計算機アーキテクチャ 93－6
(1992. 3. 23)

分散型リアルタイムシステム記述言語: DROL

高汐 一紀　　所 真理雄 [1]
takashio@mt.cs.keio.ac.jp　mario@mt.cs.keio.ac.jp

慶應義塾大学 理工学部

概要 本稿では、まず、分散型リアルタイムシステムをその性質「最大努力」、「最小被害」により特徴付ける。そして、時間制約を隠蔽し、「最大努力」、「最小被害」の機能を提供する分散リアルタイムオブジェクトモデル **DRO モデル**を提案する。DROL は DRO モデルに基づいて C++を拡張したオブジェクト指向分散型リアルタイムシステム記述言語である。DROL の最大の特徴は、リアルタイムオブジェクト間の通信セマンティックスを send、receive による通信プロトコルとして、メタレベルで記述できるところにある。これにより、時間制約を含む要求された仕様に応じた柔軟なシステムを構築することが可能となる。

DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems

Kazunori Takashio　　Mario Tokoro †

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan
Tel:+81-45-560-1150　Fax:+81-45-560-1151

Abstract In this paper, we first characterize distributed real-time systems by their properties of *best effort* and *least suffering*. Then, we propose a distributed real-time object model DRO which encapslate timing constraints, and provides the facility of *best effort* and *least suffering*. Based on the DRO model, we design the object oriented programming language DROL which is an extension of C++ and has the capability of describing distributed real-time systems. The most eminent feature of DROL is that the user can describe the semantics of message communications as a communication protocol with send and receive primitives. With this feature, it is able to construct a flexible distributed real-time system satisfying specifications which include timing constraints.

[1] †He is also with Sony Computer Science Laboratory Inc. , Takanawa Muse Building, 3-14-13 Higashi Gotanda, Shinagawa-ku, Tokyo, 141 JAPAN, mario@csl.sony.co.jp

1 はじめに

我々の計算機環境は今、急速な分散化の時代を迎えている。高速な計算機同士を高速なネットワークで結合した分散環境下でのリアルタイム処理は、マルチメディアシステム、ロボット制御、航空機管制等で用いられる重要な技術である。このようなリアルタイムシステムにおいても、その高速性はもちろん、開発の容易さ、保守性が重要になってきた。

一方、並行オブジェクト指向計算が注目されるようになって久しい。オブジェクト指向の概念はプログラムのモジュール化を容易にする。また、並行オブジェクトは、本来、各々並列に動作し得る実体であり、分散されたシステムを記述するのに有効である [5]。

分散型のリアルタイムシステムをオブジェクト指向で構築する上で問題となるのが、このような時間制約を持って動作しているオブジェクト間での通信機構をどのように提供するかである。ここでは、オブジェクト間での通信にともなう時間制約の検査と例外処理の起動の機構が問題となる。

非分散型 (集中型) のリアルタイムシステムと異なり、分散型リアルタイムシステムの実現は、**最大努力 (best effort)** と**最小被害 (least suffering)** の 2 つの性質により特徴付けられる。分散型リアルタイムシステムでは、ネットワークの負荷の変化にしたがって、受信オブジェクト側で要求される時間制約が動的に変化する。「最大努力」とは、要求された機能の要求された時間内での達成に対し最大限の努力をすることを意味する。さらに、受信側でのタイムアウト発生の通知は、送信側オブジェクトでは信頼性のないものとなる。「最小被害」とは、これらの時間制約に関わるエラーの伝播を最小に押えなければならないことを意味する。

本稿の目的は、時間制約を持ったオブジェクト間の通信機構について検討し、時間制約に対し柔軟に対処できるオブジェクトモデルを構築することにある。そこで、本稿では並行オブジェクトモデルに、「最大努力」、「最小被害」の両性質を導入し、その実現のために通信プロトコル記述の機能を持たせた分散リアルタイムオブジェクトモデル **DRO** モデルを提案する。

本稿において設計/実装される分散型リアルタイムシステム記述言語 **DROL** は、DRO モデルを基本モデルとし、C++を拡張したオブジェクト指向言語である。DROL は以下に挙げる特徴を持つ。

