オペレーティング・システム 51−10
（１９９１．７．１８）

ディペンダブルな分散型実時間処理システムに向けて

後藤 厚宏　山崎 憲一　鷲坂 光一
ＮＴＴソフトウェア研究所

現在の分散プラットフォームが提供できる応用インタフェースは、社会的に頼りがいのある応用システムを作る上では不十分な点が多い。また、応用システムは時間制約や地理的条件をシステム内に導入できる柔軟な制御機能を必要とする。このため、分散カーネル（カーネルウェア）と応用システムの間のミドルウェアの役割が重要となる。ミドルウェアにおいては、分散環境におけるアトミックな操作を支援する同期機構とトランザクション機能、ディペンダビリティのためのロギングとリカバリを標準インタフェースで提供する。さらに、クライアント／サーバーのパラダイムとともに、分散デバッグやモニターを支援するパラダイムとして、制御プログラムがセンサーとアクチュエイタを用いて環境を操作するリアクティブパラダイムが重要である。

Toward a Highly Dependable Distributed Realtime Systems

Atsuhiro Goto　Kenichi Yamazaki　Mitsukazu Washisaka
NTT Software Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180, Japan

Recent technological advances have made it cost-effective to construct large systems from collections of computers connected via networks. However, current distributed platforms are not enough to support dependable application systems for our society, which need for effective ways to support realtime constraints and geographical distribution.

Therefore, *Middleware*, which plays an important role between *Kernelware* and distributed applications, should facilitate various synchronizations and transaction primitives to support atomic operations in distributed environment. In addition, reactive paradigm where control programs handle distributed environments by sensors and actuators, is effective as well as client/server paradigm.

1 はじめに

ワークステーションおよびネットワークの普及と高速化という物理的環境の進歩により、社会的に重要性の高い応用システムを分散プラットフォーム上に構築しようとする動きが急である。しかし、現在の分散プラットフォームが提供できる応用インタフェースは、頼りがいのある分散処理システムを作る上で不十分な点が多い。また、応用システムは時間制約や地理的条件をシステム内に導入できるような柔軟な制御機能を必要とする。

我々は、金融や発券のような実社会における事務処理や協調作業の基盤とすることができる「頼りがいのある分散型実時間処理アーキテクチャ／環境 *Hydrae* [1]」を検討中である。*Hydrae* では、現在、標準化が進められている UNIX ベースの分散カーネル（カーネルウェア）と応用システムの間のミドルウェア (図 1) の役割に注目し、

- 実時間制約に対応できる分散トランザクション処理
- 分散型実時間システムのための**スケーラブル**なプロトタイプの設計／構築支援

の実現を目指している。

本稿では、今後の分散処理システムの要求機能から *Hydrae* 全体の技術課題およびミドルウェアにおけるトランザクションパラダイムについて検討する。

分散環境の仮定: ノードとネットワークに関しては以下のような仮定を置く。

分散プラットフォームの標準化の動きは、マルチベンダーシステムを可能とする。このため、応用システムで使用されるノードの種類は必ずしも同一ではない (異種性: heterogeneity)。さらに、ノードは単一のプロセッサからなる小型のものから並列プロセッサによる高性能なものまで多岐に渡ることになる。また、分散環境は、ある応用システムに向けて新たに構築されることよりも既に組織または社会において自立的に使用されているノード計算機を相互に接続して作り上げられることの方が一般的である (自立性: autonomy)。

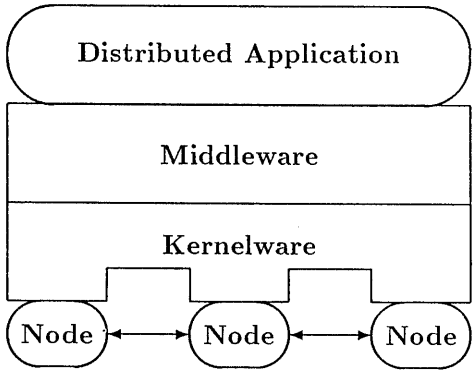


図 1: 分散処理システム

ノード間のネットワークは、LAN と WAN および両者の組み合わせであるとする。特に、大陸間規模の広域網を含むとすると、時差／国情のような社会的要因も分散処理システムに影響を与える。

