4. クライアント・サーバプログラムの試験

Testing for Client/Server Programs by Kaoru NUMATA (Nomura Research Institute, Ltd., Advanced Information Technology Division, Advanced Systems Technology Depertment).

沼田　薫[1]

1 (株) 野村総合研究所情報技術本部システム技術部

1. はじめに

最近は企業のオフィスで利用するビジネスアプリケーションでも，クライアント・サーバプログラムの利用が普通となってきた．日本のビジネスアプリケーションではパッケージソフトの採用が少なく，ほとんどは各企業の独自仕様で作られている．その中で，新規に作成されるアプリケーションは，かなりの割合がクライアント・サーバプログラムである．これは，アプリケーションを開発している情報システム部門やソフトウェアハウスの開発現場にとって，クライアント・サーバプログラムの試験が非常に身近なものだということを意味している．

本稿では，最初にクライアント・サーバプログラムを並行処理の観点から整理し，クライアント・サーバプログラムの試験の方法を紹介する．その後，ビジネスシステム開発現場での制約をふまえた実際の試験の方法を紹介する．まず，ビジネスシステムに求められる信頼性について説明し，ビジネスシステム開発現場で現実に行われているアプローチを紹介する．さらに，現実に行われているアプローチの網羅性および有効性について考察する．

なお，ここでは独立した複数本のプログラムを組み合わせて1つの目的を達成する場合に，このプログラム群をシステムと呼ぶことにする．また，ビジネスシステムは主に企業のオフィスで使うことを目的とした事務処理システムや情報提供システムとする．

2. クライアント・サーバプログラムの試験とは

2.1 クライアント・サーバプログラム試験の特徴

クライアント・サーバプログラムは，プログラム上では並行処理を意識していないことが多い．クライアントプログラムからデータベースサーバに対してSQL文を送るような基本的なクライアント・サーバシステムでは，並行処理は意識されない．では，プログラムを組み合わせてクライアント・サーバプログラムを構成した場合に，並行処理は行われないのだろうか．たしかに，クライアントプログラムとサーバプログラムが1対1に対応している場合，プログラムがそれぞれ独立して動作しているものの，同時に処理は行われない(**図-1** (a))．しかし，クライアント・サーバプログラムの特徴は，多数のクライアントプロセス（ここでは実行中のプログラムをプロセスと呼ぶ）に対してそれより少ない数のサーバプロセスが対応することである．最も基本的なクライアント・サーバプログラムの形態は，複数のクライアントプロセスに対して1つのサーバプロセスが用意される構成である．この場合は，クライアントプロセスからみるとサーバプロセスは1つであるが，サーバプロセスからみると複数のクライアントプロセスとやりとりが行われる．そのためサーバプロセスを接点とした並行処理が行われる（**図-1** (b))．

この並行処理は，クライアントプログラムとサーバプログラムのやりとりの方法によって，形態が異なる(**図-2**)．前述のSQL文の例にあるような，クライアントプログラムとサーバプログラムのやりとりが同期している場合は，並行処理はオペレーティングシステムやハードウェアの機能によって実現されている．そのため，プログラムとしては並行処理を意識していない．一方，クライアントプログラムとサーバプログラムのやりとりが非同期の場合は，プログラム上で並行処理を意識している．そのため，非同期でやりとりを行う場合は，クライアントプログラムとサーバプログラムの数が1対1でも並行処理が実行される（**図-1** (c))．非同期のやりとりを実現するために，OLTP (OnLine Transaction Processing) やMQ (Message Queueing) をミドルウェアとして採用することが多い．どちらも，最近採用事例が増えているミドルウェアである．クライアントプログラムはサーバからのメッセージを受け付けるためのコールバックインタフェースなどを備えた構造となる．