- 「最大努力」の実現のために、**時間仮想関数 (time virtual function)** の機能を導入。
- 「最小被害」を実現するために、送信側での時間制約検査及びタイムアウトの発生機構を導入。
- オブジェクトレベルでの通信機構と例外処理の意味を**通信プロトコル (communication protocol)** としてメタレベルで定義する。

本稿の構成についてその概略を述べる。第2章では、分散リアルタイムシステムについて議論し、記述言語に対する要求事項をまとめる。第3章では、言語DROLの基本モデルとなるリアルタイムオブジェクトモデル DRO モデルを提案する。さらに、第4章で DRO モデルに基づいたオブジェクト指向分散型リアルタイムシステム記述言語 DROL の言語仕様を与える。最後に第5章で研究全体を通じての結論及び将来の研究方針について述べる。

2 分散型リアルタイムシステム

本章では、以下での議論を明確にするため、分散型リアルタイムシステムを定義する。分散システム (*distributed system*) はネットワークで接続された複数のプロセッサから構成される。各オブジェクトは同一の計算機上に存在する場合もあれば、異なった計算機上に存在する場合もある。異なる計算機上に存在するオブジェクト間の通信はネットワークを介してのみ行なわれる。

このようなシステムで問題になるのが通信経路上で発生する遅延時間を無視できなくなることである。特に、このような大規模分散環境上でのリアルタイムシステムの構築において、我々は以下のような性質を考慮する必要がある。

a. **大域的な時計を持つことが不可能である。**
各ノードごとに存在する局所的な時計間の整合性をとることが不可能である。リアルタイム計算はこれらの局所的な時計にのみ依存して行なわれなければならない。

b. **反応時間を決定することが不可能である。**
ネットワークの形態は常に変化しており、ネットワークが切断されてしまうこともあり得る。また、計算機それ自体が故障してしまうかもしれない。そのため、通信経路における最悪遅延時間を決定できなくなり、要求に対する返値の到着時間が見積もれなくなる。

a. の性質により、各オブジェクトは時間制約を満たすことが可能かどうかの判断を局所的な時計によってのみ行なわなければならない。また、b. により、受信オブジェクトに許された実行時間は動的に変化する。さらに、送信側オブジェクトの立場から見ると、返値を送信あるいは受信するまでの時間を決定できないことになる。

図1に時間制約をともなったオブジェクト間通信の全体像を示す。図1において、P_{ct}はオブジェクト P で

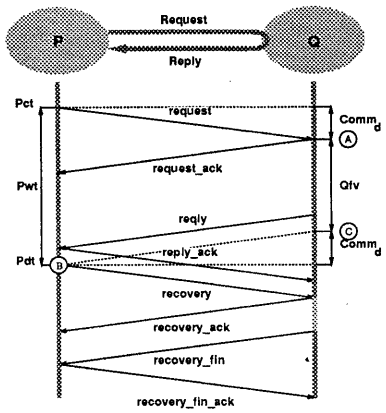

図 1: 分散リアルタイムオブジェクト間通信とタイムアウト

の現在の時刻、P_{dt}はPにおけるデッドラインを示す。すなわち、$P_{wt}: P_{dt} - P_{ct}$がP側でのメッセージ交信に関し、クライアントが厳守しなければならない時間となる。この値は、[6]においてタイムフェンス (*time fence*) と呼ばれる[1]。

ARTS カーネル [6] に代表される伝統的な手法では、最悪通信遅延時間が見積もれるとの仮定がなされていた。この場合、送信側でのタイムフェンスが受信側に通知されており、メソッド実行の最悪実行時間が決定されていれば、時間制約を満たせるかどうかを実行時の検査機構により検証することが可能である。しかし、大規模分散システムに適用する場合、この手法はいくつかの問題を含んでいる。まず第1に、上記2つの性質によりネットワーク遅延時間が見積もれないため、受信側でメソッド実行に許される時間 (受信側でのタイムフェンス) が動的に変化する。それゆえ、非分散型のリアルタイムシステムと異なり、時間制約の保証が困難となる。そこで、受信側オブジェクトでの時間制約が満たされることを保証する「最大努力」のための機構が必要となる。