さらに、そのような広域分散環境では、ノードおよびネットワークの信頼性は必ずしも十分であるとは限らない。

2 分散処理の応用領域からの要求

分散処理の応用システムは、分散環境に対して、高い性能とディペンダビリティの両立を要求する。

アベイラビリティ: ディペンダブルなシステムの第 1 条件は、ノード計算機やネットワークの一部が故障したとしても、可能な限り連続的なサービスを提供できることである。つまり、故障の範囲をできるだけ限定し、プログラムが通信しようとするノードの機能が生きており、また、それに到達できる限り実行を続けられることが求められる。このためには、分散環境自身がシステム状態のモニターとリカバリの機能を持ち、必要に応じてサーバーをマイグレートすることが必要となる。

[1] Hydrae[haidri:] – Highly Dependable Distributed Real-time Architecture and Environment

並行性と動的なシステム再構成: 分散環境に応用システムを実装する目的のひとつは、応用システムにおける本来の並行性を十分に引き出すことにある。このため、分散環境は、ハードウェアの追加によって性能の向上／応答時間の短縮／アベイラビリティの向上を容易に可能とするものでなければならない。特に、連続的なサービスを維持するためには、システムを稼働させたままで、論理的／物理的双方の変更ができることが求められる。

モジュラリティ: データと処理の分散によって、応答性とコスト効果の両面で全体の効率を上げ、さらに、システムのアベイラビリティを高めるためには、分散環境はモジュラリティのある構成でなければならない。

一貫性: ほとんどのシステムにおいて、操作されるデータの一貫性を保つことが必要である。それらの一貫性制約は、個々のデータについてだけでなく、データの集合に対しても適用される。このため、一貫性の保持を基本とするトランザクションをパラダイムとしたシステム設計が重要である。

スケーラブルなプロトタイピング: 分散処理システムのプログラミングにおける最大の難点はプロトタイピング／デバッグ／試験の難しさである。特に、実時間制約があるような場合はターゲットシステムによる高価なテストに依存してしまう。このため、今後の分散処理システムの構築にあたっては、スケーラブルなプロトタイピング、つまり、プロトタイプによって時間制約が満たされていれば、実システムでも満たされ、また、プロトタイプで大規模システムの耐故障性が調べられるような枠組が必須である。

3 *Hydrae* の技術課題

「頼りがいのある分散型実時間処理アーキテクチャ／環境 *Hydrae* 」の主眼は、実時間制約に対応できる分散トランザクション処理と分散型実時間システムのためのスケーラブルなプロトタイプの設計／構築支援にある。図2に *Hydrae* の技術課題を示す。

3.1 ノードとネットワーク

分散処理システムのベースは、耐故障性の高いノード計算機と高速通信リンクである。ノード計算機としては、ハードウェアの冗長性およびマルチプロセッサ構成によるフォールトトレラント計算機を用いる。ただし、全てのノード計算機の耐故障性が十分であるわけではなく、また、それぞれのノードの自立性により、運用管理が統一的になされるわけではないことを前提とする必要がある。

ネットワークは、社会的／地理的要因に強く影響される。公衆網を含むネットワークにおいてはリンクの信頼性／安全性がそれほど期待できないことを考慮しなくてはならない。また、今後は ATM(非同期転送モード) を用いて動画のような負荷の大きいメディア通信が分散環境に導入されることへの対応が必要である。

3.2 カーネルウエア (Kernelware)

UNIX がワークステーションの標準 OS のひとつとして定着して以来、Mach [1] や Chorus [13] のように、UNIX の分散処理機能の拡張やマルチプロセッサ対応への拡張が盛んである [14]。また、分散環境における相互接続性は標準化と表裏一体であることから、標準化 (POSIX など) またはデファクト標準を目指す動き (OSF, UI など) も活発である。カーネルウェアは、このような分散 OS または分散プラットフォームと呼ぶものに相当する。

(1) 分散実時間カーネル

カーネルウェアにおいては、Mach や Chorus を見て分かるように、カーネルを小さく単純化してモジュラリティを高めるとともに、密結合マルチプロセッサ計算機と疎結合ネットワーク環境の 2 種類の環境に対応する必要がある。スレッド (thread) による並行／並列処理、および遠隔手続き呼び出し (RPC)、安全な名前サービスが分散処理の支援機能として重要である。

