マルチメディア通信と分散処理 71－2
（１９９５．７．１３）

オブジェクト指向分散環境 OZ++ のデバッガ

吉田 泰光[1*] 濱崎 陽一[2] 大西 雅夫[3*]
音川 英之[4*] 塚本 享治[2]
1: 日本ユニシス (株) 2: 電子技術総合研究所
3:(株) 東洋情報システム 4: シャープ (株)
*: 開放型基盤ソフトウェアつくば研究室研究員

OZ++ は分散環境で動作するソフトウェアの開発と実行を容易にすることを目指してた、オブジェクトの交換と共有に基づくオブジェクト指向分散環境である。また、OZ++ はソフトウェア資源の共有により、拡張が容易で自由度が高い分散システムを構築することができる。

一般に、分散環境下では不具合が観測された後に、システムを再び起動しても同じ不具合が観測される可能性が低いため、ソフトウェアのデバッグが難しい。この問題を解決するには、不具合を再現させるために必要な情報を常に収集する方法と、不具合の発生時点でその原因を追及する方法が考えられる。OZ++ のデバッガは、後者の方法を採った。

本論文では、OZ++ のデバッガの基本機能の設計と実装について述べる。

An Object-Oriented Distributed Environment: OZ++ Debugger

Yasumitsu Yoshida[1*] Yoichi Hamazaki[2] Masao Onishi[3*]
Hideyuki Otokawa[4*] Michiharu Tsukamoto[2]
1:Nihon Unisys, Ltd. 2:Electrotechnical Laboratory
3:Toyo Information Systems, Co., Ltd. 4:Sharp Corporation
*: Researcher, Tsukuba Laboratories, Open Fundamental Software Technology Project

OZ++ is an object-oriented environment that makes easy to develop and to execute distributed softwares, in which objects and classes can be shared and transferred. OZ++ provides more flexible and more extensible distributed systems by sharing software resources.

Generally in distributed environment, when a fault has been detected, it is almost impossible to detect same fault by executing the software again, and this is a reason of difficulty of debugging. There are two approaches to solve this problem. One is to log continually informations which are enought to replay previous faulty execution, the other is to investigate a cause of fault at the time of occurrence to provide debugging informations. We designed OZ++ Debugger by the latter way.

This paper represents the design and implementation of OZ++ Debugger's basic features.

1 はじめに

OZ++ は、ネットワーク上に広く分散した計算機において、オブジェクトとクラスを相互に利用し合うことを可能にする。また、OZ++ 言語により、オブジェクトのネットワーク上の位置に依存しないプログラミングを行うことができる。OZ++ のデバッガには言語と同等程度にオブジェクトのネットワーク上の位置に依存しないデバッグ機能が必要である。

また、一般に、分散システムのデバッグでは、正確な大域的状態の把握やデバッグ用の実行時情報の操作等が問題となる [1]。一方で、OZ++ は、サービスを提供するソフトウェアが運用されている環境の下でソフトウェアの開発を行うことができる。この運用形態でのデバッグ機能も必要である。

デバッグには大きく分けると

(1) プログラムを実行する前に行うデバッグ

(2) プログラムを実行して行うデバッグ

(3) プログラムを実行した後に行うデバッグ

の3つの方法がある。一般に、分散環境下では不具合が観測された後に、システムを再び起動しても同じ不具合が観測される可能性が低い。このため OZ++ のデバッガは、不具合の発生時点でその原因を追及するためのデバッグ情報を提供するという方針 (2) を採った。OZ++ のデバッガは、ソフトウェアをデバッグのために再起動せずに、不具合が発生した時点からデバッグを開始することを目指している。

一方、デバッグの作業は次の3つに分けられる [2]。

(a) 誤りの発見
　プログラムの動作を観察し、期待した動作との食い違いを発見する。

(b) 誤り位置の特定
　異常な動作をするプログラムの位置を特定する。

(c) 誤りの修正
　異常な動作をするプログラムの部分を修正する。

OZ++ のデバッガは (a) と (b) の作業を支援をする。

OZ++ システムでは、OZ++ のデバッガと他のプログラミングを支援するツールにより (1) や (c) を行うことができる。

本論文では、2節で OZ++ の概要を述べ、3節と 4節で OZ++ のデバッガの基本機能とその実現方法について説明する。

2 OZ++ の概要

OZ++ はネットワーク上に分散したオブジェクトが互いに通信し合うことで計算が進むオブジェクト指向分散環境である [3]。OZ++ は、ネットワークを流れるメッセージの中にオブジェクトを含めることができる点、更に、そのオブジェクトのクラスもネットワークを通じて供給する機構を持つ点に特徴がある。