第2に、受信側Qからのタイムアウト通知は送信側Pでは信頼できないものとなってしまう。予測できない遅延がネットワーク上で発生した場合を考えてみる。Q側でのタイムアウトの通知がP側に到着した時点で、既にP側でのデッドライン時間P_{dt}を過ぎてしまっている可能性がある。極端な場合、ネットワークの一部が切断されたとき、送信オブジェクトPはその返答メッセージを永遠と待ち続けることになり、システムが停止してしまうことも考えられる。この問題を解決するために、送信側オブジェクトがそのタイムフェンスを管理し、満たされなかった場合にタイムアウトを発生させる機能が必要となる。これにより「最小被害」が実現される。

我々の目的は分散型リアルタイムシステムの構築にある。上で議論したように、非分散型のリアルタイムシステムと異なり、システムに「最大努力」「最小被害」を導入する能力が分散型リアルタイムシステムの記述言語に求められる。前者の機能を実現するために、我々は時間仮想関数と呼ばれる概念を導入する。後者のために、送信オブジェクト側での時間制約検査の機構を導入する。オブジェクト間通信のセマンティックスをオブジェクトごとに定義可能とすることによって、各メソッド実行の時間制約と例外処理起動の関係を明確にする。

[1]本稿でも以後この値をタイムフェンスと呼ぶことにする

3 DRO モデル

[9, 8, 4, 5] において、我々は並行オブジェクトモデルが分散システムを記述する上で非常に有効かつ強力なモデルとなることを示してきた。本章では、DROLの基本モデルとしてDROモデルを提案する。DROモデルでは、並行オブジェクトモデルに対しリアルタイム計算の導入を行う。DROモデルは2つの機能、すなわち、時間制約を満たすための「最大努力」とシステムの「最小被害」の保証のための機能を提供する。

3.1 分散リアルタイムオブジェクトと時間制約

C++等、幾つかのシステムで定義されるオブジェクトは、データを隠蔽するがその中にThread of Control (以下スレッド呼ぶ) を持たない。実行スレッド (あるいはプロセス) はオブジェクトモデルの外にあり、オブジェクトに対しその状態を知るため、あるいは状態を変更するためにオブジェクトの呼び出しを行なう。一方、並行オブジェクトは、オブジェクト内にデータを隠蔽する他に、1つ以上のスレッドを持ち、データの隠蔽と並行制御機構の隠蔽も同時に行なうモデルである。また、このモデルは時間制約をオブジェクト内に隠蔽する能力を持つ。内部に以下の特徴を持ち定義される並行オブジェクトを分散リアルタイムオブジェクトと呼ぶことにする。

- 次のような時間制約を内部に隠蔽する。
  - **局所的時間制約:** メソッドごとに定義される時間制約、すなわち、最悪実行時間 (*worst case execution time*) と例外処理。
  - **アクティブメソッド:** オブジェクト内で周期的に実行されるメソッド。開始時間 (*start*

time)、終了時間 (*end time*)、周期 (*period*)、デッドライン (*deadline*) を持つ。

- **通信時間制約:** メッセージ送信に負荷される時間制約、すなわち送信オブジェクト側でのタイムフェンス (*time fence*) と例外処理。

- 受信側の立場での「最大努力」を実現している。
- 送信側の立場での「最小被害」を実現している。

さらに、同一オブジェクト内で同時実行を許すスレッド数により次の2つのモデル、すなわち、単一スレッドモデルと多重スレッドモデルが考えられる。後者のモデルでは、オブジェクトの状態がスレッドごとに複数存在することになり、その挙動を厳密にモデル化することが不可能である。また、オブジェクト内のデータを複数のスレッドによって共有するため、オブジェクト内のスレッド間での同期操作が不可欠となる。これは、同期操作に対する時間制約を導入する必要があることを意味し、システムの記述を複雑なものにしてしまう危険性がある。単一スレッドオブジェクトモデルにより、オブジェクトの状態を1つに決定することが可能となり、オブジェクトの仕様を厳密にモデル化することが可能となる。よって、システムの設計、保守、解析が容易になる。以上の理由により、DROL では単一スレッドオブジェクトモデルを採用した。

