトがトレース対象オブジェクトとなり，このオブジェクトが実行されるときにのみトレースデータを収集する．

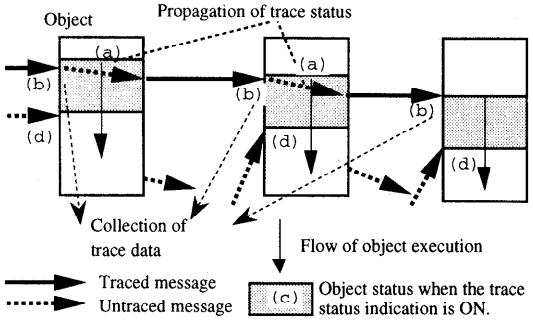


図4 トレース状態指示の伝搬
Fig. 4 Propagation of trace status.

3.3 トレース状態の伝播

前述のように，トレース対象処理の関連オブジェクト集合を事前に求め，これに属するオブジェクトのトレース状態表示を有効にしておくことは分散システムにおいては困難である．トレース対象の関連オブジェクト集合を自動的にトレース対象とするため，次の方法を用いる．

オブジェクトだけでなくメッセージにもトレース状態表示を付加し，同一メッセージフローに属するオブジェクトとメッセージの間でトレース状態表示を引き継いでいく機構を提供する（**図4**(a)）．これを**トレース状態の伝播**と呼ぶ．メッセージのトレース状態表示はメッセージの制御データエリアに格納される．

トレース状態表示が有効なメッセージを**トレース対象メッセージ**と呼ぶ．オブジェクトが受信したメッセージがトレース対象メッセージであれば，カーネルはトレースデータを収集する（図4(b)）．

トレース状態の伝播を実現するため，カーネルに以下の機能を追加する．

(1) トレース対象オブジェクトから送られるメッセージを自動的にトレース対象とする（メッセージのトレース状態を有効にする）．

(2) オブジェクトが受信したメッセージがトレース対象かどうかを調べ，トレース状態ならばオブジェクトのトレース状態表示を一時的に有効にする（図4(c)）．トレース対象でなければオブジェクトのトレース状態表示は変化しない．別のメッセージフローに属する次のメッセージがくるまで（図4(d)），このオブジェクトから送られるメッセージのトレース状態表示は有効となり，トレース状態表示が伝播される．オブジェ

クトが次のメッセージを受信すると，トレース対象メッセージ受信前の状態にオブジェクトのトレース状態表示を戻し，本処理を繰り返す．

以上によりオブジェクトが受信したメッセージのトレース状態表示が同一メッセージフローに含まれる次のメッセージに引き継がれる．試験担当者等がトレース開始オブジェクトのトレース状態表示を有効にすることで，同一メッセージフロー中のすべてのオブジェクト，メッセージのトレース状態表示がカーネルにより自動的に有効となる．説明を簡単にするため，オブジェクトのトレース状態表示は有効か無効の状態を持つとした．しかしトレース開始オブジェクトのトレース状態表示を有効にしても，それがメッセージを送る前に他のオブジェクトからトレース状態でないメッセージを受信すると，トレース状態表示が無効になってしまうという問題が生じる．同様の問題はメッセージフロー中の任意のオブジェクトについても生じるが，これについては3.6節で詳しく述べる．

ここでは，トレース開始オブジェクトに関する対処法を述べる．これは，トレース状態表示を拡張し，オブジェクトのトレース状態として，非トレース状態，初期トレース状態，一時トレース状態を設けるものである．非トレース状態と一時トレース状態は，それぞれ前述のトレース状態表示が無効と有効の状態である．

初期トレース状態はトレース開始オブジェクトの初期状態である．この状態にあるオブジェクトが送るメッセージは自動的にトレース状態になる．メッセージ受信によりトレース状態は変化しないが，メッセージを送出すると非トレース状態になる．

3.4 トレースデータの収集

あるトレース対象処理の実行をトレースすることにより生成されるトレースデータの並びを**トレースデータシーケンス（TD Seq）**と呼ぶ（図3）．1つのトレース対象処理に1つのTD Seqが対応する．図1におけるTD Seqは次のようになる．