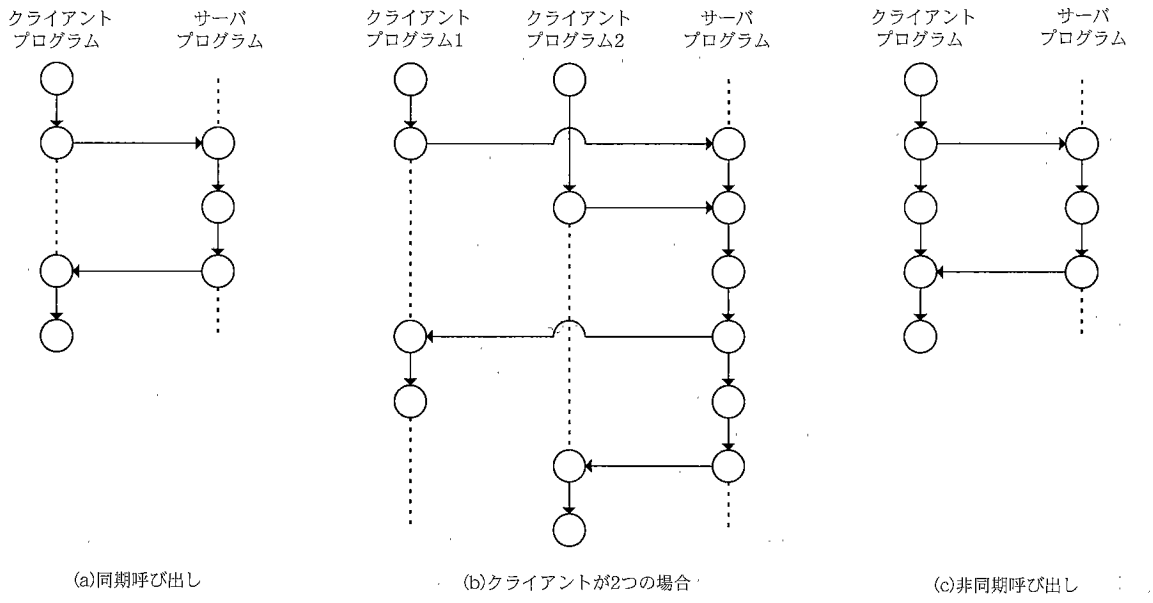


図-1 クライアントプログラムとサーバプログラムの関係

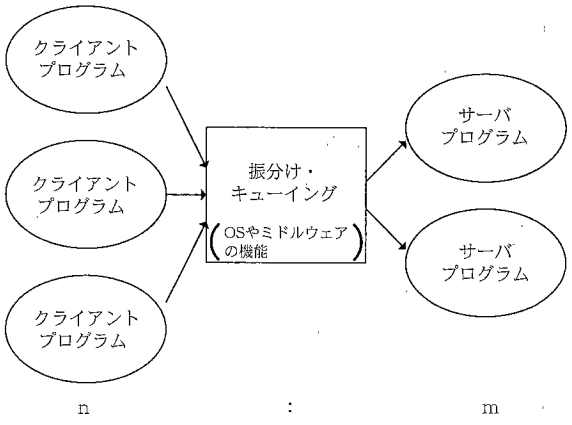


図-2 サーバプログラムの呼び出し方法

2.2 クライアント・サーバプログラム試験の方法

クライアント・サーバプログラムの試験を説明するにあたり，並行処理の部分のみを取り出すことは，全体が見渡せなくなるのであまり意味がない．したがって，ここではクライアント・サーバプログラム全体の試験について説明を行う．ただし，**2.1**節で述べたように，クライアント・サーバプログラムは基本的な構成でも実行時に並行処理となるため，結果的には並行処理の試験についても説明することになる．最初は，ビジネスシステムでの試験の分類を説明し，その次にクライアント・サーバプログラムの試験で特徴的な内容を説明する．

ビジネスシステムの試験では，試験を目的別に分類して実施するのが一般的である．これは，クライアント・サーバプログラムに限らず，ビジネスシステムでは一般的に行われている方法である．この試験の分類にはいくつかの種類があるが，「単体試験，結合試験，総合試験，運用試験」あるいは「ユニット試験，コンポーネント試験，統合試験，システム試験」のように分類することが多い．また，「試験」は「テスト」と呼ばれることもある．これらの試験の目的は次のようなものである．

(1) 単体試験

プログラム単体に不正が存在しないことを確認する試験．「ユニット試験」と呼ばれることもある．この試験は，プログラムが呼び出すサブルーチンあるいはプログラムの呼び出し元となるプログラムには不正が存在しないと仮定するか，スタブなどで置き換えて試験することが多い．プログラムの仕様で想定したすべての処理に対して試験を実施するため，プログラム上のすべての命令が最低1回は実行される．この試験は通常，プログラムの作成者によって行われる．

(2) 結合試験

プログラム同士を関連付けた場合に，不正が存在しないことを確認する試験．「コンポーネント試験」，「統合試験」と呼ばれることもある．この試験では，プログラムが呼び出すサブルーチンや利用するデータなども試験対象となり，(1) の単体試験は完了していることが前提である．この試験により，プログラム同士のインタフェース，データの種類による処理内容，関連プログラムの不正による処理内容などに対する検証ができる．

クライアント・サーバプログラムでは，クライアントプログラムとサーバプログラムのやりとりが非同期であれば，この試験で並行処理の検証が必要である．

(3) 総合試験

プログラムを組み合わせたシステムという形態に不正が存在しないことを確認する試験．「システム試験」と呼ばれることもある．この試験では，単体試験，結合試験では検証できなかった処理を確認する．実際のシステム構成に近い環境を用いて行うため，ハードウェアやネットワークとの整合性の試験，多くの機器やプロセスなどを稼働させた試験，性能面の試験，システム全体としてのセキュリティや信頼性の試験などを実施する．また，利用者にとっての操作性の検証も実施する．