3.2 分散リアルタイムオブジェクトの構造

図2に示すように、分散リアルタイムオブジェクトは2つのレベル、**オブジェクトレベル (object level)** と**メタレベル (meta level)** を持つ。以下のオブジェクトが存在する。

a. **ベースオブジェクト:** オブジェクトレベルに位置するオブジェクト。並行オブジェクトモデルに基づき定義されるオブジェクトであり、1つのスレッド、内部データ、プロシージャ、仮想プロセッサを持つ。ベースオブジェクトは受理されたメッセージに対応するメソッドの実際の実行を担当するオブジェクトであり、その実行はメタオブジェクトによって管理される。

b. **メタオブジェクト:** メタレベルに位置し、メタレベルでの計算を担当するオブジェクト。ベースオブジェクトにおける仮想プロセッサに意味を与える。すなわち、ベースオブジェクトの振舞いの制御する。メタオブジェクトは、主に、仮想プロセッサに対する到着メッセージのスケジューリング、オブジェクト間通信プロトコルの意味定義/監視を担当する。

c. **タイマオブジェクト:** オブジェクト内でのメソッド実行及びオブジェクト間通信にともなう時間制約の監視に用いられるタイマ。各ノードごとに存在する局所的な時計を抽象化したオブジェクト。メタオブジェクトとトリガオブジェクトにより参照される。

d. **トリガオブジェクト:** アクティブメソッドの実行を担当するオブジェクト。周期的にアクティブになりベースオブジェクトのアクティブメソッドを起動する。

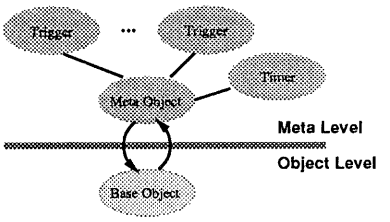

図 2: DRO モデル オブジェクト構造

3.3 メタオブジェクトによるメッセージ同期モデル

DRO モデルでは、メタオブジェクトはベースオブジェクトを以下で定義される有限オートマトン、**抽象オブジェクトオートマトン (abstract object automata)** を用いてモデル化する。メタオブジェクトは、このオートマトンに基づいて、現在のオブジェクトの状態を把握/管理する。

定義 3.1 抽象オブジェクトオートマトン A は次の *6* 項組で定義される。

$$A = < S,\ s_0,\ O,\ P,\ \delta,\ \rho >$$

ここで、S: オブジェクトの状態集合、s_0: 初期状態 $s_0 \in S$、O: オブジェクトが持つオペレーション (メソッド) の集合 (オートマトンの入力アルファベット)、P: オブジェクト間通信 (例外処理起動) プロトコル集合、δ: $S \times O \rightarrow 2^S$ 状態遷移関数[2]、ρ: $O \rightarrow P$ プロトコル定義関数である。

EOをある状態で受理可能なオペレーションの集合としよう。この集合を**実行可能集合 (enable-set)** と呼ぶ。

[2]本稿では 2^X を集合 X のベキ集合の表記法として用いる

定義 3.2 状態 s $(s \in S)$ における実行可能集合 EO

$$EO = \{o \mid (\delta(s,o) \neq \phi,\ s \in S) \wedge (eo \neq \varepsilon)\}$$

DROモデルにおける同期セマンティックスは、このような実行可能集合を前述の抽象オブジェクトオートマトンから定義することにより提供される。メタオブジェクトでのメッセージ同期メカニズムは次のようになる。1) 到着メッセージをプライオリティ付きキューに格納する。2) メタオブジェクトは現在のオブジェクトの状態から実行可能集合を求め、実行可能な要求メッセージの中でもっともプライオリティの高いものと同期をとり、メソッドを実行する。3) 同一プライオリティに複数の実行可能要求メッセージが存在した場合、待ち時間のもっとも長い要求から処理する。同様の手法は [7, 1] に見ることができる。