処理P1: `Ta1.1 Tb1.1 Tc.1 Ta1.1 Tc.1 Ta1.1`

処理P2: `Ta2.2 Tb2.2 Tb2.2 Tc.2 Ta2.2 Tb2.2`

処理P3: `Ta3.3 Tb3.3 Tc.3`

トレース対象のメッセージフローの内容を表示するプログラムを**メッセージビューア**と呼ぶ．メッセージビューアは，複数のノードから送られるトレースデータを分析して，トレース対象のメッセージフローを再現する．これによりプログラムの機能確認と試験の容易化がはかれる

図5に示すように，トレース対象メッセージが受信されると，カーネルがトレースデータを収集し，メッセージビューアにメッセージとして送る．このメッセージを**トレースデータメッセージ（TD メッセージ）**と呼ぶ．TD メッセージは，トレース対象メッセージの内容と，TD Seqを識別するための情報を含む．

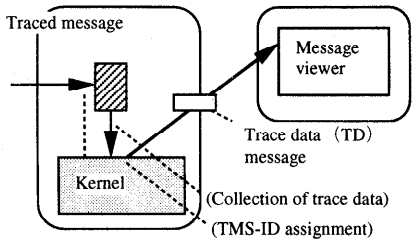


図5 トレースデータの収集
Fig. 5 Collection of trace data.

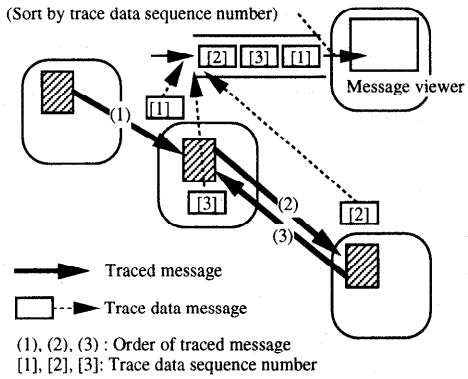


図6 トレースデータメッセージ
Fig. 6 Trace data message.

3.5 メッセージフローの識別

3.5.1 トレースシーケンス番号

メッセージビューアは保守・運用を目的とする特定のノードに配置されることを想定している．複数のノードのカーネルから並行してTD メッセージがメッセージビューアに送られるが，分散システムでは，**図6**に示すように，トレースデータの生成順序と，これらがメッセージビューアに到着する順序が一致する保証はない．メッセージビューアでは，1つのトレース対象処理に関するトレースデータをその発生順序に対応して並べ替える必要がある

発生順序に対応してトレースデータを並べ替えるために，カーネル内のメッセージトレース機構は，オブジェクトとメッセージの制御データに**トレースデータシーケンス番号（TD Seq 番号）**を付与する（図6）．以下に述べる方式により，トレース状態が次に引き継がれるときに，メッセージのTD Seq番号は1増やされる．

(1) オブジェクトが生成されたとき，あるいはオブジェクトのトレース状態が有効から無効に変化したとき，オブジェクト制御データのTD Seq番号を初期値（通常は0）に設定する．
(2) 受信したメッセージのトレース状態が有効ならば，カーネルは，メッセージのTD Seq番号（有効TD Seq番号と呼ぶ）をオブジェクトの制御データに格納し，オブジェクトのトレース状態表示を有効にする．
(3) この状態でメッセージを送信するとき，有効TD Seq番号に1を追加した値をこのメッセージのTD Seq番号として設定する．

TD Seq番号は，TDメッセージに含めてメッセージビューアに送られ，メッセージビューアはTDメッセージをTD Seq番号の順番で並び替えてTD Seqを復元する．