クライアント・サーバプログラムにおいて，クライアントプログラムとサーバプログラムのやりとりが同期していれば，この試験で並行処理の検証が必要である．

(4) 運用試験

実際のシステムの運用を意識した試験．日次処理，週次処理，月次処理を含めたシステムの運用を繰り返し行った後のデータの整合性などを検証する．総合試験は処理ごとの整合性の確認に重点が置かれていたが，運用試験では時間軸からの整合性の確認に重点が置かれる．

単体試験および結合試験においては，プログラムの全体にわたる試験を行うために，プログラムの構造に着目したホワイトボックス試験を行う．ただし，結合試験については，プログラム上の命令をすべてというよりは，プログラムの関連についての全体を試験すると考えたほうがよい．一方，統合試験および運用試験は仕様書に着目し，仕様書上の機能を満たすことを確認するブラックボックス試験を実施する．

多くのクライアント・サーバプログラムは同期したやりとりを行っているため，並行処理の試験を行うのは（3）の総合試験となる．結合試験ではプログラム同士の整合性の検証を行うのが主な目的のため，クライアントプロセスとサーバプロセスは1対1の関係で試験を行うことが多いが，総合試験ではクライアントプロセスを複数実行して試験を行うからである．

ここでクライアント・サーバプログラムの試験において特徴的な点を説明したい．クライアント・サーバプログラムの試験で特徴的な点は，GUI（Graphical User Interface）と複数のクライアントプロセスを同時に実行するシステムの構造の2つといえる．

ビジネスシステムは企業のオフィスで使われるシステムであるため，ユーザとのインタフェースが必須である．クライアント・サーバプログラムでは，このユーザインタフェースがほとんどの場合GUIである．Microsoft社のVisual BasicのようなGUIの開発ツールは，画面上のボタンやテキストボックスなどのオブジェクトに対応してプログラム（スクリプトやマクロともいう）が記述されている．そのため単体試験で1つ1つのプログラムを試験しても，プログラム間の実行の順番についての検証は行われていない．しかし，プログラムの構造上，プログラム同士でグローバル変数，メモリ，ファイルなどを共有していることが多いので，プログラム間の実行の順番によってシステムの動作が異なる可能性がある．プログラムの実行は，ユーザが画面上の操作で任意に行うことができるので，プログラムの実行の順番についてはあらゆる組合せを想定しなくてはならない．この組合せの数は非常に多いため，すべてを試験することは大変である．図-3に示したような，画面上に3つのオブジェクト（テキストボックスが1つとボタンが2つ）しかない簡単な画面でも，それぞれに4つのイベント（たとえば，Microsoft Windowsのclicked, doubleclicked, getfocus, losefocusなど）があれば，実行順序を考慮した論理的な組合せは，384通り（${}_3P_3\times4\times4\times4$）になる．現実にはイベントに対応するプログラムがない場合もありすべての組合せに対して試験が必要なわけではないが，画面上のオブジェクトの数が10を超えることは普通である．

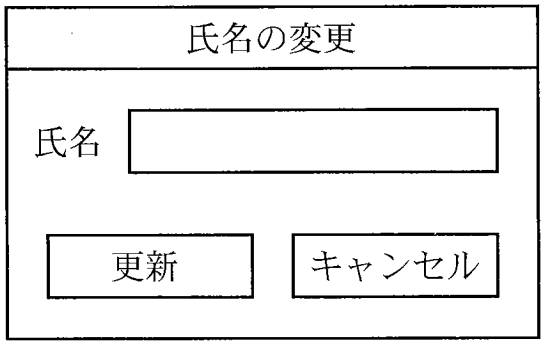


図-3 簡単なGUI画面

一方，複数のクライアントプロセスが同時に実行されるシステムの構造では，各クライアントプロセスがサーバプロセスとやりとりするタイミングの組合せは非常に多くなる．クライアントプロセスがサーバとやりとりするタイミングはユーザが画面上の操作で任意に行うことができるので，サーバとのやりとりについてはあらゆるタイミングを想定しなくてはならない．図-1（b）のようにクライアントプロセスが2つで，それぞれが1回しかサーバとやりとりをしない場合でも，6通り（$2\times{}_3C_1$）の組合せを考慮する必要がある．現

実には，クライアントプロセスの数，クライアントプロセスからのやりとりの回数はこれ以上に多く，さらにやりとりの内容も1種類ではないため，考慮が必要な試験項目はさらに多くなる．