3.4 メソッド実行の時間多様性

オブジェクト指向言語の重要な性質として、メソッドの多様性 (*polymorphism*) がある。リアルタイムシステムでは、オブジェクトは要求された時間内で処理を終了しなければならない。さらに、2章で議論したように、分散型リアルタイムシステムでは、非分散型リアルタイムシステムと異なり、「最大努力」という性質を支援しなければならない。そこで、DROモデルは送信側オブジェクトにより要求された時間制約内で実行可能なメソッドを動的に選択する機構を提供する。分散リアルタイムオブジェクトでは、その実体を持たない仮想的なメソッド、時間仮想メソッド (*time virtual method*) を定義することができる。時間仮想メソッドには、異なる時間制約を持った複数のメソッドが実メソッドとして宣言される。実行時には、要求された時間制約を満たし、かつ最長の実行時間を持つメソッドが動的に選択、実行される。我々は、この概念を**メソッドの時間多様性**と呼んでいる。これにより、要求時間内での柔軟な対応が可能となる。

3.5 オブジェクト間通信と例外処理

非分散型リアルタイムシステムと異なり、分散型リアルタイムシステムは、性質「最小被害」を実現するために、送信オブジェクト側でのタイムアウトを支援する必要がある。DROモデルでは、ユーザが記述言語上で基本的なプリミティブとユーザ定義のメソッド(例外処理のための関数等)を組み合わせることによって、プログラム上で現れるメッセージ通信に関する独自のプロトコルを構築し、その意味を与えることを可能にする。そのために、オブジェクトレベルでの通信モデルと、メタレベルでの通信モデルを区別する。

3.5.1 オブジェクトレベルでの通信モデル

オブジェクトレベルすなわちメソッド記述レベルでの通信モデルとして、DROモデルは同期交信と非同期交信そして非同期返答を支援する。分散型リアルタイムシステムは、最悪遅延実行時間の見積もりが不可能であるという特徴を持つ。そこで、「最小被害」を実現するために、DROモデルは時間制約の送信側での検査、タイムフェンスを満たせなかった場合の例外処理の起動のための機構を提供する。

送信側での時間制約検査を導入した場合、そのタイムアウトの発生にともなう例外処理の起動方法が複雑になる。例えば、受信側オブジェクトが送信側からのタイムアウト通知を受信したとき、受信側は既に返値を送ってしまっている場合も考えられる。その例外処理の起動はそのオブジェクトが置かれた環境と、要求されるシステムの仕様により異なり、そのプロトコルを唯一つに定めるのは不可能である。そこで、DROモデルは、オブジェクト間の通信と例外処理の意味を通信プロトコル (*communication protocol*) としてメタレベルで定義する機能を提供する。

3.5.2 メタレベルでの通信モデル

DROモデルでは、メタレベルの基本通信モデルとして **send**、**receive** による非同期のメッセージ送受信を採用する。その上で、送信側/受信側の状態遷移図を定義する。これにより、オブジェクト間のメッセージ通信と例外処理の起動に対するプロトコルを定義し、その意味を与える。

DROモデルが提供するプリミティブイベントとその意味を図3に示す。これらのプリミティブはリアルタイムオブジェクト間でメッセージ交信を行なう際に発生するイベントとそれに対応する acknowledgement からなる。通信プロトコルにおけるプリミティブイベントの送受信の流れの概観は、図1に示した通りである。ここで、送信側は request を送信すると同時にメタオブジェクト内でタイマを起動し、送信側でのタイムアウトを監視する。ユーザは、これらのプリミティブイベントを選択的に使用し、状態遷移図を構築することにより、プロトコルを定義する。プロトコルの定義例は、4.4節で示す。

4 DROL 言語仕様

DROL は DRO モデルに基づき C++を拡張した分散型リアルタイムシステム記述言語である。以下でその言語仕様について述べる。

プリミティブイベント	意味
request	リクエストメッセージ
request_ack	request メッセージへの ack
reply	返答メッセージ
reply_ack	reply メッセージへの ack
recovery	例外処理の依頼
recovery_ack	recovery メッセージへの ack
recovery_fin	例外処理の終了の通知
recovery_fin_ack	recovery_fin メッセージへの ack
self_recovery_fin	自身の例外処理の完了
time_out	タイムアウトの発生