(2) ディペンダブルな通信プリミティブ

分散プログラミングを複雑で困難にしている主な原因は、メッセージの到着や到着順が保証されていない

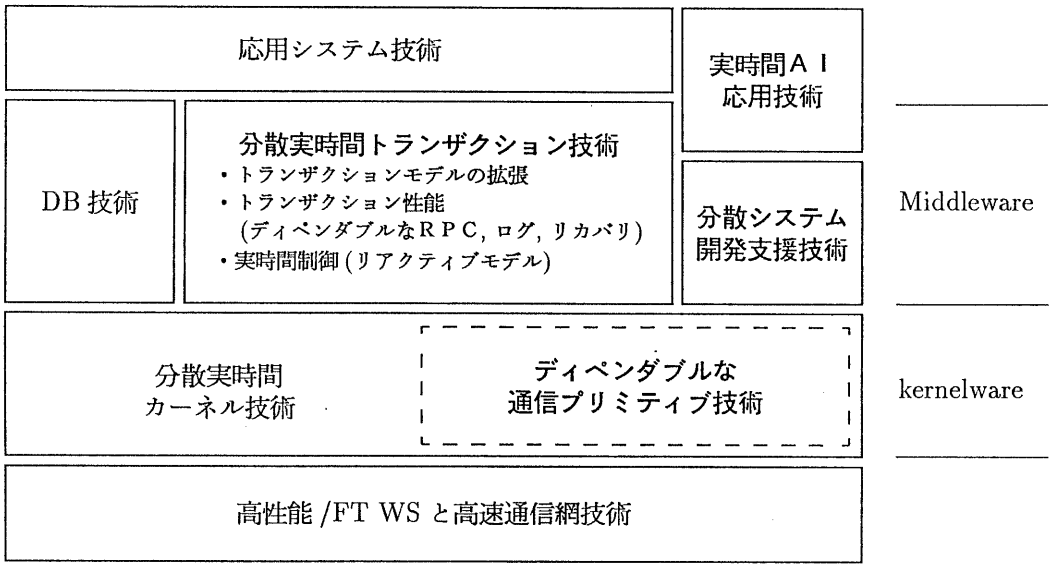


図 2: ディペンダブルな分散型実時間処理システムの技術課題

こととと、タイムアウトでリプライが帰ってこない際に相手が過負荷なのか故障しているのかがわからないことである。このような問題を解決するプログラミングツールセットとしてはISIS [4] が提案されている。ISISは仮想同期 (virtual synchrony) [3] によりプロセスグループに対するアトミックマルチキャストを実現する。ただし、現状では性能面の問題があるため、カーネルウェアに組み込む必要があろう。また、end-to-endで実時間制約を満たすことができる通信プロトコルと広域網において時間制約を柔軟に設定できるプログラミング支援が重要である。

3.3 ミドルウェア (Middleware)

ミドルウェアは、分散実時間カーネルOS上で高位のアプリケーションインタフェースを提供するものであり、従来の汎用計算機システムにおけるDB/DCシステムやOSF/DCE、UI/ONCに相当する。*Hydrae* におけるミドルウェアでは、分散実時間トランザクション技術、分散システム開発支援技術、実時間AI応用技術、が重要な課題となる。

(1) 分散実時間トランザクション技術: トランザクション処理はクレジットカードの照会、銀行のＡＴＭ、チケット予約など、ビジネス分野の大部分の応用システムにおいて基盤となる技術であり、社会的依存度は増える一方である。また、産業的にみても、トランザクション処理関連の総売り上げは近年急速に伸びており、全コンピュータ関連売り上げにおける割合が5割を越える勢いであると報告[2]されている。

トランザクションを、これまでの応用システムだけに限らず、より広範囲の分散処理システムをディペンダブルなものにするための基本パラダイムとしていくことが重要である。その時の課題は:

- トランザクションモデルの拡張
- トランザクション性能
- 実時間制御

であり、これらについて4章で考える。

[2]DataQuest Report

(2) 分散システム開発支援技術: 前述(2章)したように、分散処理システムのプログラミングにおける最大の難点はプロトタイピング／デバッグ／試験の難しさにある。このような分散システム開発を支援する技術は未成熟であり難しい課題が多い。特に、時間制約を分散プログラミングにおいて柔軟に指定できるトランザクション記述言語が課題である。

分散ターゲット上でのデバッグにおいては、事象間の因果関係の把握と時間制約の判定が重要である。前者については、トランザクションのパラダイムがデバッガーによるシステム制御に利用できる。また、後者については4.4節で述べるリアクティブモデルが有用であろう。

