3E-3

サービスベースシステムのユーザインタフェース

何 千山　田中英彦
東京大学工学部

1 はじめに

我々が開発しているサービスベースシステム (SBS) は、ユーザに便利、有効な分散環境を提供することを目的とする。既存の計算機機能、計算機で蓄えられたデータやプログラムを自由に組み合わせ、容易に新しい機能をユーザに提供することができるのが、サービスベースシステムの特徴である。本稿では、SBS のユーザインタフェースを概要し、その中の Remote Service Call (RSC) の概念を紹介する。また、RSC および SBS ライブラリーの実装を報告し、現在の分散システムでよく使われている Remote Procedure Call (RPC) と RSC との比較も論議する。

2 SBS および SBS のユーザインタフェース

ネットワーク上に存在するサービスに関する情報を自ノードで記述し、これらのサービスを利用したい場合、自ノードにある情報を基づいてサービスを組み合わせるのが、SBS の基本的な考え方である。すでに、サービスを管理するために、サービスに関する情報を三層ビューに分け、list 或は tree の形で記述する方法を提案し[1]、各ノードの構成は [2] の構成方法を提案した。

一方、SBS を構築する為に、使いやすいユーザインタフェースは非常に重要である。図 1 に示すように、SBS のユーザインタフェースを以下のようにまとめることができる：

1. ユーザに提供する機能

   ユーザは、SBS のインタプリタを使用して以下のような機能を利用することができる：

   - サービスの利用、組み合わせ
   - ある特定のサービスの検索
   - 新しいサービスの記述など

2. 応用プログラムに提供する機能

   応用プログラムから直接にサービスを利用することができる。これを実現するために、RSC の概念を導入する。応用プログラムは、SBS のライブラリーを利用して、分散性を意識せずにサービスを組み合わせることができる。

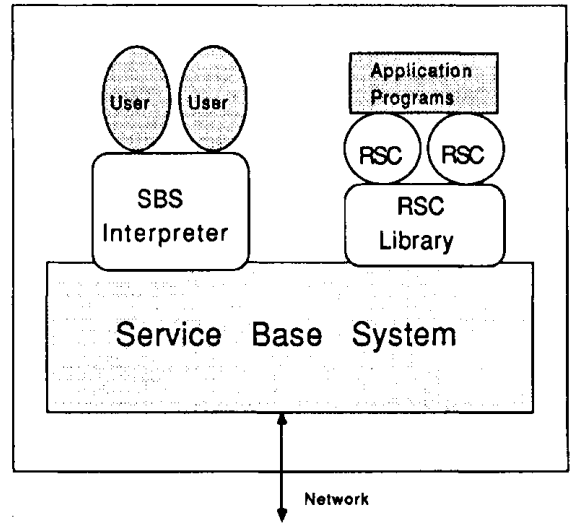

図 1: SBS のユーザインタフェース

ユーザに提供する機能については、新しいサービスを記述するために、SBS は、仕様記述言語 SDL/SAL を提供し[1]、簡単にサービスを記述することができる。本稿では、主に応用プログラムに提供する機能について述べる。

3 RSC と RPC

分散システムを実装するために、最近では RPC がよく使われている。RPC はシステムプログラム或はユーザレベルの応用プログラムに通信ファシリティを提供する。つまり、高レベル言語は、ローカル・プロシージャを呼び出すのと同じ syntax と semantics でリモート・プロシージャを呼び出すことができる。RPC は OSI のプリゼンテーション層の通信プロトコルである。RPC を使えば、応層のプログラム間通信をサポートすると同時に、応用プログラムがデータ・プリゼンテーション、トランスポート・プロトコルを意識しなくでもよいとい

An User Interface for Service Base System
Qianshan HE, and Hidehiko TANAKA
University of Tokyo.

うメリットがある。

SBSでは、応用プログラムから直接にサービスを利用したい場合、RSCというファシリティを使用すればよい。RSCは高いレベル言語の応用プログラムをサポートし、よいユーザインタフェースを提供する。RSCの基本的な考え方はRPCと同じであるが、以下の点について区別がある：

- RSCは、分散透明性を実現することができる。つまり、リモート・サービスについては、SBSはその存在場所などの情報を持っているので、RSCでそのサービスの名前だけ指定すれば、システムは自動的に実際存在するノードにサービス要求を出す。一方、RPCでは、プロシージャの存在する場所を指定しなければならない。

- 各ノードでのデータ構造の表示方法は異なる場合があるので、RPCでは、ネットワークにデータを伝送する前に、データをネットワークの標準表現形式（例えば、SUNのRPCでは、External Data Representationという）に変換する必要がある。SBSでは、サービスに関する情報は入出力データのタイプという属性を含む。リモート・ノードにデータを伝送する前に、タイプのチェックを行い、同じでない場に自動的に適当なタイプに変換するので、ユーザレベルの応用プログラムでは、データの変換という心配は必要がない。

以上の比較から見ると、RSCはサービスベースシステムを基づい応用プログラムをサポートするので、RPCより使いやすいものである。

4 RSCの実装

RSCを実現するために、SBSでは一つのRSCライブラリーを用意している。このRSCライブラリーは、サービス要求用のルーチン群からなり、応用プログラムからこれらのルーチンを呼び出すことにより、RSCを行うことができる。現在のSBSでは、C言語用のRSCライブラリーは実装されている。このRSCライブラリーは、既存のトランスポート層以下のプロトコル（TCP/IP、バーチャル・サーキットのsocketなど）に基づいて実装したものである。例えば、以下のようなルーチンを用意している：

ReqID = send_request(ReqestMessage);
Result = receive_result(ReqID);
receive_data(ReqID, FileName);
stop_reqest(ReqID);
end_request(ReqID);

RSCを行うために、まず*send_request()* を呼び出してサービス要求を行う。このルーチンを実行してから、一つのrequest IDを返す。応用プログラムから同時に複数の*send_request()* を呼び出すことができ、それぞれをrequest IDによって区別する。*recieve_result()* は、サービスの実行結果を待つ。途中で*stop_request()* を呼び出して一つのサービス要求を取り消すこともできる。サービスの実行結果を返してから、*end_request()* を使ってサービス要求を終了させる。

以下では、一つの簡単な例を取り上げ、RSCの使用方法を示す。これは π を計算する例で、リモート・ノードに *arctan()* を計算するプログラムがあると仮定する。自分のノードのSBSでは、サービス*arctan()* の存在するノード、入出力データタイプの情報を持っている。

$\pi = 16\arctan(\frac{1}{5}) - 4\arctan(\frac{1}{239})$

$\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - .....$

以下では、π を計算するローカルノード上の応用プログラムを示す：

```
reqid1 = send_request("arctan 16 5");
a = receive_result(reqid1);
end_request(reqid1);
reqid2 = send_request("arctan 4 239");
b = receive_result(reqid2);
end_request(reqid2);
pi = a-b;
```

5 おわりに

本稿では、remote service callという概念を用いたサービスベースシステムのユーザインタフェースについて述べた。応用プログラムはRSCを使用して容易にSBSを利用することができる。現在のRSCファシリティはC言語用だけであり、他の言語をサポートするために、それと対応するRSCラリブラリーを作る必要がある。

参考文献

[1] 何　千山他、「サービスベースシステムにおける分散資源 及び資源の仕様記述」、情報処理学会第35回全国大会、3U-7、1987.3.

[2] 荻野　正他、「サービスベースシステムのノード構成」、情処理学会会第35回全国大会、3U-6、1987.3.