図 3: オブジェクト間通信プリミティブイベント

4.1 オブジェクト定義

DROL では、C++のオブジェクトに加え、DRO モデルに基づく分散リアルタイムオブジェクトを定義することができる。キーワード DRObject を C++でのクラス宣言の前に付加することによりリアルタイムオブジェクトを定義する。リアルタイムオブジェクトとして宣言されたクラスはオブジェクト生成時にベースオブジェクトとメタオブジェクトの両者を生成する。C++でのオブジェクト定義と異なる点は、private 部、public 部の他に state 部、protocol 部を持つ点である。state 部では、先に導入された抽象オブジェクトオートマトンに基づき、オブジェクトが取り得る状態とその遷移関係を定義する。この仕様は、到着メッセージとの同期機構としてメタオブジェクトにより利用される。また、protocol 部ではオブジェクト間交信のセマンティックスを定義する。図 4にオブジェクトの定義例を示す。protocol 部の定義方法は 4.2節で言及する。

4.1.1 メソッド定義と時間制約支援

DROL で定義される各メソッド[3]には、局所的な時間制約を宣言することが可能である。ここでは、メソッドごとの最悪実行時間とそれが破られた場合の例外処理の指定を行なう。図 4の例では、関数 read() は最悪の場合でも 20msec で処理が終了しなければならないことが定義されている。もし 20msec 以内に終了しなければ関数 read_abort() が呼ばれる。

周期的に実行されるメソッドの定義は、メソッドの定義時にキーワード active を付加することにより行われる。図 4の例では、メンバ関数 screen() が能動的に呼び出される関数として宣言されている。この関数は周期 60sec 毎に呼び出され、デッドライン (この場合は同じく 60sec) をミスした場合に関数 screen_abort() を呼び出すことを宣言している。

[3]C++の用語ではメンバ関数であるが、本稿では他のオブジェクト指向言語同様、メソッドと呼ぶことにする。

```
 1 DRObject class BBuffer {
 2 private:
 3   int   count;
 4   int   read_abort();
 5   int   write_abort();
 6   int   write_with_screen(char* data)
 7     within(0t20) timeout(write_abort());
 8   int   write_without_screen(char* data)
 9     within(0t10) timeout(write_abort());
10 public:
11   int   read(char* data, int size)
12     within(0t20) timeout(read_abort());
13   t-virtual
14     int   write(char* data)
15           = {write_with_screen(),
16              write_without_screen()};
17   active  int   screen()
18     start() end() period(0t60s)
19             deadline() timeout();
20 state:
21   $EMPTY  := write.<$NORMAL,$EMPTY>
22            + screen.$EMPTY;
23   $NORMAL := write.{<[count == Max]$FULL
24                        | [count < Max]$NORMAL
25                      , $NORMAL>}
26              + read.{<[count == 0]$EMPTY
27                        | [count > 0]$NORMAL
28                      , $NORMAL>}
29              + screen.$NORMAL;
30   $FULL   := read.<$NORMAL,$FULL>
31              + screen.$FULL;
32 }
```

図 4: オブジェクト定義例

4.1.2 時間仮想関数定義

DRO モデルでは、メソッドの時間多様性を導入した。DROL では、この機構を時間仮想関数として実現する。時間仮想関数は、キーワード t-virtual を用いて次のように定義される。

t-virtual passive function()
= { subfunc1(), subfunc2(), ..., subfuncn() };

上記の記述で定義された関数 function() が public 部で、その他の副関数 subfunc1(), ..., subfuncn() が private 部で定義されていたとしよう。function() が呼び出された場合、まず要求されたデッドラインを調べる。そして、副関数の中から最悪実行時間 (*worst case execution time*) が最大のものでかつデッドラインを満たす関数が実際に呼び出される。図 4では、仮想時間関数 write() に対して副関数

`write_with_screen()` と `write_without_screen()` を定義している。前者は時間的に余裕のある場合に呼ばれる関数で、バッファへの書き込みと同時にその結果を画面表示する関数であり、後者は時間に余裕のない場合の書き込みのみを行なう関数である。