3.5.2 メッセージシーケンス識別情報

トレース対象処理が1つだけのとき，すなわちTD Seqが1つだけのとき，TDメッセージを上記TD Seq番号順に並べ替えて表示すればよい．トレース対象処理が同時に複数存在するとき，TDメッセージをトレース対象処理ごとに分類する必要がある．たとえば図1において，処理P1とP2をトレース対象とした場合，TD Seq (`Ta1.1 Ta2.2 Tb1.1 Tc.1 Tb2.2 Ta1.1 Tb2.2 Tc.2 Ta2.2 Tb2.2 Tc.1 Ta1.1`) から (`Ta1.1 Tb1.1 Tc.1 Ta1.1 Tc.1 Ta1.1`) および (`Ta2.2 Tb2.2 Tb2.2 Tc.2 Ta2.2 Tb2.2`) を分離する必要がある．

この分類を行うためには，TDメッセージごとにトレース対象処理を識別するための情報を付加しておく必要がある．この情報を**トレースメッセージシーケンス識別情報（TMS-ID）**と呼ぶ．TMS-IDは，TDメッセージに付与されてメッセージビューアに送られる．

TMS-IDは分散システム内で一意に識別できなければならない（重複があってはいけない）．分散システム内で一意にTMS-IDを与える方法としては，以下のような方法がある[3]．

(1) トレース対象処理対応にTMS-IDを1つだけ与える方法（固定付与方式）．TMS-IDをトレース対象メッセージに含め，トレース伝播機構を用いて，次のメッセージに引き継いでいかなければならない．
(2) 各ノードでTDメッセージごとに付与し，全TDメッセージ内のTMS-IDを解析することにより，TDメッセージが対応するトレース対象処理を識別する方法（個別付与方式）．

後述する試作システムでは，実装上容易な固定付与方式をとっている．

3.6 トレース対象処理とメッセージフローの一致性

トレース状態伝搬機構によりトレース状態表示が有効なオブジェクトとメッセージからなるメッセージフローを特に**トレースメッセージフロー**と呼ぶことにする．メッセージトレース方式の主要な機構であるトレース状態の伝搬はカーネルが行う．カーネルは複数の処理の中から特定の処理を識別して，トレース対象かどうかの判断を行うことはできない．

したがって，トレース対象処理と，トレースメッセージフローの不一致が生じる場合がある．トレース対象処理の内容を考慮せずに，トレース対象処理とトレースメッセージフローを完全に一致させることは困難であるため，ここではどのような場合に不一致が生じるか，またそれへの対処の可能性，について議論する．

3.6.1 枝分かれ問題

オブジェクトがあるメッセージに起因してトレース状態になり，その状態でメッセージが2つ以上出た場合をメッセージフローの**枝分かれ**と呼ぶ．枝分かれが生じる場合としてはマルチキャストの場合がある．

トレース対象処理の途中でトレース対象オブジェクトがマルチキャストを行うと，トレース対象メッセージが非常に増える可能性がある．たとえばマルチキャストメッセージを送った先で，さらにマルチキャストを行う処理が続く場合である．本稿で対象とするメッセージはプログラムの実行を制御するものであるので，そのような処理はほとんどないと考え，メッセージが爆発的に増えるおそれはないものとする．

この場合は，マルチキャストされたメッセージに同一のTMS-IDを付与する必要がある．前述の固定付与方式の場合は問題ないが，個別付与方式では，識別できないおそれがある．検証，デバッグの対象となるトレース対象メッセージフローが同時に存在する数は少ないので，実用的にはTMS-IDを乱数で与える固定付与方式で十分であろう．

ただしメッセージビューアで表示する際，メッセージの並べ替えの工夫が必要である．枝分かれがない場合は，TD Seq番号で並べ替えることができる．枝分かれがあるトレースメッセージフローが，一般的な有効グラフとなる場合について議論するのは困難であるので，ここでは木構造の場合について考察する．実際には木構造でほとんどのメッセージフローがカバーできると考えられる．

簡単のため，**図7**を用いて，メッセージフロー中の1つのオブジェクト(a)で2つに枝分かれするものと

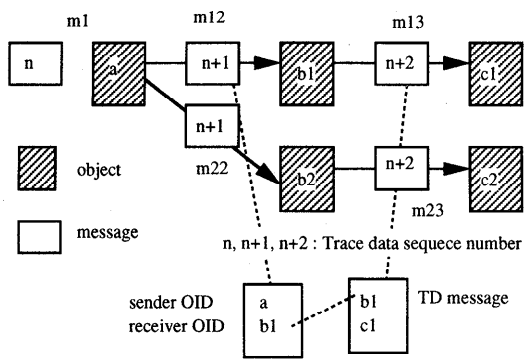


図7 分岐メッセージのトレース
Fig. 7 Multicast mesage trace.