既存ＯＳ上で動作する OZ++ システムは、プログラミング言語としての OZ++ 言語、分散環境上で利用できるクラスライブラリ、ブラウザ、デバッガなどのプログラミング環境を提供する。

2.1 OZ++ 言語

OZ++ 言語は、OZ++ システムにおいてネットワーク上のオブジェクトの位置に依存しない（位置透過）ソフトウェアの記述を容易にするオブジェクト指向プログラミング言語である [4]。C言語をベースにし、オブジェクト指向、並列プログラミングなどの言語機能を取り入れている。

OZ++ システムのオブジェクトには、大域的に識別可能なＩＤを持ちサービスの提供を行うためのグローバルオブジェクトと、グローバルオブジェクトの実装に使われるローカルオブジェクトとがある。一つのグローバルオブジェクトと任意の数のローカルオブジェクトの集まりをセルと呼ぶ（図 1）。一つのオブジェクトは一つのセルにのみ属する。

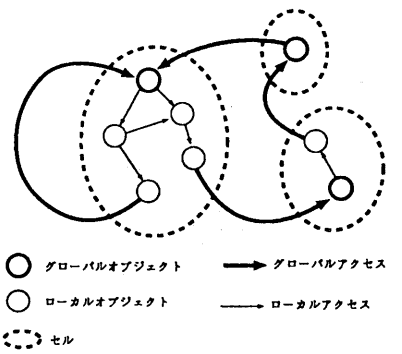

図 1: OZ++ 言語におけるオブジェクト

2.2 OZ++ ランタイムシステム

オブジェクト管理システムやオブジェクトを実行するメ

カニズムがランタイムカーネルである。オブジェクトは、ランタイムカーネルが供給する多数のランタイムルーチンを呼び出す形で処理を実行する。この OZ++ システムのランタイムカーネルをエグゼキュータと呼ぶ。

エグゼキュータは次のような機能を持つ。

スレッド管理 オブジェクトの実行可能コードは複数のスレッドで実行され、エグゼキュータはそれらのスレッドの生成、実行、消滅を管理する。

オブジェクトコード管理 オブジェクトのコードは、それが必要になった時に、ネットワークを通じて配送され、エグゼキュータにダイナミックロードされる。

メモリ管理 オブジェクトのためのメモリを確保し、ガベージコレクションを行う。

通信機能 オブジェクトが互いにメソッド起動を行う機能を提供する。

2.3 オブジェクト管理システム

OZ++ システムでは、ネットワーク上の各種サービスはもちろん、オブジェクトの基本的な振る舞いを管理する部分についても、すべて、OZ++ 言語で記述されたオブジェクトが行う。これらのオブジェクトの動作に必須であるいくつかのオブジェクトを総称してオブジェクト管理システムと呼ぶ [5]。また、それらのオブジェクト中でエグゼキュータとのインターフェースの役割を果たすオブジェクトを特にオブジェクトマネージャ（OM）と呼ぶ。

3 OZ++ のデバッガ

既に述べたように、OZ++ のデバッガはランタイムデバッグの方法を採り、誤りの発見と誤り位置の特定を支援するために次に述べる基本機能を持っている。

3.1 オブジェクトのインスペクト

OZ++ のデバッガは、OZ++ ソースコードのコンパイル時に生成されたインスタンス変数のシンボル情報と、エグゼキュータから読み出したオブジェクトのイメージデータとから、オブジェクトのインスタンス変数を表示する。

3.2 プロセスのインスペクト

OZ++ 言語では、メソッド呼出しによるスレッドの連鎖をプロセスと呼ぶ。更に、OZ++ のデバッガではスレッドの連鎖によって発生する一連のスタックの集まりをプロセススタックと呼ぶ。プロセスのインスペクト時にはそれらのスレッドのスタックを統一的に操作する事ができる。

また、プロセスの実行状態を図 2のようにモデル化している。OZ++ のデバッガは、ネットワーク上に分散した複数のスレッドから成るプロセスのスタックの内容を表示したり、プロセスが図 2中のどの実行状態にあるかを調べることができる。

