分散アプリケーション開発環境の検討*

1J−4

芳竹 宣裕　伊波 通晴　三上 理　河津 正人　上道 悟

日本電気(株)　C&Cシステムインタフェース技術本部

1 はじめに

近年，複数のコンピュータをネットワークに接続して，個々のコンピュータではなし得なかった効率，使いやすさを追求する分散コンピューティングが注目を浴びている．利用者は分散コンピューティングの仕組みを有効利用するのに，様々な目的のための多くの分散アプリケーションを使う．

分散アプリケーションは，処理実体が複数個存在するアプリケーションであり，処理実体間でデータを互いにやり取りしながら処理を連携し一つの目的を達成する．

ところが，数多く開発されるべき分散アプリケーションは，開発環境の不備により開発しづらく，現状では稀少であることは否めない．

本稿では，現状の開発環境の問題点を指摘し，これらを解決すべく分散アプリケーション開発環境を検討する．

2 開発環境の問題点

現状の開発環境下の分散アプリケーション開発は，以下の点において多くの工数を必要とし，ソフトウェアの品質，メンテナンスの面でも大きな障害を持つ．

- コーディング
  分散コンピューティングにおける通信アプリケーションインタフェースは，業界標準のプリミティブなもの（ソケットインタフェース，遠隔手続き呼出し等）の提供のみに留まる．これはネットワークを利用できる器だけを用意したに過ぎない（利用法，制限等によりコーディングには経験と注意深さが必要）．

  特に分散アプリケーションのコーディングは，通常のアプリケーションに比べて処理の同期，データの授受等の操作が困難である．従来の開発環境ではこれらの注意を環境側で賄えず，開発者自ら意識的に行う（デッドロックを防止するコーディング等）．

- テスト / デバッグ
  分散アプリケーションは複数の処理実体がネットワーク上に分散するので，テストおよびデバッグ作業が極端に行いにくい（現状では，ソースコードを変更して処理実体の状態を表示するデバッグルーチンを埋め込む等で対処）．

  また，処理実体間の通信のタイミングも問題になり，適当なテスト / デバッグ環境が現状では存在しない．

- 運用
  正常動作の保証のため，分散アプリケーションの各処理実体の動作を監視する実体を常駐させる必要がある．運用管理として各処理実体が持つ情報に矛盾がないよう注意しなければならない．

これらの点を解決しない限り開発の効率化はおぼつかなく，分散アプリケーションの生産性の向上は見込めない．

3 開発環境の概要

分散アプリケーション開発環境において必要とされる機能を検討する（図1参照）．

コーディング，テスト / デバッグおよび運用の問題点に注目して通信インタフェース，ビルド機能，モニタ機能，デバッグ機能を用意し，分散アプリケーションの処理実体が動作するコンピュータ毎に各機能を提供する．複数のコンピュータのこれらの機能は一括的に制御され，開発者へはネットワークに跨った複数の処理実体を統一したビューとして見せる．

