コンポーネントウェアにおける コンポーネントインタフェースの抽象化手法

中道 真一† 永宗 宏一† 水口 孝夫† 毛利 公一†† 大久保 英嗣††
†立命館大学大学院理工学研究科 ††立命館大学理工学部

1 はじめに

現在, 我々は, 計算機資源やネットワーク帯域といったコンテキストの変動に応じてソフトウェアの構成を動的に変更可能とするために, コンポーネントウェアによるアプリケーションプラットフォームの研究を行っている. コンポーネントウェアは, 1 つのアプリケーションを複数のコンポーネントプロセスで構成するアプリケーション構成手法である. コンテキストの変動に応じて, 別のコンポーネントに交換したり, 不要になったコンポーネントの切り離しを行うことによって環境に適応することができる.

しかし, コンポーネントプロセス間の通信に, RPC[1] や RMI[2] のような従来のサーバ/クライアントモデルに基づいた通信機構を使用した場合, プロセス間のメッセージインタフェースとして, サーバおよびクライアントで共通なものを定義しなければならない. 従って, コンポーネントごとに入出力フォーマットの仕様が異なると, 同じ機能を提供するプログラムであっても通信を行うことができない. すなわち, 厳密なアプリケーションプロトコルを定義したり, サーバおよびクライアントプログラムを同時に開発・作成しなければならない. また, 動的にコンポーネントを切り離したり交換をすることは困難である.

このような問題を解決するために, 本稿では, コンポーネントプロセスが直接通信を行うのではなく, ミドルウェアが通信に介入し, それぞれのコンポーネントが持つインタフェースの違いを調整する手法を提案する. 本手法では, 通信するコンポーネント同士で入出力インタフェースを完全に一致させる必要はない. 従って, インタフェースの異なるコンポーネント間での通信やコンテキストの変動に応じたコンポーネントの動的な交換が可能になる. また, コンポーネントを開発する際, サーバ側およびクライアント側の両方のコンポーネントを開発する必要はなく, 必要なコンポーネントとそのインタフェースを用意するだけでよい. 本稿では, 特に, 本アプリケーションプラットフォームにおけるコンポーネントプロセス間通信の手法について述べる.

2 通信の抽象化手法

図 1 にシステムの全体構成を示す. コンポーネントプロセスは, メッセージ通信を行う際に, 通信先コンポーネントプロセスと直接通信を行わず, ミドルウェアであるブリッジに処理結果を伝える. ブリッジは, 分散環境においてブリッジ間で通信を行い, 相手コンポーネントにデータを伝える. ブリッジがプロセスに対して通信を隠蔽することにより, プロセスがインタフェースの違いやプロセスの位置を考慮することなく通信を行なうことが可能になる. 次に, 本システムでブリッジがコンポーネントプロセスに対して提供する通信の抽象化の概念と特徴について述べる.

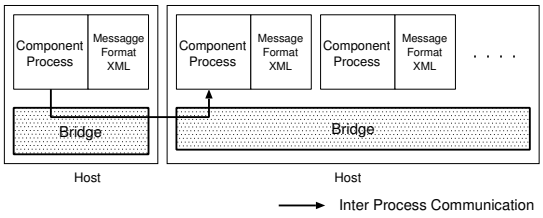

図 1 全体構成

- データアクセス透過性
  コンポーネントプロセス間で通信するためには, 計算機間でのアーキテクチャ等の違いによるデータ表現方法の相違やコンポーネントプロセスのインタフェースの違いを隠蔽しなければならない. 本システムでは, 図 2 に示すように, それぞれのコンポーネントプロセスが, 入出力の仕様の異なるプロセスと通信を行う. ブリッジは, プロセスに対して, 通信相手の仕様であるメッセージフォーマットの違いを隠蔽する. ブリッジは, クライアント側コンポーネントプロセスから受信したデータをサーバ側が受信可能な状態に変換し, サーバ側に伝える. それぞれのコンポーネントの入出力フォーマットは XML 形式で記述・保存され, コンポーネントに付属する.

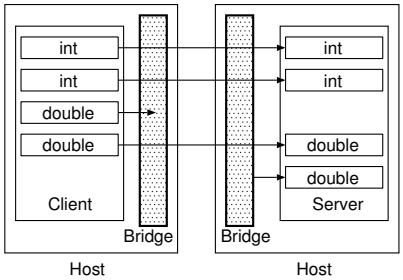

図 2 インタフェースの違い

- 位置透過性
  本システムにおいて, コンポーネントプロセスは, 通信相手となるプロセスが物理的にどこに存在するか, あるいはどのプロセスと通信するかを知る必要がなく, ブリッジが通信相手の位置を保持・管理する. ブリッジが管理する位置情報には, 通信相手の IP アドレス, ポート番号, コンポーネントポート ID, 同一ホスト内で起動しているコンポーネントプロセスのリストがある. コンポーネントポート ID は, 起動しているコンポーネントプロセスの入力

Shinichi Nakamichi†, Koichi Nagamune†, Takao Mizuguchi†, Koichi Mouri††, and Eiji Okubo††
†Graduate School of Science and Engineering, Ritsumeikan University
††Department of Computer Science, Faculty of Science and Engineering, Ritsumeikan University

ポートを一意に識別する番号である.