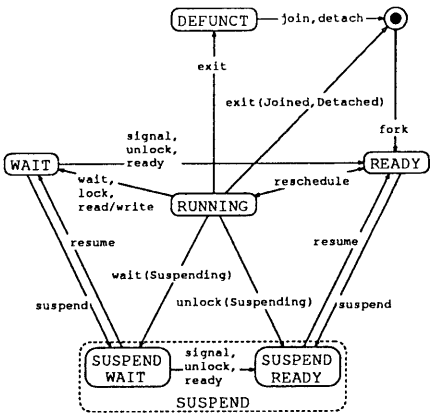

図 2: プロセスの実行状態の遷移モデル

3.3 例外の捕捉

OZ++ 言語 では、プロセスを構成するメソッド呼出しの連鎖における大域脱出として例外の発生と例外処理を記述することができ、ネットワーク上を越えて任意のオブジェクト間で例外の受け渡しが可能である [6]。

OZ++ システムでは、例外処理は例外の発生によって生じた状態を正常な状態にするための特別な処理であると位置付けている。例外の発生によって生じた状態は、必ずしもデバッグの対象であるとは限らないが、デバッグの手掛かりとしては重要な情報である。一般に、例外処理を記述することは難しい。その結果、例外処理の記述がない例外が発生することになり、意図した結果を得られない。OZ++ のデバッガは、例外の発生の位置にかかわらず次の例外の発生を捕捉し、デバッグを開始できる。

- 例外処理の記述のない例外の発生

- デバッガから指定した例外の発生

3.4 デバッグ文の制御

一般に、実行しようとしているプログラムのデバッグを行う際の最も基本的な手法は、予めポイントとなる位置にデバッグメッセージなどのデバッグのためのコード（以降ではデバッグコードと呼ぶことにする）を組み込むことである [7]。

通常、ソフトウェアの開発過程では、なんらかのデバッグコードが存在している。そして、開発が終了するとそれらのデバッグコードは取り除かれる場合が多い。ソフトウェアの開発過程の初期段階のデバッグコードは、そのソフトウェアの作成者以外の者にとって特に意味あるものではない。しかし、ソフトウェアの開発の最終段階のデバッグコードは、そのソフトウェアの動作をチェックするという意味合いが強い。この最終段階のデバッグコードはそのソフトウェアを利用した新たなソフトウェアのデバッグにも有効であると考えることができる。つまり、デバッグコードもソフトウェアの一部であると考えることができる。

運用形態のデバッグを可能にするために、OZ++ システムでは、言語にデバッグコードのためのデバッグ文を導入した。予めデバッグ文をクラスのプログラム中に記述し、OZ++ のデバッガがプログラムの実行時にデバッグ文を実行するか否かを制御する。

OZ++ システムでは、クラスの版管理 [8] の機構を用いてデバッグコードがある版のクラスとない版のクラスを作成することによりデバッグコード方式のデバッグが可能であるが、既に実行されているオブジェクトのクラスをダイナミックに変更することはできない。

4 OZ++ のデバッガの実現

3節で述べた OZ++ のデバッガの基本機能を実現するための OZ++ システムの機能と OZ++ のデバッガ自体の処理について説明する。

また、一般に、分散システムでは一度に送受信されるデータ量の抑制、送信側と受信側のデータの整合性の保持が求められる。OZ++ のデバッガでは、オブジェクトのインスペクト時はオブジェクトのイメージ全体を一度に表示せず一定のまとまりに分けて表示し、プロセスのインスペクト時はスレッドのスタック毎に分けて表示することにより、送受信されるデータ量の抑制を行っている。データの整合性の保持は、オブジェクト、あるいは、プロセスを一時停止することにより実現している。

4.1 構成

OZ++ のデバッガは、次の部分により構成されている（図 3）。

- デバッガ（グローバルオブジェクト）
  インスタンス変数のシンボル情報やメソッドのシンボル情報等の静的な情報をオブジェクト管理システムから獲得し、エグゼキュータの組み込み部から得た動的な情報と合わせてオブジェクトのインスペクトやプロセスのインスペクトに必要な表示用のデータを作成する。

- エグゼキュータの組み込み部
  エグゼキュータに対する低レベルの基本処理を行うスレッドである。スレッドの一時停止、オブジェクトの一時停止、メモリ（スレッドのスタック、オブジェクトのイメージ等）の読み書き、イベント（例外の発生, ダイナミックロード等）の捕捉など、主に動的に変化する情報の処理を行う。デバッガの要求に応じて、これらの処理は複数のスレッドで処理される。