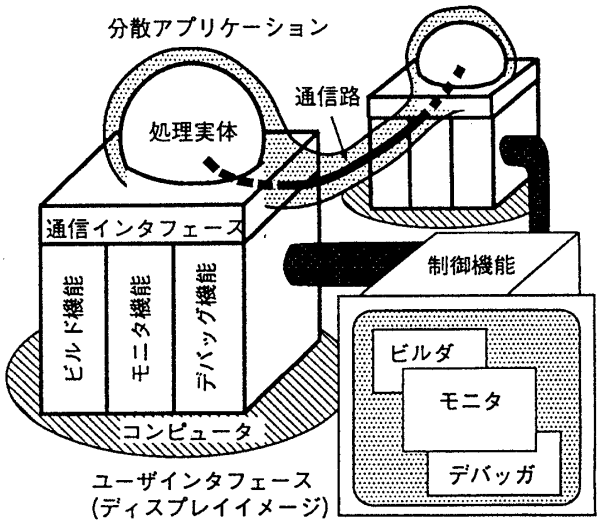


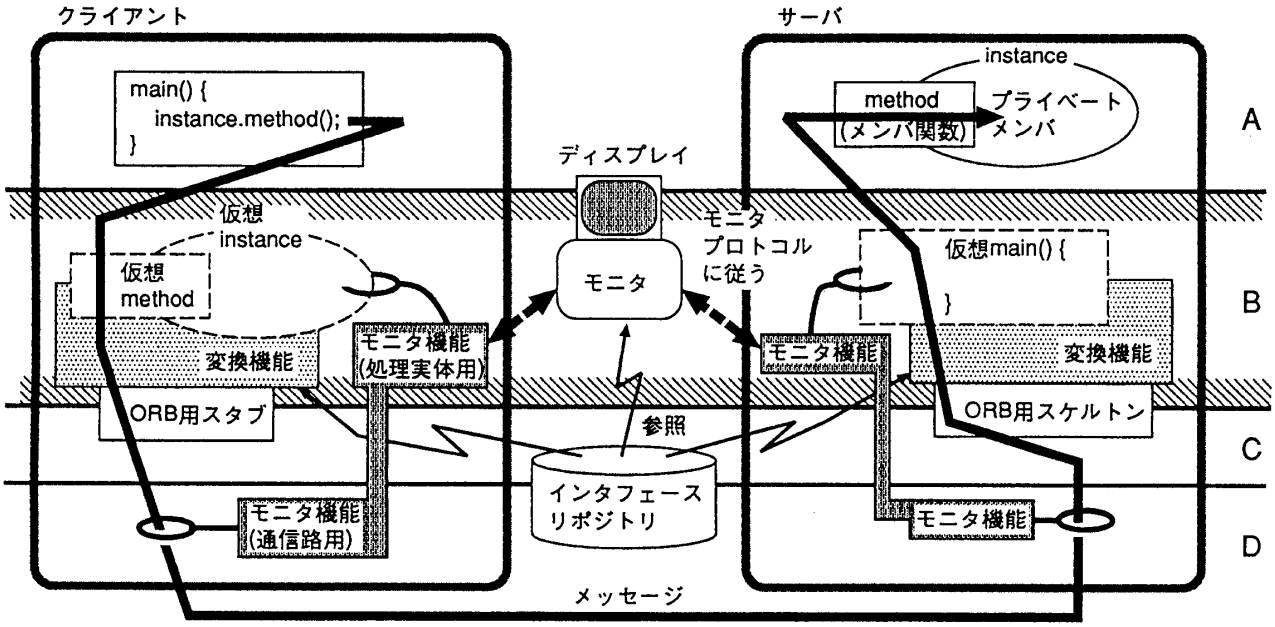
図 1: 分散アプリケーション開発環境

以下に各機能の説明を開発フェーズに従って行う．

- コーディングフェーズ
  図2は，本開発環境で開発された分散アプリケーションの動作例である．プログラミング言語はここではC++ 言語とした．

  開発者は各コンピュータで動作させたい処理ルーチンをクラス定義として記述する（図2のAの部分）．この時，通信を意識する必要はない．

*A Study on Development Environment for Distributed Applications, *Nobuhiro* YOSHITAKE, *Michiharu* IHA, *Osamu* MIKAMI, *Masato* KAWATSU and *Satoru* UEMICHI, NEC Corporation



A: 開発者がアプリケーションをプログラミング言語(ここではC++言語)で記述する部分
B: 通信インタフェース部分
C: オブジェクトリクエストブローカ(ORB)の実装部分
D: 通信プリミティブ部分(RPCあるいはTLI)

図 2: 分散アプリケーションの動作例

後はビルド機能を利用して通信インタフェース部分を自動生成するだけである（図2のBの部分）．通信処理は自動生成された仮想 instance，仮想 main 等を経由して行われる．これらは，開発者からはあたかも処理ルーチンが自コンピュータに存在するかのように見えるが，その内部には通信利用のための機構が組み込まれる．このように通信まわりの繁雑なコーディングを開発環境側で肩代りし，開発者はアプリケーション本来の処理ルーチンに専念させる．

通信の部分は将来的に業界標準になるであろうオブジェクトリクエストブローカ（ORB）[1]準拠とし，異機種，マルチベンダにも対応がとれる形態をなす（図2のCの部分）．

また，ORB のコンポーネントであるインタフェースリポジトリに追加機能としてクラス定義を一括管理させて，ネットワーク上のコンピュータ間でのクラスの再利用，一貫性保持を強化する．

- テスト / デバッグフェーズ

  モニタ機能（図2参照）は分散アプリケーションの動作に関連する通信状態を開発者に見せる．処理実体間で伝達されるデータの分析も可能であり（通信路用モニタ機能），また，通信状態を一時的に凍結して処理実体の状態を観察することもできる（処理実体用モニタ機能）．

  デバッグ機能はコードレベルの通常のデバッグ機能に加えて，通信回りのデバッグを強化する．複数処理実体の協調動作の状況を開発者に容易に把握させ，通信の同期やデータの授受の失敗によるバグの検出を速やかに行わせる．

- 運用フェーズ

  分散アプリケーションの処理実体の位置等の情報の入手および動作監視は，遠隔手続き呼出しで利用する管理実体（SunRPC の rpcbind に相当）を用いる．

  追加機能として処理実体が動作する環境生成の情報（通信プロトコルのプロファイル等）の一括管理も行い，運用管理情報の矛盾をなくし開発者を運用時の設定の煩わしさから解放する．

  また，セキュリティ対策としてインタフェースリポジトリにクラス毎のアクセス制御機構を導入する．

各機能は互いに連携し合って一つのソフトウェア開発支援環境（CASE）の形態をなす．

4 おわりに

分散アプリケーションを効率良く開発するための開発環境に関して検討を行った．現在，検討結果に従い，プロトタイプを開発中である．また，今後は異機種間の分散コンピューティングに注目し，本開発環境の適用範囲を広げる予定である．

参考文献

[1] Object Management Group, “The Common Object Request Broker: Architecture and Specification,” Draft 26, Aug. 1991.