する．メッセージm1を受信して，トレース状態のままaが2つのメッセージを送るものとする．このとき，メッセージフローは，M1（m12, m13）とM2（m22, m23）の2つに分かれる．もとのトレースメッセージフロー（..., m1）をMとする．M，M1，M2を1つのメッセージフローとして認識する方法として，以下の2つがある．

(1) M1とM2にMと同一のTMS-IDを付与．メッセージフローを再構築するためには，まず，同一TMS-IDのTDメッセージを解析し，TD Seq番号の順に並べ直す．次に，TD Seq番号が隣り合う2つのTDメッセージに含まれる，トレース対象メッセージの送信オブジェクト識別子と受信オブジェクト識別子を用いて，メッセージの送受関係を認識する．すなわちあるTDメッセージ（TD Seq番号をnとする）の送信オブジェクト識別子と次のTD Seq番号（n+1）を持つTDメッセージの受信オブジェクト識別子が一致するものが直接メッセージを送受したオブジェクトとして，隣接して並べる．これによりメッセージフローを再構築できるが，M1とM2の時間順序関係を再現することはできない．メッセージビューアでの処理が増える．

(2) M1とM2にMとは異なるTMS-IDを付与．この場合，M1，M2のTMS-IDおよびそれらがMから派生したものであることを，カーネルはメッセージビューアに通知する必要がある．ただし，m12とm22の時間順序関係を再現することはできるが，それ以降のM1とM2のメッセージ間では時間順序関係を再現することはできない．メッセージビューアでの処理は，(1)に比べて簡単になる．

本稿で対象とするような通信網システムでは，カーネルの処理のオーバヘッドが少ない方式(1)が望ましい．

3.6.2 処理の紛れ込み

2.2.3項で述べたようにサーバオブジェクトではトレースメッセージフローとトレース対象処理が一致しない場合がある．サーバオブジェクトに処理要求を出すクライアントオブジェクトAがサーバオブジェクトSに要求メッセージm1を送る．Aがトレース対象であれば，m1もトレース対象となり，Sがm1を受信した時点で一時トレース状態となる．

このときSが他のクライアントオブジェクトからの要求を受け付ける前にm1に基づく処理を完了し，その処理結果m2を返せば，m2もトレース対象となり，トレース対象処理とトレースメッセージフローは一致する．サーバオブジェクトのトレース状態表示は，m1受信前の状態に復帰する．

しかし，m2を返す前に，別の要求メッセージm3を受信したとすると，Sのトレース状態表示は，m1受信前の状態（s1）に復帰する．s1がトレース対象でないとすると，m2を返送する際にSのトレース状態表示が有効でないので，m2のトレース状態表示を有効にすることができない．したがってm2はトレース対象とすべきであるにもかかわらず，トレース対象とならない可能性がある．

PLATINAでは，メッセージは，オブジェクトとは独立の領域から確保されたメッセージバッファに格納されて送られる[4]．これを利用して，この問題に対して次のように対処する．すなわち，サーバオブジェクトにおいて，処理要求を運んできたメッセージバッファはただちに解放するのではなく，保留したままにしておき，このときメッセージのトレース状態表示は変更しない．

これまでは，オブジェクトのトレース状態表示が無効ならば，その際に送出されるメッセージのトレース状態表示も無効にするとしたが，メッセージ送信時に，オブジェクトのトレース状態表示が有効でなければメッセージのトレース状態表示を変更しないものとする．これにより，サーバオブジェクトは受信した要求メッセージと同じメッセージバッファを応答メッセージに用いることで，要求メッセージと応答メッセージの間でトレース状態を引き継ぐことができる．また，メッセージバッファ捕捉時に初期設定を行い，トレース状態表示を無効にしておくことで，誤ってメッセージをトレース状態にすることは防止できる．