- デバッガフロントエンド
  デバッガに対して使用者からの指示を伝えたり、デバッガから得た表示用のデータを実際に表示する処理を行う。

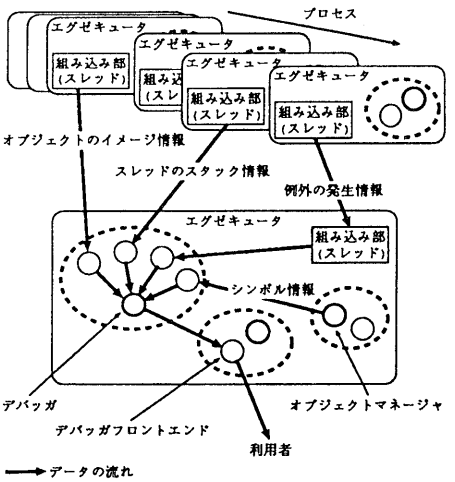

図 3: OZ++ のデバッガの構成とデータの流れ

4.2 デバッグフラグ

運用形態でのデバッグを可能とするためにプロセス、オブジェクト、クラスの実行可能コードはデバッグフラグを持つ。デバッグフラグは次に挙げるフラグの集合である。

- デバッグ文を実行するか
- 例外の発生を OZ++ のデバッガで捕捉するか
- プロセスの生成時にデバッグフラグの状態を伝播するか
- オブジェクトの複製時にデバッグフラグの状態を保存するか
- 当該オブジェクトに対するメソッドの起動時に、当該スレッドを一時停止するか

デバッグフラグは OZ++ のデバッガによって変更され、メソッドの起動、例外の発生、オブジェクトの複製時に参照される。このため、通常時のデバッグフラグの参照の機会が多くなり実行速度の低下が懸念される。しかし、デバッグフラグを実行に必然的に必要となるオブジェクトのイメージやオブジェクトコードの内部に配置し、デバッグ時以外にデバッグフラグを処理するための関数を呼び出さないようにすることにより極度の実行速度の低下は防止できると考えている。

4.3 スレッドの一時停止

オブジェクトの一時停止やプロセスの一時停止を行うためにはスレッドを一時停止する必要がある。この時、エグゼキュータ上で共有している資源を操作中のスレッドを一時停止すると、その資源を操作しようとしている他のスレッドは待ちの状態になり、最終的にはエグゼキュータ上の全てのサービスが停止してしまう可能性がある。この状態では、エグゼキュータ上のスレッドとして動作している OZ++ のデバッガのエグゼキュータ組み込み部も待ちの状態になり、一時停止したスレッドの再開もできなくなる。このため OZ++ システムでは、エグゼキュータ上で共有している資源を操作するスレッドに対する一時停止処理は、その操作が終了するまで遅延される必要がある。

エグゼキュータの各ランタイムルーチンは、これらの共有資源の操作時に一時停止処理をブロックする。一時停止処理は、一時停止のブロック解除と同時に行われる。一時停止をブロックしたスレッドが待ちの状態になった場合は、そのスレッドは一時停止の状態として扱われる。

4.4 オブジェクトの一時停止

オブジェクトのインスペクトを行う場合、OZ++ のデバッガはオブジェクトを一時停止、すなわち、オブジェクトを参照する全ての スレッドを一時停止する。

4.5 プロセスの一時停止

プロセスをインスペクトするには、プロセスを一時停止する必要がある。OZ++ システムでは、プロセスは、メソッド呼出しによるスレッドの連鎖であり、それらのスレッドはネットワーク上に分散して存在している。OZ++ のデバッガは、このスレッドの連鎖を各スレッドの一時停止／再開を繰り返しながら順次追跡し、連鎖の終点であるスレッドを一時停止することによりプロセスの一時停止を行っている。

4.6 例外の捕捉

OZ++ システムの例外は、スレッドのスタック内にコンテキスト、例外ＩＤ、例外処理に引き渡すパラメータの情報から成る例外フレームを埋め込み、例外の発生時に例外フレームを使ってコンテキストの切り替えを行うことで実現している [9]。例外フレームは捕捉する例外のリストを持っている。このために、例外の発生時にその例外が最終的に捕捉されるか否かを判断することができる [9]。この判断の過程で、デバッグフラグの指示に応じてプロセスを生成したエグゼキュータを処理対象としている OZ++ のデバッガに例外の発生が通知される。OZ++ のデバッガの応答を待ってコンテキストの切り替え等の処理が行われる。但し、デバッグフラグの指示がない場合は OZ++ のデバッガに例外の発生は通知されない。