本システムでこれらの透過性を実現することによって, コンポーネントプロセスの交換や異なるメッセージフォーマットを持つコンポーネント同士の通信が可能になる.

3 コンポーネント間通信

コンポーネントプロセス起動時における, 通信開始のための手続きを次に示す. コンポーネントプロセスが起動するのは, アプリケーション起動時およびコンポーネント交換時である. また, 本プラットフォームでは, ユーザが使用する端末上でシェルが動作し, これがプロセスの起動や交換の指示やポリシの決定を行う.

1. コンポーネントプロセス起動時に, シェルは, 通信相手ブリッジの IP アドレスおよびポート番号を, コンポーネントプロセスが起動するホストのブリッジに通知する. また, ブリッジは, 起動したコンポーネントポート ID を保持する.
2. ブリッジは, コンポーネントプロセスに付属するメッセージフォーマット XML を, 通信相手となるブリッジと交換する.
3. クライアント側となるブリッジは, 通信相手ブリッジから受け取ったメッセージフォーマットから, 送信するデータとサーバ側から受信する戻り値データを決定する. サーバ側となるブリッジは, 受信するデータとクライアント側に送信する戻り値データを決定する.

アプリケーション起動後の通信の流れを図 3 に示す. また, 詳細を次に示す. 本体プログラムとブリッジ間および各ブリッジ間の通信には, INET ドメインのソケットを利用する.

1. クライアントプロセスは, クライアント側ブリッジにデータを送信する.
2. クライアント側ブリッジは, メッセージフォーマットに基づいて, 受信側が受け付けないデータを判別し, 送信するデータを決定する.
3. クライアント側ブリッジは, データ本体をサーバ側ブリッジに送信する.
4. サーバ側ブリッジは, 受信したデータを変数ごとに分割し, サーバ側のメッセージフォーマットに基づいて, 構造体にデータを挿入する. このとき, クライアント側から送信されず不足しているデータには、次章で述べるメッセージフォーマットに示された値を入れる.
5. サーバ側ブリッジは, サーバプロセスが要求する状態に整えたデータをサーバプロセスに送信する.

戻り値がある場合は, 同様の手順でサーバ側から通信を行う.

4 メッセージフォーマット

本システムでは, メッセージフォーマットを XML で記述し, コンポーネントオブジェクトにファイルとして付属させる. 現時点では, XML に記述可能な型は, C 言語で定義されている変数の型および配列を想定している. また, 必要なメッセージフォーマットは次の 4 種類である。

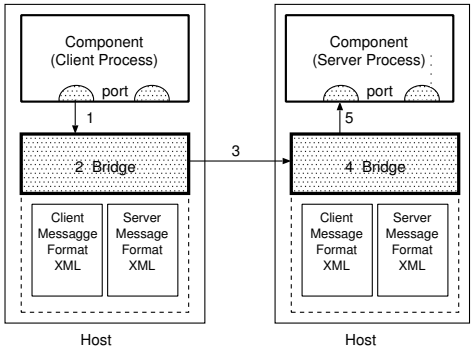

図 3 通信の流れ

- クライアント側送信メッセージフォーマット
- クライアント側戻り値メッセージフォーマット
- サーバ側受信メッセージフォーマット
- サーバ側戻り値メッセージフォーマット

サーバ側受信メッセージフォーマットの例を次に示す. それぞれの要素の内容は, その変数の意味を表し, これらが共通するプロセス同士でデータのやり取りが可能となる. depend 属性は, そのデータが必要なことを示す. また, none 属性には, そのデータが受信されなかった場合に, ブリッジがコンポーネントプロセスに送信する値を記述する. none 属性がない場合には, null 値が挿入される.

```
<data type="server_receive">
  <struct>
    <int depend="yes">opcode</int>
    <int>sound</int>
    <int>video</int>
    <array element="256">
       <char none="no title">title</char>
    </array>
  </struct>
</data>
```

5 おわりに

本稿では, コンポーネントウェアにおける, コンポーネントプロセス間通信の手法について述べた. プロセスは, ブリッジを介して通信する. 通信の際, 両プロセスが受け付けるメッセージフォーマットをブリッジが考慮し, メッセージデータの差異を隠蔽することにより, 仕様の異なるプロセスの接続が可能になる. 今後は, プロトタイプの実装を行う予定である. また, 本機構を用いたアプリケーションについても検討する.

参考文献

[1] A. Birell, B. Nelson, “ Implementing Remote Procedure Calls,” ACM Trans. Comp Syst., Vol. 2, No. 1, pp. 39–59, Feb. 1984.

[2] Troy Bryan Dowing, “ Java RMI: Remote Method Invocation,”JhonWiley & Sons, 1998.