4.1.3 状態遷移関係の定義

state 部では、到着メッセージとの同期に関わるオブジェクトの振舞いを仕様化する。ある状態からの遷移アクションは、その状態での実行可能集合を示している。図 4の例では、BBuffer が取り得る 3 つの状態 EMPTY、NORMAL、FULL を定義している。例えば、状態 NORMAL の定義は次のような意味を持つ。a) メソッド `write()` を実行 (完了) し、i) かつ実行後に条件 [count == Max] を満たす場合は状態 FULL に遷移する。ii) 条件 [count < 0] を満たす場合には状態 NORMAL に留まる。b) メソッド `write()` の実行が完了しなかった場合は状態 NORMAL のままである。c) また、メソッド `read()` が実行された場合、i) 条件 [count == 0] を満たすならば状態 EMPTY に、ii) [count > 0] を満たす場合には状態 NORMAL に遷移する。d) 実行が完了しなかった場合、状態 NORMAL に留まる。また、e) メソッド `screen()` の実行によっての状態遷移は起こらない。

上記の表記法で用いられるオペレータを以下にまとめる。

オペレータ	意味
a . B	連接演算子
B + B	選択演算子
{B | B }	条件付き遷移式
< $State_a$, $State_b$ >	時限選択演算子
{ expression }	後条件式

時限演算子では、メソッド実行が正常終了した場合は状態 $State_a$、時間制約を満たせずアボートした場合は状態 $State_b$へ遷移する。また、後条件式には、C 言語及び C++の式を記述できる。

4.2 通信時間制約記述

DROL は、通信にともなう時間制約記述のために、時間制約付き呼びだし **invoke** 文を用意している。invoke 文の記述は次のようになる。

```
1 invoke (<invoke expression>; <within time>;
2         <protocol>) {
3   execute:
4     // <asynchronous execution body>
5   timeout:
6     // <timeout body>
7 }
```

ここで、< *invoke expression* >は C++におけるメソッド呼びだし表記である。< *within time* >には送信側オブジェクトのタイムフェンス値を< *protocol* >にはオブジェクト間通信に用いられるプロトコル (4.3 節で述べる) を指定する。< *within time* >で指定された時間内に reply が帰ってこない場合は、例外処理として< *timeout body* > が実行される。また、< *asynchronous execution body* >には、相手オブジェクトに request を送信した後に実行されるプログラムを記述する。すなわち、< *asynchronous execution body* > が存在する場合は非同期交信、存在しない場合は同期交信を意味する。

また、DROL は返答値の送信手段として 2 つの機構を用意している。すなわち、`return` 文と `reply` 文である。`return` 文の意味は C++同様である。一方、`reply` 文の場合、メソッド記述の任意の時点で返答値を送信オブジェクトに返すことができる。

```
1 ...
2 reply reply_value
3 ...
```

オブジェクト間の交信は、この非同期返答が完了した時点でコミットしたとみなすことができる。

4.3 通信セマンティックス

DROL では DRO モデルに基づき、メタレベルでの通信セマンティックス定義のための枠組みを提供する。この定義、宣言はオブジェクト定義の際の `protocol` 部を用いて記述される。プログラマはオブジェクトが取り得る状態と、その間の遷移アクションから状態遷移を構築することにより、プロトコルの定義を行う。遷移アクションは、図 3で示したプリミティブイベント (10 種) の送受信 (*send, receive*) に相当する。

プロトコルは各オブジェクトで定義され、通信を行う際にプロトコル名で参照される (4.2節)。送信側オブジェクトと受信側オブジェクトでのプロトコルの整合性はコンパイル時にコンパイラによって検査される。プロトコルの継承について述べる。スーパクラスで定義されたプロトコルは、そのサブクラスで利用することが可能である。送信側と受信側オブジェクトが同じスーパクラスを継承することでプロトコルの共有が実現できる。C++が支援する多重継承を利用することにより複数のプロトコルを継承することも可能である。