(3) 実時間AI応用技術: 分散環境でのジョブ制御／負荷制御を行い、システムのアベイラビリティと性能を高めるためには各種のヒューリスティクスを用いた分散スケジューリングが重要である。このような実時間AIは、応用システムにおける時間制約や通信コスト制約のある探索を行う技術としても注目できる。

4 トランザクションパラダイム

4.1 トランザクション

トランザクションは分散処理システムの構築を単純化する上で重要な概念である。直観的には、トランザクションはアトミックな仕事の単位であり、その実行は、最後まで実行されてコミットするか、まったく実行されなかったかのようにアボートするかのいずれかとなる。そのようなトランザクションに必要な性質[2]は次の4点にまとめられ、ＡＣＩＤ特性(ACIDity)と呼ばれる。

Atomicity(原子性): トランザクション処理の操作は、全て実行されるか、まったく実行されないかのいずれかである。つまり、トランザクションが失敗したときは、その途中結果は取り消されなければならない。

Consistency(一貫性): ひとつの独立したトランザクションが、一貫性のあるデータに対して実行されたときは、その結果も一貫性のあるものでなければならない.

Isolation(分離性): 実行中のトランザクションは、それがコミットされるまでは、その途中結果を他のトランザクションに見せてはいけない。

Durability(永続性): 一旦、トランザクションがコミットされたら、システムはそのトランザクション操作の結果が失われないように保証しなければならない。

4.2 トランザクションモデルの拡張

これまで銀行システムなどで用いられてきたトランザクションシステムは、大型汎用機を用いた集中型システムであり、個々のトランザクション処理も単純なデータベースの更新処理がほとんどであった。しかし、トランザクションを分散処理の基本パラダイムとして用いて、CAD、グループウェア、協調処理などの広い範囲の応用に適用したり、また、異種性のある分散システム環境を想定する場合、従来型ソリッドなトランザクションモデルだけでは十分でなく、より柔軟なものに拡張する必要がある。特に、タイムアウトのような時間制約を自然な形で扱えるトランザクションモデルが重要である。

(1) ネスティドトランザクション: (親)トランザクションの中から別のトランザクション(サブトランザクション)を発することを可能にするネスティドトランザクション(Nested Transaction [12])が、トランザクションをビルディングブロックとして分散システムを構築するための基本的な枠組である。

ただし、このようなトランザクションのネストを可能とするためには、資源のロック状態などの実行環境を親トランザクションとサブトランザクションの間で相互に受け渡す機構とコミットプロトコルが必要となり、その実行コストは必ずしも小さくない。トランザクションを広域網で用いたり、また、ＣＡＤにおける共同作業のような協調処理に用いる場合、ひとつのトランザクションが長時間に渡ることになる。このような長時間に渡るトランザクションのすべてをネスティ

ドトランザクションとすると、全体がコミットするまで個々のステップでアクセスする資源を解放できないため、並行性が低く、実行コストが大きくなってしまう。そこで、ネスティドトランザクションにおいて前述のＡＣＩＤ特性の一部を緩和したトランザクションモデルを併用することを考える。

(2) 補償トランザクション　：既にコミットされたトランザクションの効果を打ち消すために発するトランザクションを補償 (compensating) トランザクションと呼ぶ [7, 8]。親トランザクションの中で発するサブトランザクションが補償可能 (compensatable)、つまり、逆操作に相当する補償トランザクションによってロールバックできる場合、親トランザクションの実行途中でそのようなサブトランザクションを順次コミットし資源も解放する。もし、親トランザクション全体をアボートするときは、既にコミットしているサブトランザクションに対応する補償トランザクションを発する。これは、親トランザクションの分離性を緩和したことに相当する。もちろん、補償トランザクションを発することによってロールバックを実現できるかどうかは応用依存である。補償トランザクションは、トランザクションの並行性を高めると共に、トランザクション管理システムの実現コストを抑える手段として重要である。

(3) 陽なトランザクション制御　：分散処理の応用において、データの一貫性制約をチェックする処理やトランザクションによって更新された内容をレプリカに伝える処理は、陽にトランザクションの最後にスケジュールしたい。また、トランザクション中のある事象の生起をトリガーとして別のトランザクションを元のトランザクションの外側で起動したい場合もある[3]。このためには、前述のネスティドトランザクションを目的に応じて制御することが必要であり、そのひとつとして後述のリアクティブモデルが必要となる。