4.7 デバッグ文の制御

メソッドが起動されると、オブジェクト、プロセス、クラスの実行可能コードの各デバッグフラグを参照し、デバッグフラグが設定されているかを調べる。デバッグフラグが設定されていると、デバッグ文を実行するか否かの制御を行うランタイムルーチンを呼び出す。このランタイムルーチンは OZ++ のデバッガへの問い合わせや詳細なデバッグ条件のチェックを行う。

5 課題

- オブジェクトのレベルでの一時停止のブロック
  OZ++ のデバッガは OZ++ システム上で動作し、オブジェクトのインスペクト時に使用するインスタンス変数のシンボル情報、プロセスのインスペクト時に

使用するメソッドのシンボル情報や例外のシンボル情報をOMを介してオブジェクト管理システムから得る。従って、OZ++ のデバッガの次のような副作用を伴うデバッグはできない。

- オブジェクト管理システムを一時停止させてしまうようなデバッグ
- オブジェクト管理システムのクリティカルセクションを実行中のプロセスを一時停止するようなデバッグ

また、クリティカルセクションを実行中のプロセスやオブジェクトを一時停止させることによるサービスの停止が考えられる。エグゼキュータ内部の処理では、一時停止をブロックすることができるので、オブジェクト単位での一時停止のブロック等を行うことができれば、これらの問題を避けることができる。

- 分散システムに対応したブレークポイントの設置
  一般に、デバッグの対象に対する知識が十分にある場合は、ブレークポイントの設置によるデバッグは有力なデバッグの方法である。OZ++ のデバッガでは、メソッドのレベル、あるいは、OZ++ 言語のレベルのブレークポイントの導入を考えている。

- 誤りの修正
  1節では述べなかったが、実行可能コードの二重化することにより、プログラムにダイナミックな変更を加え、一次的な誤りの修正を行うデバッグ機能の導入を考えている。

6 おわりに

オブジェクト指向分散環境OZ++ において、位置透過なデバッグと運用形態でのデバッグを可能にするOZ++ のデバッガの基本機能とその実現方法を述べた。現在、これらの基本機能を実装するためにOZ++ 言語の仕様追加とオブジェクト管理システム、エグゼキュータの拡張／改良の作業を行っている。今後、実際にOZ++ システムでのプログラム開発を行いながら、5節で述べた機能の実装と、これらの基本機能を基にしたより高度なデバッグ機能の検討を行う予定である。現在、OZ++ の基本システムが公開されている（http://www.ipa.go.jp/OFSTP/obj.html）。

謝辞

本研究を通じて，熱心な討論をいただいている，千葉滋氏(東京大学理学部)，および当プロジェクトのメンバー諸氏に感謝する。

この研究は情報処理振興事業協会（ＩＰＡ）が実施している「開放型基盤ソフトウェア研究開発評価事業」の一環として行われたものである。

参考文献

[1] W.H.Cheung, J.P.Black and E.Manning, A Framework for Distributed Debugging, IEEE Software, Jan. 1990

[2] R.L.Sedlmeyer and W.B.Thompson, Knowledge-based Fault Localization in Debugging, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging, Mar. 1983

[3] 塚本他, 「オブジェクト指向分散環境 OZ++ の基本設計」, 情報処理学会研究報告 93-OS-61-3(SWoPP 軺の浦), Aug. 1993.

[4] 西岡他, 「オブジェクト指向分散環境 OZ++ の言語の基本設計」, 情報処理学会第 46 回全国大会, Mar. 1993.

[5] 西岡他, 「オブジェクト指向分散環境 OZ++ のオブジェクト管理系の設計」, 情報処理学会研究報告 94-DPS-66, Jul. 1994.

[6] 音川他, 「オブジェクト指向分散環境 OZ++ の言語における例外処理の記述」, 情報処理学会第 50 回全国大会, Mar. 1995.

[7] K.Araki, Z.Furukawa, and J.Cheng, A General Framework for Debugging, IEEE Software, May. 1991

[8] 新部他, 「OZ++ コンパイラによるクラスの版管理」, 情報処理学会研究報告 SWoPP'94, Jul. 1994.

[9] 濱崎他, 「オブジェクト指向分散環境 OZ++ のプロセスと例外処理の実装」, 情報処理学会第 50 回全国大会, Mar. 1995.