

並列オブジェクト指向言語A-NETLのデバッグシステム

片平 透 吉永 努 馬場 敬信
宇都宮大学工学部

並列プログラムのデバッグには振舞いの非決定性やプローブ効果の影響、大域的な時間順序の欠如、デバッグ情報の複雑化などの問題がある。我々は、並列オブジェクト指向トータルアーキテクチャA-NETの研究の一つとして、並列オブジェクト指向言語A-NETLにより記述された並列プログラムをデバッグするシステム"A-NETLデバッガ"を開発中である。本稿では、上記の問題を踏まえ、A-NETLデバッガにおいて採用したイベントを用いたプログラムのリプレイ方式とブレイクポイント方式を併用してデバッグする手法、複数の表示法を用いてデバッグ情報を抽象化・視覚化する手法について述べる。

A Program Debugging System for the Parallel Object-Oriented Language A-NETL

Tohru Katahira Tsutomu Yoshinaga Takanobu Baba
Department of Information Science
Utsunomiya University
Utsunomiya 321, Japan

Difficulties associated with debugging parallel programs include nonrepeatability of the behavior, the probe effect, and the lack of a synchronized global clock. As a part of the A-NET total architecture project, we have been developing a program debugging system, called "A-NETL Debugger", for the Parallel Object-Oriented Language A-NETL. Our design integrates many facilities, such as execution replay, breakpointing, and an event-based view of the execution of the A-NETL program. This paper describes the design principles and the implementation.

1. はじめに

我々の研究グループでは並列オブジェクト指向トータルアーキテクチャA-NET[1,2]の一環として、プログラム記述言語である並列オブジェクト指向言語A-NETL[3]を定義した。さらに現在、A-NETL言語によるプログラム開発支援システムの1つとしてプログラムデバッグシステム"A-NETLデバッガ"を設計・開発中である[4]。

並列プログラムのデバッグには従来の逐次プログラムに対する技術では解決できない様々な問題があり、近年盛んに研究が行なわれ、様々なアプローチが報告されている[5,6]。

我々はそれら多くのアプローチを検討した結果、従来のブレークポイント方式に加えてイベントベース方式を採用した。A-NETLデバッガでは、プログラム詳細部の解析には各オブジェクトに対するブレークポイントデバッグを用い、プログラム全体の解析にはプログラム全体の振舞いを記録したイベント履歴のブラウジング、イベント履歴に基づくプログラムのリプレイをアニメーション表示を用いた解析を実現する。

本論文では、A-NETL言語の特徴とデバッグする際に生じる問題点を挙げ、A-NETLデバッガの設計方針と機能を紹介して並列プログラムのデバッグに関する問題の解決法を述べる。

2. 並列オブジェクト指向言語A-NETL

2. 1 A-NETLの特徴

並列オブジェクト指向A-NETLは、アクター理論に基づきA-NET超並列計算機上での実行を前提とした並列処理記述言語である。

A-NETLによるプログラミングモデルでは、オブジェクトが独自の計算資源と計算能力を持つ並列処理の単位であり、1つのプログラムは多数のオブジェクトにより構成される。各オブジェクトは、同期的または非同期的なメッセージパッシングにより互いに協調しながら並列に計算を行なう。また、オブジェクトにはユーザが静的に定義した静的オブジェクト、クラスにより実行時に生成される動的オブジェクトの2種類があり、また静的オブジェクトには、同種の静的オブジェクトを複数定義するインデックスオブジェクトがある。A-NETLでは、プログラムはオブジェクト単位に記述され、オブジェクトの記述には状態変数宣言部とメソッド宣言部がある。

メッセージ送信には、past型（非同期型）とnow型（同期型）そしてfuture型（非同期型+返答メッセージ待ち）の3タイプがある。メッセージは受信側オブジェクトにおいて到着順に実行される。