[3] アクティブデータベースを対象としたトランザクションモデルの提案が [5] にある。

4.3　ディペンダブルなRPC

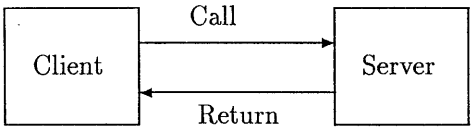
分散環境におけるトランザクション機能は、これまで、CMU の Camelot/Avalon [6]、MIT の Argus [9] で検討されている。分散トランザクションモニタにおいては、トランザクション性能とアベイラビリティを決める分散ロギング、および、前述のトランザクションモデルの拡張に伴う分散コミットプロトコルの設計が鍵となる。

一方、遠隔手続き呼び出し (RPC) は、ネットワークに分散するサービスの利用を単純化できる強力なパラダイムであり、ＵＮＩＸ系の分散環境においてはＲＰＣ技術によって分散アプリケーションの開発期間が短縮できるようになってきた。しかし、現状のＲＰＣでは、ＲＰＣの失敗のセマンティクスが明確でなく、クライアントはタイムアウトになった時、ＲＰＣが実行されたのかどうかはっきりわからない (at-most-once semantics)。このため、通常のＵＮＩＸにおけるＲＰＣをトランザクションの中で用いることが難しかった。

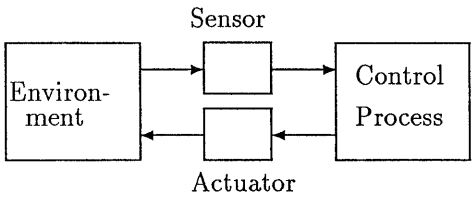
そこで、信頼性の高い分散処理システムを構築するための基本パラダイムとしてＲＰＣを用いるために、ＲＰＣの失敗のセマンティクスを明確にする。つまり、クライアントが発したＲＰＣがタイムアウトになった時、クライアントは、そのＲＰＣがサーバーでまったく実行されなかったかのように、そのＲＰＣを含むトランザクションをアボートすることを可能とする。このようなセマンティクス (exact-once semantics) を持つ２種類のＲＰＣを導入する。

トランザクショナルＲＰＣ　(transactional RPC) はネスティドトランザクションとして、[6, 9] で提案されているものであり、ＲＰＣの呼び出し時には、そのＲＰＣを実行しようとする親トランザクションの環境情報をメッセージに付加し、また、呼び出しの終了時には２相コミットプロトコルを用いる。これにより、トランザクションの中から任意のサービスに対してＲＰＣを発することができる。今後はコミットプロトコルとロギングのコストの解決が重要な課題となる。

補償可能ＲＰＣ　(compensatable RPC) は補償トランザクションモデルを用いるＲＰＣである。ＲＰＣ自



クライアント／サーバーモデル



リアクティブモデル

図 3: リアクティブモデルとクライアント／サーバーモデル

身の失敗のセマンティクスはトランザクショナルＲＰＣと同様に明確であるが、各ＲＰＣは呼び出し側とは独立にコミットしてしまう。親トランザクションがＲＰＣの効果をロールバックしたいときは、逆操作に相当する補償可能ＲＰＣを用いる。補償可能ＲＰＣは、トランザクショナルＲＰＣに比べて呼び出し時に付加する情報が少なくて良く、また、コミットプロトコルが簡略化できるため実行コストが小さい。

4.4 リアクティブモデル

分散処理の利点のひとつは、システムの冗長性を利用して高いディペンダビリティを得られることである。しかし、そのためには、分散システムの状態を監視し、動的に実行環境を再構成できるメカニズムが必要である。このためには、現在主流となっているクライアントからの呼び出しで動作する受動的なサーバーのパラダイム (図3上) だけでは不十分であり、サーバー自身が状態を監視して能動的に動作するようなパラダイムが必要となる。

リアクティブモデル [11, 10] は、制御プログラムが環境をモニターし、特定の事象が生じたときにコマンドを送って反応するシステムである (図3下)。クライアント／サーバーモデルとの違いは次の２点にまとめられる。