要求メッセージと応答メッセージで異なるメッセージバッファを用いる必要がある場合（たとえば要求メッ

セージバッファの大きさが，応答メッセージを送るために十分な大きさを持たない場合）は，要求メッセージのトレース状態表示を応答メッセージに引き継ぐ処理を，サーバのプログラムとして陽に記述する必要がある．

4. 試作と評価

4.1 メッセージトレースの試作

PLATINAの研究・開発の一環として，分散処理環境を模擬する環境（分散処理実験システムと呼ぶ）をSunワークステーション上に構築した[5]．本システムは，ノードを1つ以上のUnixプロセスで模擬し，オブジェクトをSunOSのLight Weight Process機構を用いて模擬している．

Unixプロセスとして実行するアプリーケーションプログラムには，PLATINAカーネルを模擬する擬似カーネルのプログラムをリンクする．擬似カーネル内にメッセージトレースの機能を組み込んでいる．またメッセージビューアを分散処理実験システムの一部として実現し，Xウィンドウシステムを用いて，グラフィカルな表示を行っている．

また実PLATINAカーネルもメッセージトレースの機能を持ち，実PLATINAシステムと実験システム上のオブジェクトは互いに通信できる．このため，実PLATINAシステムあるいは実験システム上のみだけでなく，実験システムの仮想ノード上のプログラムと，実PLATINAシステムのプログラムの間で送受されるメッセージのトレースも可能である．

メッセージビューアは，実/仮想ノードから送られるトレース情報を収集し，メッセージフローを再構築し，送られるトレースデータを用いてXウィンドウシステムを用いて図形表現にして表示する[6]．**図8**に表示例を示す．メッセージビューアの主な機能を以下に示す．

トレースされたメッセージフローが複数ある場合があるので，これらのうちの1つを選択し，そのシーケンスを再現表示する．各メッセージの内容をシンボリックに表示することができる．また再構築したメッセージシーケンスをファイルに格納する機能を持つ．

メッセージの流れを把握しやすくするため，表示するメッセージフローの範囲を静的あるいは動的に変化させることができる．たとえば，任意のメッセージを直接指定することにより特定メッセージのみの表示の指定またオブジェクト識別子の指定により，該オブジェクトに送受されるメッセージのみの表示の指定等が可能である．

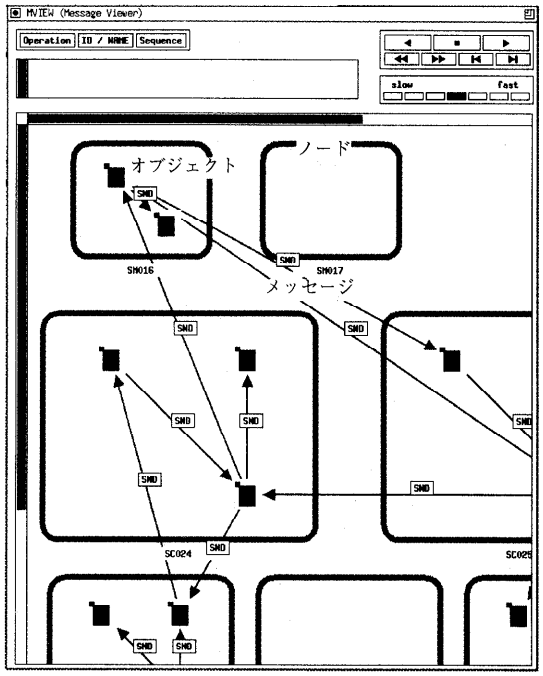


図8 メッセージの表示例
Fig. 8 Message viewer.