2. 2 A-NETLプログラムのデバッグに関する問題

A-NETLプログラミングモデルにおいて起こりえる実行時エラーは、オブジェクト内部で閉じているエラーと、オブジェクト間通信の影響により引き起こされるエラーの2種類が存在すると考えられる。特に、オブジェクト間通信の影響により引き起こされるエラーをデバッグする際に起こりえる問題としては、以下のようなものが考えられる。

(1) 振舞いの非決定性

オブジェクトの振舞いや内部状態は、メッセージの実行順序に依存する。メッセージの実行順序は現在のところメッセージの到着順序に等しく、メッセージの通信経路の関係からユーザが予期していたメッセージの実行順序とは異なってしまう可能性がある。また、A-NET計算機のノード間ネットワークの状態は動的に変化し、メッセージも決まった経路を常に一定の時間で転送されるとは限らない。よってプログラムの振舞いは非決定的となり、以前に実行した際に得られたような振舞いをもう一度再現できる保証はない。

(2) プローブ効果

一般に、逐次プログラムをデバッグするには、コンパイラにおいてプログラムコード内にデバッグ用の特殊命令を挿入し、その命令により実行を中断させ、プログラムの状態を観察する手法が採られている。しかし、A-NETLプログラミングモデルのように複数のオブジェクトが独立して並列に実行される場合、観察したいオブジェクトの実行を中断させてしまうと対象オブジェクトの状態は観察できるが、その後のプログラム全体の振舞いが通常時とは異なってしまう。また各ノードプロセッサが独立で動作し、非同期的な通信を想定しているA-NET計算機の構成上、すべてのノードプロセッサを同時に止めてオブジェクトの状態を観察することは現実的ではない。よって、観察対象のオブジェクトに対するデバッグ用操作はプローブ効果としてプログラム全体の振舞いに影響を及ぼす。

(3) グローバルな時間順序の欠如

A-NET計算機では、すべてのノードプロセッサを同一のクロックで制御することは想定しておらず、A-NETLプログラミングモデルにおいても1つ

のオブジェクトを超えたプログラム全体に対する統一的な時間概念を規定することはできない。また、オブジェクト間の実行順序関係はメッセージの送信側と受信側の順序関係のみが明確であり、通信を行っていないオブジェクト間の関係に関しては、明確な時間順序関係を規定することができない。よって、実時間を基準にしたプログラム全体の振舞いの表現が困難となる。

3. A-NETLデバッガの設計方針

3. 1 基本アプローチ

我々は、以下に挙げるような点を踏まえ、イベントベース方式とブレイクポイント方式の併用を採用した。

(1) イベントベース方式

A-NETLプログラミングモデルにおけるオブジェクトの振舞いは、

- A. オブジェクト間のメッセージ通信
- B. オブジェクトの内部状態の変化

により決定される。

A-NETLデバッガではこれを踏まえ、イベントベース方式を用いてAやBのオブジェクトの振舞いを決定する事象をイベントとして記録することにより、非決定的な振舞いの1パターンの実行過程の軌跡を決定的に解析する。

この方式は、イベント履歴を生成するためにプログラムを1度テスト実行する必要があるが、イベント記録処理によるブロープ効果の影響を受ける。しかし、記録する事象と記録する情報をできるだけ限定して記録処理を軽くすることにより、ブロープ効果の影響をなるべく小さく抑えることができる。

(2) 論理的な時間概念の利用

A-NETLプログラミングモデルでは、実時間ではプログラムの実行順序を表現できないが、イベントの発生順からプログラムの論理的な実行順序を割り出すことができる。

図-1は、オブジェクト間の実行順序関係と論理時間の関係を示したものである。例えば、オブジェクトAにおいてEA3番目に発生したイベントとしてオブジェクトBへのメッセージ送信があり、オブジェクトBにおいてEB2番目に発生したイベントとしてオブジェクトAからのメッセージ受信があったとする。"メッセージを送信した時間は、メッセージを受信した時間よりも早い。"ということが論理的に成り立つので、この関係をもとにオブジェクト