4.4 プロトコル定義例

3.5節で述べたように、DRO モデルは送信側での時間制約検査を導入した。そのタイムアウトの発生にともなう例外処理の起動方法には、次の 3 つの場合が考えられる。

a. 受信側での回復処理の要求を必要としない場合(関数的な処理要求)。

b. 受信側での回復処理を要求はするがその確認を必要としない場合。

c. 受信側での回復処理を要求しかつその確認を必要とする場合。

a. の通信プロトコルを定義すると図5のようになる。ユーザが特にプロトコルを指定しなかった場合、コンパイラはデフォルトとして用意されたb. のプロトコルを使用する。

```
 1 DRObject class Example {
 2   private:
 3     abort();
 4   public:
 5     func() within(0t20) timeout(abort())
 6           protocol(simple);
 7     ...
 8   protocol:
 9     simple [>sender
10             #ACTIVE  := ~request
11                         . #WAITING_REPLY;
12             #WAITING_REPLY
13                      := reply . #ACTIVE
14                           + time_out
15                           . #RECOVER;
16             #RECOVERY:= self_recovery_fin
17                         . #ACTIVE;
18     simple [>receiver
19             #DORMANT := request . #ACTIVE;
20             #ACTIVE  := ~reply . #DORMANT
21                         + time_out
22                         . #RECOVER;
23             #RECOVER := self_recovery_fin
24                         . #DORMANT;
25 }
```

図 5: プロトコル定義例

5 結論

本稿で、我々はまず最初に、分散型リアルタイムシステムをその性質「最大努力」、「最小被害」により特徴付けた。そして、「最大努力」、「最小被害」のための機能を提供する新しいオブジェクトモデル **DRO モデル**を提案した。さらに、DRO モデルに基づき、C++を拡張したオブジェクト指向分散型リアルタイムシステム記述言語 **DROL** を構築した。

DROL では、時間仮想関数を導入することにより、「最大努力」を実現した。また、送信側オブジェクトでのタイムアウトの機構を導入することにより、「最小被害」を実現した。この2つの機能により、分散型リアルタイムシステムの記述が可能となる。さらに、通信プロトコル記述の機能を導入することにより、柔軟なシステム構築が可能となる。

我々は今後の課題として以下の研究項目に興味を持っている。まず第1に、他のリアルタイム支援環境上での実行系の構築を考えている。特に、本研究で導入した DRO モデルに近い概念を持つ Muse オペレーティングシステム [8] 上での実行系の構築に興味を持っている。第2に、我々は、CCS [2] に時間概念を導入した並行計算モデル **Timed CCS** [3] を開発している。DROL で記述したプログラムをこれら並行計算モデルにマッピングするコンパイラを実装することにより、システムの静的な検証系を構築することが可能になると考えている。

謝辞

本稿の作成にあたり、貴重なコメントを頂きました電子技術総合研究所 石川裕 博士、ならびに慶應義塾大学理工学部 佐藤一郎 氏に感謝いたします。

参考文献

[1] Dennis G. Kafura and Keung Hae Lee. Inheritance in Actor Based Concurrent Object-Oriented Languages. In *ECOOP89*, 1989.

[2] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[3] Ichiro Satoh and Mario. *Timed Process Algebla Semantics of A Real-Time Concurrent Object-Oricuted Programming Language*. Technical Report, Keio University, 1991.

[4] Mario Tokoro. Computational Field Model: Toward a New Computational Model/Methodology for Open Distributed Environment. In *Proceedings of the 2nd IEEE Workshop on Future Trends in Distributed Computing Systems*, Sep. 1990. Cairo, Egypt.

[5] Mario Tokoro. Towards Computing Systems in '2000. In *LNCS*, Springer-Verlag, 1992.

[6] Hideyuki Tokuda and Clifford W. Mercer. ARTS: A Distributed Real-Time Kernel. *Operating System Review*, 23(3), 1989.

[7] C. Tomlinson and V. Singh. Inheritance and synchronization with Enabled-Sets. In *OOPSLA'89*, 1989.

[8] Yasuhiko Yokote, Fumio Teraoka, Atsushi Mitsuzawa, Nobuhisa Fujinami, and Mario Tokoro. The Muse Object Architecture: A New Operating System Structuring Concept. *Operating System Review*, 25(2), 1991.

[9] A. Yonezawa and M. Tokoro, editors. *Object-Oriented Concurrent Programming*. MIT Press, 1987.