メッセージトレースを行うためには，処理の初めのオブジェクトあるいはメッセージをトレース状態としなければならない．前述の分散処理実験システムでは，オブジェクトの状態の表示と制御が可能な仮想端末を用いて，オペレータがトレース開始オブジェクトのトレース状態を有効にする．

4.2 評　　価

4.2.1 機能確認とデバッグの容易化

メッセージトレースは，複数の処理が同時に行われる分散システムにおいて，特定の処理のみのトレースを行うことを可能にする．また，本方式ではソースプログラムを修正する必要がない．したがって，1つの通信サービスが，異なるベンダーで開発された複数のプログラムにより提供される場合でも，トレースを行うことができる．

メッセージトレースを行うためにはカーネルにトレース機構を組み込む必要がある．本機構は，受信メッセージのチェックと必要な場合にのみ受信メッセージの情報を保存するだけの単純なものであるので，組み込むことは容易である．また4.2.2項で述べるように，システムの性能を大きく劣化させることもない．今後，分散プログラムが協調して様々なサービスを提供する場合が多くなると予想されるが，この場合にこれらのプログラムを同時に実行しても着目した処理（サービ

ス）に関する情報のみを自動的に収集できる本方式は分散プログラムの機能確認とデバッグにきわめて有効である．

本方式は，3.4 節で述べたように自動的にまた本来の処理を遅延させることなくトレースデータを収集できるため，トレース対象プログラムにおいて，時間に依存する処理への影響がほとんどない．このため，従来のトレース対象プログラムの処理を一時中断させる方式に比べて，処理時間に依存する誤りを検出する確率が高く，デバッグの効率化に寄与する．

本方式を活用すると，新たに測定用のツールを組み込むことなく，メッセージ通信の状況を把握できる．メッセージビューアは，メッセージフローのデータをファイルに格納できるので，これを用いてメッセージフローの分析を行うことができる．今後，分散配置されたオブジェクトにより通信サービスが提供される場合が多くなると予想されるが，その際，性能向上を図るためには，オブジェクトを適切なノードに配置する必要がある．本方式により，オブジェクトの分散配置の設計の際に有効な情報を得ることが容易になる．

4.2.2 カーネルのオーバヘッド

メッセージトレースの機構はカーネルに組み込まれる．したがって実行時のオーバヘッドが増大するが，以下に述べるようにその影響は小さく，実用上問題ない．メッセージあるいはオブジェクトのトレース状態の検査はメッセージバッファあるいはオブジェクトの実行制御用データのデータを読み出して値を調べるだけなので，命令数は 10 ステップ以下である．これはメッセージ送信，受信全体の処理の 10%以下である．メッセージ通信以外の他のカーネル機能には，メッセージトレース機構組み込みの影響はないため，カーネル全体のオーバヘッドは数%以下である．また処理全体に占めるカーネルの処理の割合はたかだか数 10%と予想されるので，システム全体の処理の中でメッセージトレース機構組込みによるオーバヘッドの増大は 1%以下になり実用的には無視できる．

分散システムにおいて多数ある処理の中でトレース対象処理は 1 つか，複数あっても少数である．カーネルがトレースデータを収集するのは，トレース対象メッセージが到着した場合のみであり，その確率は少ない．たとえば本方式の適用対象である通信網システムにおいては，同時実行される処理は数千に及ぶため，その確率は 1%以下である．したがって収集すべきトレースデータの量は，すべての処理をトレースする場合に比べてきわめて少なくなる．

また，3.4 節で述べたようにトレースデータの解析は別ノードで実行されることを想定しているので，TD メッセージ送信は，トレースデータ収集と同期して行う必要はない．いったんメモリに格納しておき，行うべき処理がなくなったときに，カーネルが TD メッセージの送信を行えばよく，アプリケーションの処理に与える影響は小さい．TD メッセージの解析はカーネルとは別のメッセージビューアで行われるため，トレース処理のためのオーバヘッドの増加は少ない．

分散システムの運用時にメッセージトレース機構によるオーバヘッドも許容できないような厳しい条件がある場合でも，以下のようにメッセージトレース機構の着脱は容易に行える．カーネルの機構の中で置換えが必要なのはメッセージ送信と受信部だけであり，メッセージトレース機構を持つメッセージ送受信プログラムと持たないメッセージ送受信プログラムをカーネルに組み込んでおき，トラップベクタの置換え等により両者を切り換える．