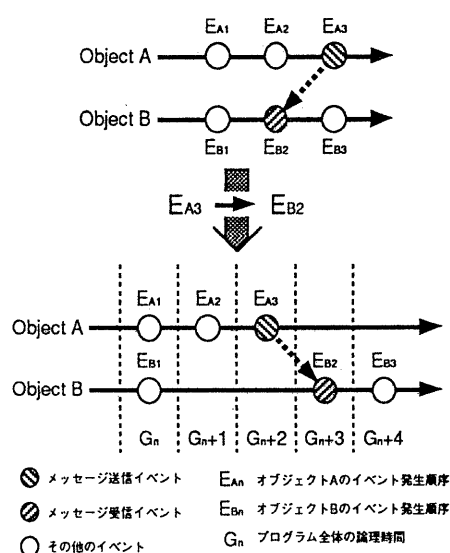


図-1 オブジェクト間の実行順序関係と論理時間

AとオブジェクトBの実行順序関係が規定される。この考え方をプログラム全体に適用することにより、プログラム全体の論理的な時間Gが得られる。

A-NETLデバッガにおける論理時間の算出は、イベント記録時に各ノードプロセッサにおいてイベントが発生することにタイムスタンプを刻み、メッセージ送信時に送信側のタイムスタンプをメッセージに付加してタイムスタンプの値をイベントデータとして記録し、イベント記録後にメッセージ通信関係から割り出すことにより実現できる。

(3) ブレイクポイント方式の併用

各オブジェクトの内部を観察するには、各オブジェクトに対するブレイクポイント方式のデバッグが効果的である。A-NETLプログラミングモデルにおけるオブジェクトは独立した計算資源を持っているので、その実行形態は1つの逐次計算機を占有して実行しているのに等しい。

よって、A-NETLデバッガでは、イベントベースデバッグ機能と各オブジェクトに対するブレイクポイントデバッグ機能の両方をサポートし、オブジェクト内部レベルの振舞いをブレイクポイントデバッグ機能、オブジェクト間通信レベルの振舞いをイベントベースデバッグ機能により解析する。

3. 2 デバッグ処理の流れ

次頁の図-2は、A-NETLデバッガにおける処理の流れを示したものである。プログラムのデバッグを開始する際、ユーザは次のいずれかを選択する。

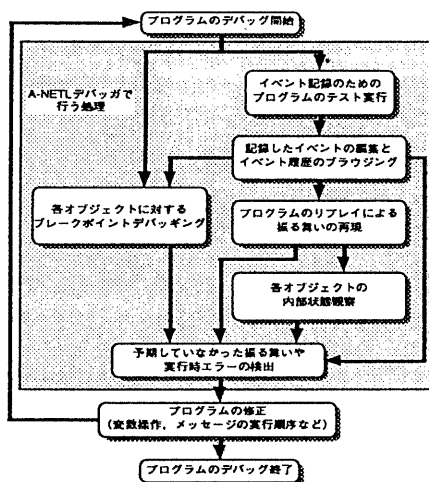


図-2 デバッグ処理の流れ

(1) 各オブジェクトのブレークポイントデバッグ

ユーザが解析対象のオブジェクトを選択し、そのオブジェクトのメソッドのステートメント実行や変数の変化にブレークポイントを設定する。この際、A-NET計算機では、デバッグ対象のオブジェクトが割り付けられているノードに対してのみデバッグモードでプログラムを実行する。このデバッグ方式では、オブジェクト間通信の影響による実行時エラーは解析できないが、オブジェクト内部の実行時エラーを効率よくデバッグすることができる。

(2) イベント履歴を用いたプログラムの解析

一度イベントの記録のためのテスト実行を行い、記録したイベントに対するブラウジング、リプレイを行う。この方式では、ブレークポイントデバッグでは解析できないオブジェクト間通信レベルの実行時エラーをデバッグすることができる。