- クライアントが明示的にサーバーを呼び出すのに対し、環境は制御プログラムを呼び出さない。そのかわりに、環境にはセンサーとアクチュエイタが備え付けられており、制御プログラムは、センサーから事象を読み出して環境の状態を変更する。

- 環境は制御プログラムを明示的に呼び出さないため、プログラム間の同期はランデブーのようなものを使わずに同期をとる必要がある。そのひとつとして、時刻によってシステムの動作を定義することが考えられる。

このようなリアクティブモデルをトランザクショナルなものとしてクライアント／サーバーモデルと併用することの利点を下記に示す。

例外処理や障害処理など、分散処理システムの実行状態の監視に基づく操作の記述を応用システムの本来の処理の記述と切り離して明確に記述でき、結果としてフォールトトレラントな機能が導入し易くなる.

トランザクションの実行制御機構のようにミドルウェアの内部で用いる場合、分散環境の負荷状態や時間制約を付加されたトランザクションの実行状態を検出できるセンサーを埋め込む。制御プログラムは、定期的にセンサーからシステム状態をモニターすることにより、時間制約を満たせないトランザクションをアボートしたり、必要に応じてサーバーマイグレーションの起動やネットワークルーティングの変更処理を行うことができる。

また、リアクティブモデルを応用システムの記述の中で用いることもできる。例えば、株の売買のシステムにおいて、株市場 (環境) に株価や売買高に関するセンサーを定義する。売買を制御するプロセスは、適宜センサーから情報を得て、条件に応じて売買操作のサブトランザクションを発する。このように、リアクティブモデルは、高レベルトランザクションの自然な定義に有用であり、グループウェア、協調処理などの応用システムにおいても適用できる。

5 まとめ

将来の頼りがいのある分散型アーキテクチャ／環境 *Hydrae* の研究の一貫として、ミドルウェアを中心に技術課題を検討してきた。今後は、具体的な応用システムについて *Hydrae* のプロトタイピングを行い、技術課題の詳細化を進める予定である。

謝辞: 本研究を進めるにあたって、研究の機会を与えて下さったソフトウェア基礎技術研究部の磯田部長と尾内リーダーに感謝します。

参考文献

[1] M. Accetta et al. Mach : A New Kernel Foundation For UNIX Development. In *Proc. of the Summer USENIX Conference*, July 1986.

[2] P.A. Bernstein et al. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987.

[3] K. Birman and T. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *Proc. of the 11th ACM Symposium on Operating System Principles*, pp. 123–138, 1987.

[4] K. Birman and K. Marzullo. ISIS and Meta Project. *Sun Technology*, Vol. 2, No. 1, pp. 90–104, 1989.

[5] U. Dayal, M. Hsu, and R. Ladin. A Generalized Transaction Model for Long-Running Activities and Active Databases. *IEEE Data Engineering Bulletin*, Vol. 14, No. 1, pp. 4–8, March 1991.

[6] J.L. Eppinger, L.B. Mummert, and A.Z. Spector. *CAMELOT AND AVALON – A Distributed Transaction Facility –.* Morgan Kaufmann Publishers, 1991.

[7] H. Garcia-Molina. Modeling Long-Running Activities as Nested Sagas. *IEEE Data Engineering Bulletin*, Vol. 14, No. 1, pp. 14–18, March 1991.

[8] Y. Leu. Composing Multidatabase Applications using Flex Transactions. *IEEE Data Engineering Bulletin*, Vol. 14, No. 1, pp. 29–33, March 1991.

[9] B. Liskov. The Argus Language and System. In *Distributed Systems – Methods and Tools for Specification – LNCS 190*, chapter 7. Springer-VerLag, 1985.

[10] K. Marzullo and M. Wood. Tools for Constructing Distributed Reactive Systems. Technical Report TR91-1193, Cornell University, 1991.

[11] K. Marzullo and M. Wood. Tools for Monitoring and Controlling Distributed Applications. Technical Report TR91-1187, Cornell University, 1991.

[12] J.E.B. Moss. *Nested Transaction: An Approach to Reliable Distributed Computing.* MIT Press, 1985.

[13] M. Rozier et al. Overview of the CHORUS Distribuetd Operating Systems. Technical Report CS/TR-90-25, Chorus Systems, 1990.

[14] 徳田英幸. 分散 OS の動向 —Mach を中心として. 1990 年代の分散処理シンポジウム – 情処シンポジウム論文集. 情報処理学会, 1990.