5. 関連研究

メッセージパッシングの並行プログラムを再演させる機構の提案がある[7]．これはメッセージの送出順序だけを記録し，メッセージの内容は再演により再生されるものである．すべてのメッセージに対して何らかのトレース情報を出さなければならないので，トレース情報の記録に大量の記憶装置が必要となる．本手法では，特定の処理に関する情報のみを収集するので，トレース情報の記録のための記憶装置は少なくてよい．

並列処理プログラム，分散プログラムをモニタする手法は多く提案されている[8)~10)]．これらの主要な技術はトレースデータの量を削減するものであり，本手法と目的は同じであるが，処理の数に比例してトレースデータの数も増大する．これに対して本手法は特定の処理に関してのみトレースデータを収集するので，他の処理が増加することの影響を受けない．たとえば本方式の適用対象である通信網システムにおいては，同時に実行される処理は数千に及ぶため，既存方式に比べて収集すべきトレースデータの数は 1%以下となる．

また本手法に類似する手法として，プロセス内部で起こる様々な事象やプロセス相互の生起を，まとめてイベントとしてとらえ，イベント間の時間順序や各種の依存関係を表示したり，解析機能を提供する手法がある[11),12)]．これらはプログラム全体の実行に着目しており，本稿で述べたように，複数の処理の中から特定のものを抽出することは考慮されていない．

様々な分散アプリケーションに対応できるデバッガ

も提案されているが[13),14)]，これらは，アプリケーションごとに表示の方法を変更するものである．複数のアプリケーションの中から選択的にトレース情報を収集する手法については考慮されていない．

プログラムの実行の視覚化に関して，様々なレベルでの視覚化機能を提供し，またブレークポイントもユーザ定義可能とすることで，デバッグ対象範囲を限定していく手法が提案されている[15),16)]．これらはプログラムのどこが実行されるか把握している場合には非常に有効な方法と考えられるが，本稿で対象としている通信網システムのようにプログラムが広域に分散し，どのプログラム（オブジェクト）が実行されるかが事前に把握できないような場合には適用が困難である．

オブジェクトの動的振舞いを可視化する手法が提案されている[14)]．プラットフォーム（カーネル）と言語に依存しない方法であるため，この適用範囲は広いが，プログラム中に実行経過を出力するためのプログラムを埋め込む必要がある．すでに運用中のプログラムを試験対象とすることはできない．本稿で提案した手法は，実行経過の収集はプラットフォームが自動的に行う．

本稿で提案した手法は，以下の特徴を持つ．分散システムで多数の処理が同時に実行されている環境で，特定の処理に関するプログラムの実行のみをトレースし，その結果を再演する．またカーネルのサポートにより，実時間性を損ねないという観点からプログラムの実行への影響を与えずに，特定の処理の識別を自動的に行う．

6. まとめ

オブジェクトとメッセージの両者にトレース状態を制御するデータを設け，またメッセージのトレース状態を伝播させることにより，分散プログラムのトレースを自動的に行う方式，およびトレースデータに基づいてメッセージ通信を再現する手法を提案し，その実現手法についても述べた．本方式は，複数の処理が同時に行われる分散システムにおいて，特定の処理のみのトレースを行うことを可能にするため，分散プログラムの検証，デバッグに有効である．また，メッセージ通信の状況を把握できるため，オブジェクトの分散配置の設計のための情報を得るためにも用いることができる．本方式は分散処理実験システムとして試作し，その効果を確認した．

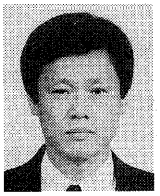
謝辞 本研究に関し，有益なご意見，コメントをいただいたNTTネットワークサービスシステム研究所丸山勝己主席研究員（現国立国文学研究資料館），NTT光ネットワークシステム研究所山田茂樹主幹研究員，田中聡主任研究員，松尾真人主任研究員，中村元紀研究主任，武本充治社員，およびプロトタイプシステム開発にご協力いただいた（株）SRA林幸弘氏，岡村保男氏に感謝いたします．