3. 3 イベント履歴を用いたプログラムのリプレイ

記録したイベント履歴によるプログラムの解析法として、ブラウジングとプログラムのリプレイを行う方式を採用する。

A-NETLデバッガにおけるプログラムのリプレイは、単にホスト計算機上でイベント履歴を実行順に表示していく方式ではなく、実際にプログラムをA-NET計算機上で再実行し、記録したイベント履歴の実行順序をもとに再実行時の振舞いを矯正してイベント記録時の実行と等価な実行過程を再現することにより実現される。

この方式では、デバッガがOSのスケジューリン

グに参与してシステム処理の負荷が大きくなるが、A-NETLプログラミングモデルにおいては、リプレイに必要な情報を含む事象が、メッセージ通信やオブジェクト内の実行メソッドの切り替え等の制御の流れを変える事象に限定される。また、情報量も、その事象が発生した際のスケジューリングに必要な情報に限定される。よって、記録するイベント履歴の事象とその情報量が少なくなり、イベント記録処理における負荷を軽くすることができる。

また、A-NETLデバッガでは、リプレイ時の各オブジェクトに対するブレークポイントデバッグ方式による解析機能を設ける。これにより、リプレイ時の各オブジェクトの詳細を観察することができる。

3. 4 デバッグ情報の表示法

高並列システムにおけるデバッグ処理においては、必要とされる情報量が非常に多く、大量の情報をユーザに理解しやすい形式で効率よく表示する技術が必要となる。この問題に対し、A-NETLデバッガでは、プログラムの振舞いに関する情報をマルチウィンドウを用いて3つのレベルに階層化し、各抽象化レベルごとに情報を表示するウィンドウを用意することで対応する。

4. A-NETLデバッガの機能

前節に述べた設計方針をもとに、A-NETLデバッガの基本構造と各モジュールの持つ機能を次のように定義した。

4. 1 基本構造

A-NETLデバッガは、分散デバッガモデルを採用している。その構成は、図-3 に示すようにローカ

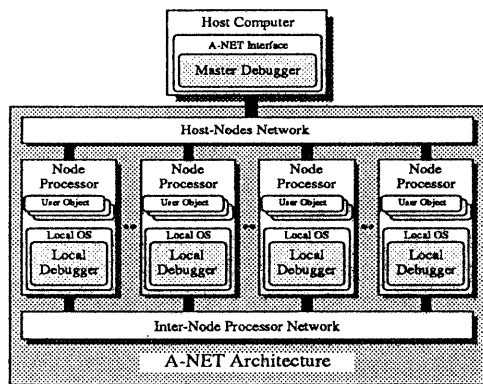


図-3 A-NETLデバッガの基本構造

ルデバッガとマスターデバッガから成る。マスターデバッガは、ホスト計算機上のA-NET計算機インターフェイスプログラムのサブモジュールである。ホスト計算機としては、UNIXワークステーションを想定しており、マスターデバッガはA-NETLプログラムの開発支援環境とともに、X Windowシステムを用いたアプリケーションプログラムとして実現され、ホスト計算機上で動作する。ローカルデバッガは、A-NET計算機の各ノードの要素プロセッサ(PE)に配置されるA-NET計算機のOSであるローカルOSのシステムメソッドと、PE内のデバッグ用マイクロプログラムにより実現される。

4. 2 ローカルデバッガ

図-4は、PE内の処理におけるローカルデバッガの位置づけを示したものである。ローカルデバッガは、各ノードのPEに対するブロープの役割だけでなく、PE上に割り付けられている各オブジェクトに対するリプレイ時の実行制御機能やブレイクポイントデバッグ機能を実現する。

A-NETLデバッガでは、ブレイクポイント検出とデバッグ処理への分岐をマイクロプログラムレベルで行なう。A-NET計算機のPEは、マイクロプログラム制御方式を採用しており、高機能な機械命令セットとなっている。マイクロプログラムレベルにデバッグ処理の一部を行わせることは、通常実行時においても常にデバッグ処理の実行の有無を確認するルーチンが走ることになる。しかし、プログラムコードにデバッグ処理用の特殊機械命令を挿入するよりも、機械命令境界においてマイクロプログラムレベルでデバッグ処理の一部を行なう方がイベントの記録やブレイクポイント検出処理のオーバーヘッドを抑えられると考えられる。また、デバッグ処理を行うためにプログラムコード自体を変更する必要がないので、デバッグ時と通常実行時の振舞いの差異をプログラムコードにデバッグ処理を挿入する方式よりも小さくできると考えられる。

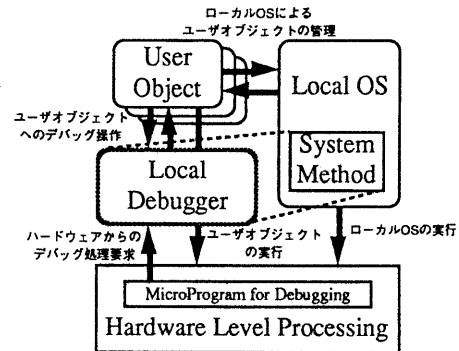


図-4 PEにおけるローカルデバッガの位置づけ

以下にローカルデバッガの持つ機能を示す。

(1) イベントの記録

図-5の(2)は、イベント記録のためのテスト実行時の各ノードのPE内の処理イメージである。このとき、ローカルデバッガは各ノードのPEに割り付けられたオブジェクトが関与したメッセージ通信やPE上の実行イメージの遷移などの事象をイベントとして記録し、各ノードごとに1つのファイルに格納する。記録したイベントは、マスターデバッガによりノード間のイベント発生時間の調整を行なってイベント履歴として整形され、履歴のブラウジングやリプレイの際に使用される。また、ローカルデバッガは、イベントを記録する際にイベント発生順のタイムスタンプを同時に記録し、前章において述べた論理時間の生成を支援する。

(2) リプレイ処理におけるプログラムの矯正再実行

図-5の(3)は、リプレイ実行時の各ノードのPE内の処理イメージを示している。リプレイを行なう際、ローカルデバッガはイベント記録時と等価なプログラムの振舞いを得るために、ローカルOSのスケジューリングに関与してプログラムを再実行させ、イベント履歴に記録されているイベントの発生順序をもとに再実行時の実行過程を矯正する。

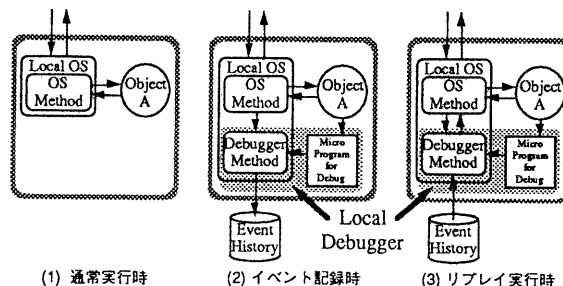


図-5 デバッグ時のPE内の処理イメージ

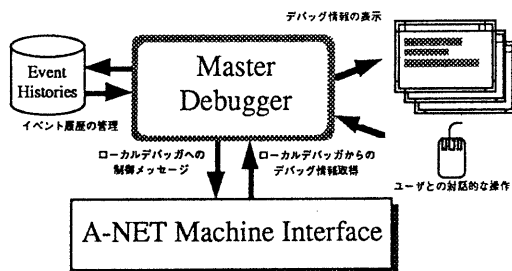


図-6 マスターデバッガの位置づけ

ローカルOSのスケジュールの単位は、ユーザオブジェクトのメソッドに対応する実行イメージである。実行イメージは、対応するメッセージを受信し、そのメッセージが実行可能な状態になったときに生成され、ローカルOSの制御下に置かれる。イベント記録時に、この実行イメージの遷移とメッセージ受信をイベントとして記録しているの、イベント履歴内のスケジューリングに関連するイベントの発生順に矯正することにより、各ノード内においてイベント記録時と等価な実行過程を得ることができる。