参考文献

1) 久保田 稔，丸山勝己：通信網のための分散処理プラットフォーム，電子情報通信学会論文誌，Vol.J79-B-I, No.5, pp.301–309 (1996).
2) 渡辺次郎，城 正彦，五十嵐 譲，石橋奈津子，松下政好：マルチタスク交換向きCTRON用並行タスクデバッガ，電子情報通信学会技術研究報告，SSE88-161, pp.1–6 (1988).
3) 久保田 稔，金 東輝：オブジェクト指向分散プログラムのトレース，情報処理学会研究報告システムソフトウェアとオペレーティングシステム，63–3 (1994).
4) 久保田 稔，丸山勝己，田中 聡：通信網ソフトウェア分散処理プラットフォームのカーネル，情報処理学会論文誌，Vol.35, No.12, pp.2602–2612 (1994).
5) Kubota, M., Maruyama, K., Tanaka, S. and Nakamura, M.: Integrated Environment for Distributed Telecommunication Software Development and Operation, *ISS '95*, Vol.2, pp.163–167 (1995).
6) 久保田 稔，岡村保男，中村元紀：オブジェクト間通信においてトレースされたメッセージの表示手法，1996電子情報通信学会総合大会，D-98 (1996).
7) LeBlanc, T.H. and Mellor-Crummey, J.M.: Debugging Parallel Programs with Instant Replay, *IEEE Trans. Comput.*, Vol.C-36, No.14, pp.471–482 (1987).
8) Joyce, J., Lomow, G., Slind, K. and Unger, B.: Monitoring Distributed Systems, *ACM Trans. Computer Systems*, Vol.5, No.2, pp.121–150 (1987).
9) Dodd, P.S. and Ravishankar, C.: Monitoring and Debugging Distributed Real-time Programs, *Software–Practice and Experience*, Vol. 22, No.10, pp.863–877 (1992).
10) Netzer, R.H.B., Subramanian, S. and Xu, J.: Critical-Path-Based Message Logging for Incremental Replay of Message-Passing Programs, *The 14th International Conference on Distributed Computing Systems*, pp.404–337 (1994).
11) LeBlanc, R.J. and Robbins, A.D.: Event-Driven Monitoring of Distributed Programs, *The 5th International Conference on Distrib-*

uted Computing Systems, pp.515–522 (1985).
12) Fidge, C.J.: Partial Orders for Parallel Debugging, *ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, pp.183–194 (1988).
13) Kilpatrick, C. and Schwan, K.: ChaosMON-Application-Specific Monitoring and Display of Performance Information for Parallel and Distributed Systems, *ACM/ONR Workshop on Parallel and Distributed Debugging*, pp.57–67 (1991).
14) Pauw, W.D., Helm, R., Kimelman, D. and Vlissides, J.: Visualizing the Behavior of Object-Oriented Systems, *OOPSLA '93*, pp.326–337 (1993).
15) 館村純一，小池汎平，田中英彦：並列論理型言語 Fleng のマルチウインドウデバッガ HyperDEBU，情報処理学会論文誌，Vol.33, No.3, pp.349–359 (1992).
16) 下村隆夫：FIND：デバック自動化支援システム，電子情報通信学会論文誌，Vol.J79-D-I, No.7, pp.446–456 (1996).

(平成 8 年 9 月 27 日受付)
(平成 9 年 11 月 5 日採録)



久保田 稔（正会員）

1954 年生．1978 年東京大学工学部計数工学科卒業，1980 年同大学院工学系情報工学専門課程修士課程修了，同年日本電信電話公社入社．以来，電子交換プログラムの実行制御方式，実時間 OS，分散処理システム，ソフトウェア開発環境の研究開発に従事．1987～1988 年マサチューセッツ工科大学客員研究員．現在 NTT 光ネットワークシステム研究所，主幹研究員，工学博士．電子情報通信学会，IEEE，ACM 各会員．