(3) ブレークポイントデバッグ

ローカルデバッガは、各オブジェクトに対するブレークポイントデバッガとしての機能も備えている。ブレークポイントデバッグ実行時には、ユーザがメソッドのソースコード1行分に対応するブレークポイントを設定するかステップトレースを行ない、対象メソッドのソースコードレベルの実行過程と変数の状態遷移を観察する。

4. 3 マスターデバッガ

図-6は、ホスト計算機におけるマスターデバッガの位置づけを示したものである。マスターデバッガは、デバッグ時のユーザインターフェイスであり、各ノードのローカルデバッガの制御やローカルデバッガから得られたデバッグ情報を収集して整理し、ユーザに提供する役割を果たす。

各ノードからのデバッグ情報を整理する際、マスターデバッガは情報をノード単位からオブジェクト単位に変換し、ハードウェアに関する情報を隠蔽してプログラムオリエンテッドなデバッグ情報をユーザに提供する。

以下にマスターデバッガの持つ機能を示す。

(1) 記録したイベントの編集

各ノードのローカルデバッガが記録したイベン

トデータから、プログラム全体における各イベントの発生順序を求める。メッセージ通信を行なったときのタイムスタンプから、全章で述べた方法でイベント発生時の各ノード間の論理時間を割り出す。

(2) ローカルデバッガの制御

マスターデバッガは、イベントの記録のためのテスト実行やリプレイにおけるプログラムの矯正再実行など、各ノードのローカルデバッガと協調して処理を行なう際に、各ノードのローカルデバッガとのメッセージ通信によりローカルデバッガを操作し、それらの状態を制御や必要なデバッグ情報の収集を行なう。

また、マスターデバッガは、各ノードのローカルデバッガに(1)において割り出した論理時間を単位として1論理時刻に対応する実行過程の再実行を制御させ、各ノードから得られたオブジェクトの状態に関する情報を表示する処理をユーザの制御のもとでインクリメンタルに実行していくことにより、リプレイを実現する。

(3) デバッグ情報の表示

マスターデバッガは、前章において述べた3タイプの情報表示ウィンドウをユーザに提供する。情報表示ウィンドウの詳細については、次章において述べる。

5. デバッグ情報の視覚化

A-NETLプログラムをデバッグするには、オブジェクトの内部状態に関する情報と、オブジェクト間の通信関係に関する情報のどちらか一方、または両方が必要となる。これに対し、A-NETLデバッガでは、ブレークポイントデバッグ時とイベントのリプレイ実行時の各オブジェクトの内部状態を観察するためのウィンドウと、イベントのリプレイ実行時に着目したい幾つかのオブジェクトの振舞いの変化を観察するためのウィンドウ、そしてイベント履歴からプログラム全体の実行過程の軌跡を観察するためのウィンドウの3種類のデバッグ情報表示用ウィンドウを用意する。

(1) オブジェクト内部状態表示ウィンドウ

図-7にオブジェクト内部状態表示ウィンドウのイメージを示す。このウィンドウでは、表示内容を1オブジェクトレベルの情報に限定し、各オブジェクトのメソッドの実行過程をソースコードのテキストレベルでの表示と、オブジェクトの変数状態の表示をユーザに提供する。

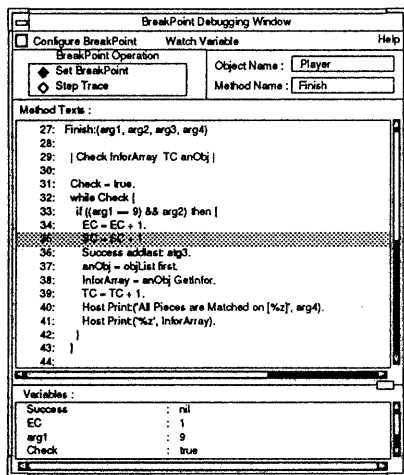


図-7 オブジェクト内部状態表示ウィンドウ

ユーザは、このウィンドウを用いてブレークポイントデバッグまたはリプレイ実行時の各オブジェクトに対するブレークポイントを設定する。図-7中のMethod Textsエリアは、ユーザが設定したソースコード1行分に対応するブレークポイントを表示している。また、Variablesエリアは、ユーザが解析対象として指定した変数の内容を表示している。ユーザは、トグルスイッチにより、プログラムのブレークポイントまでの実行させるか、ステップトレースを行うかを選択することができる。また、このウィンドウは各オブジェクトごとに開き、各オブジェクトに対してそれぞれのブレークポイントを設定することができる。

A-NETLにおけるプログラム記述単位はオブジェクトであり、オブジェクト内部の処理は逐次であるので、オブジェクトの内部表示には従来の逐次プログラムと同様のテキスト形式が有効であると考えられる。

(2) リプレイ用アニメーション表示ウィンドウ

図-8にリプレイ用アニメーション表示ウィンドウのイメージを示す。このウィンドウでは、表示内容をユーザが選択したオブジェクト数個レベルに限定し、リプレイ時におけるオブジェクトの状態を論理的な時間単位ごとのスナップショットとしてアニメーション的に表示する。

図-8のウィンドウ中のスクロールバーのある領域には、ユーザが指定したオブジェクトの実行過程のスナップショットが表示される。領域中の丸いグラフはオブジェクトを表し、濃色のグラフは、そのオブジェクトが観察している時点においてメソッド実行状態であることを示し、淡色のグラフは、オブ

ジェクトが休眠中であることを示している。また、矢印で表示されているグラフは、観察しているオブジェクトに対して送信されたメッセージを示している。ユーザは、各グラフをマウスで選択することにより、オブジェクト内で実行中のメソッド情報やメッセージの内容などを見ることができる。

このウィンドウは、リプレイ時に1つだけ開くことができ、リプレイ時のインターフェイスとなる。ユーザは、ウィンドウ下部のコントロールパネルによりリプレイの実行を制御することができる。

イベント履歴に基づくリプレイを行うことにより、基準時間単位での各オブジェクトの状態遷移に関する情報が得られる。この情報をユーザに提供するには、各オブジェクトの状態やメッセージ通信などのイベントを図形化し、その基準時間ごとの変化を動的に表示していく方法が有効であると考えられる。

また、この方法でプログラム中のすべてのオブジェクトを一度に表示しても、ユーザの混乱を招くと考えられる。よって、表示するオブジェクトは、ユーザが選択する方式を採る。ウィンドウ中のオブジェクトの配置についても、ユーザが調整して把握しやすいビューを構築する方式を採る。

(3) イベント履歴ブラウジング用ウィンドウ

図-9にイベント履歴ブラウジング用ウィンドウのイメージを示す。このウィンドウでは、記録したイベント履歴をもとに、プログラムの実行順序関係を論理的に割り出した時間概念と各オブジェクトによる2次元ダイアグラム表示により、プログラム全体のイベント記録時の振舞いに関する情報をユーザに提供する。

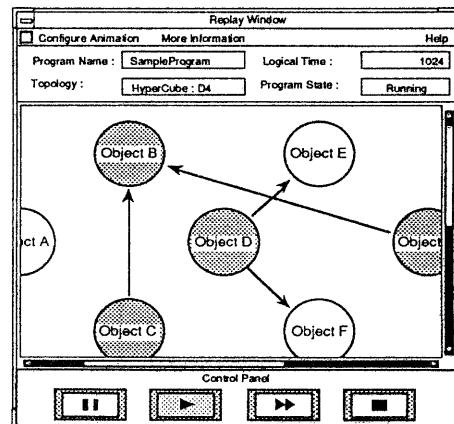


図-8 リプレイ用アニメーション表示ウィンドウ

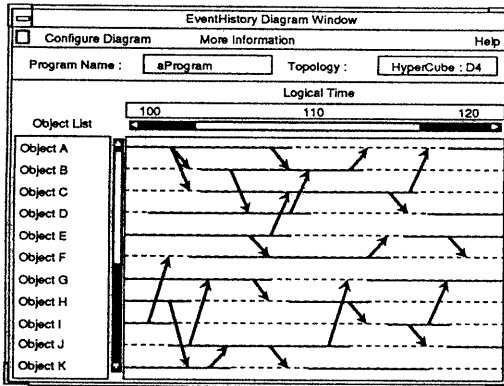


図-9 イベント履歴のブラウジング用ウィンドウ

図-9中のスクロールバーのある領域には、論理時間ごとのオブジェクトの実行過程のダイアグラムが表示されている。ダイアグラムの縦軸は、ユーザが選択した観察対象のオブジェクトを表し、横軸はプログラム全体の論理時間を表している。ダイアグラム中の矢印のグラフは、メッセージを示している。矢印の始点の位置は、メッセージを送信したオブジェクトと送信時の論理時刻に対応する座標であり、終点は、メッセージを受信したオブジェクトと受信時の論理時刻に対応する座標である。また、各オブジェクトに対応する軸に対応する直線は、オブジェクトが有る論理時間においてメソッドを実行中であることを示している。

ユーザは、ダイアグラム表示領域を操作して多数のオブジェクトの実行過程を表示させることができる。この際の縦軸のオブジェクトの配置は、ユーザが指定する。また、ユーザはグラフをマウスで指定することにより、グラフに関する詳細情報を表示するサブウィンドウを開くことができる。

記録したイベントをブラウジングしてプログラム全体レベルの振舞いを観察することにより、ユーザはデバッグ対象を絞り込むことができる。また、その際に必要な情報は、イベントの発生の有無とその概要程度で十分あると考えられる。よって、イベント履歴ブラウジング用ウィンドウでは、イベントをグラフ化して表示し、ユーザに選択によりメッセージの詳細情報やメソッドの実行状況に関する情報をサブウィンドウを用いて表示する方法を採る。

6. おわりに

本論文では、A-NETLデバッガの設計方針と基本的な機能について述べた。現在、A-NETLデバッガ

は、UNIXワークステーション上にプロトタイプを開発中であり、本論文で述べた並列オブジェクト指向プログラミングモデルに対するデバッグ手法を評価するまでには至っていない。今後は、A-NETLデバッガのプロトタイプを完成させ、A-NETLプログラムの振舞いを解析し、採用したデバッグ手法の評価、デバッグシステムとしての評価を行う予定である。

参考文献

- [1] T.Baba, et al. : "A Parallel Object-Oriented Total Architecture : A-NET", Proc. Supercomputing'90, pp.278-285 (1990).
- [2] T.Yoshinaga, and T.Baba : "A Local Operating System for the A-NET Parallel Object-Oriented Computer", J. Inf. Process (1991).
- [3] T.Yoshinaga, and T.Baba : "A Parallel Object-Oriented Language A-NETL and Its Programming Environment", Proc. COMPSAC'91, pp.189-196 (1991).
- [4] 片平 他 : "並列オブジェクト指向トータルアーキテクチャA-NETのプログラムデバッグシステム", 情報処理学会第43回大会, 2N-11 (1991).
- [5] C.E.McDowell and D.P.Helmbold : "Debugging Concurrent Programs", ACM Computing Surveys, 21, 4, pp.598-622 (1989).
- [6] 馬場敬信 : "超並列マシンへの道", 情報処理学会誌, Vol. 32, No. 4, pp.348-364 (1991).