3. ビジネスシステム開発現場における試験

3.1 ビジネスシステムに求められる信頼性

ビジネスシステムに求められる信頼性は，そのシステムを適用する分野により異なる．

医療用機器，航空機の制御などのようなシステムはプログラムの不正が人命を脅かす可能性があり，銀行や証券会社のオンラインシステムなどは対象範囲の大きさと要求される処理時間の制約からシステム停止時の代替手段を確保することが難しい．これらのようなシステムでは，システムに対する信頼性の要求は非常に厳しい．

しかし，通常のビジネスシステムにおいては，プログラムの不正によってシステムが停止しても、人間による手作業を含めた何らかの方法で代替できることが多い。たとえば，保険の契約では，顧客からの申込書の受け取りと保険料の入金があれば契約は成立しており，保険証券を即時に発行する必要はない．つまり，システムが停止していても基本的なビジネスは成立しているのである（もちろん長時間の停止は，事務処理においてさまざまな問題を引き起こすのは当然である）．このようなビジネスシステムにおいては，システム停止による業務処理全体への影響が少なければ，プログラムの不正がある程度許容される．不正をとりのぞくために必要な費用や時間がシステムの停止による影響（損害）より大きければ，経済性の観点からシステムの停止を許容して，システムとしての信頼度を低くすることが可能である．

また，ビジネスシステムでは，プログラムの不正は起こり得るという前提でシステムの設計を行うのが普通である．現実的には，どんなに試験を行っても，プログラムの不正をなくすことができない．そのため，あるプログラムに不正があった場合でも，システムとしては処理を続行できるように設計を行う．これには，プログラムの不正によりプロセスが停止した場合に，データの整合性を確保した上で停止したプロセスを再実行する仕組みや，システム全体が停止するような不正があった場合には，待機中のハードウェアに切り替えて処理を続行するフェイルセーフ構造などがある．ビジネスシステムでは，業務処理に影響を与えないようなシステムとしての信頼性が求められるが，プログラム単体の完全性は必ずしも必要ではない．

3.2 ビジネスシステム開発現場における試験のアプローチ

ビジネスシステム開発現場における試験は，時間と費用の制約の中で行われる．システムの利用を開始する日時が外部の要因によってあらかじめ決まっているために試験の期間が十分に確保できない場合や，システム開発のための総費用があらかじめ設定されているために試験に費やせる費用が限定される場合などである．そのため，ビジネスシステム開発現場における試験は，論理的に実施可能な試験の全項目を実施できないことがほとんどである．したがって，限られた範囲で効果のある試験を実施することが大切となってくる．ここでは試験の分類ごとに，ビジネスシステム開発現場での試験のアプローチを簡単に説明する．

(1) 単体試験

単体試験においてはプログラムの構造に着目したホワイトボックス試験を行い，プログラム全体の試験を行う．これは，時間や費用の制約があったとしても，プログラム単体としての信頼性が確保できていなければ，プログラムを組み合わせたシステムは成り立たないためである．

(2) 結合試験

結合試験においてもプログラムの構造に着目したホワイトボックス試験を行うことが多い．ただし，単体試験と比べた場合，試験の対象となる範囲が広いため，論理的に可能なすべての試験項目を実施するとは限らない．その中でも試験項目の数が多くなるファイルやインタフェース上のデータやGUI入力については，限られた組合せだけを試験することが多い．

たとえば，2つのプログラム間においてプログラムAからプログラムBにデータが引き継がれるようなインタフェースがあったとしても，プログラムAが理論上出力できないデータの組合せについては，プログラムBの入力とはなり得ないため試験を行わない場合などがあげられる．このようなプログラムBへの入力は，プログラムAに不正がないかぎり起こり得ない内容であり，単体試験の終了を前提とすれば必要ないからである．また，GUIの試験項目数が多いことは**3.1**節で述べたが，理論上は入力可能であっても処理としては意味をもたない入力についての試験は行わないことが多い．

これとは別に部品となるプログラムが再利用される場合は，使い込まれた部品についての信頼性は十分高いと考え，試験の項目を省略する場合がある．

(3) 総合試験

総合試験では，結合試験よりさらに試験の対象範囲が広くなるため，理論的に可能な試験の項目をすべて

表-1　試験のシナリオの分類

分　類	内　容
正常系	・ユーザによる通常の操作 ・外部からの通常のデータ引継ぎ
異常系－ユーザ	・ユーザが操作可能な通常外のシーケンス ・ユーザが入力可能な通常外のデータの値 ・外部から引継ぎ可能な通常外のデータの値
異常系－システム	・関連プログラムの不正による通常外のシーケンス ・関連プログラムの不正による通常外のデータの値
障害系	・ハードウェアの障害による通常外のシーケンス ・ハードウェアの障害による通常外のデータの値

試験することはほとんどない．この試験は仕様書上の機能を満たすことを確認するブラックボックス試験である．試験の実施は，実際のシステムの使い方を想定したシナリオを用いることが多く，そのシナリオについては**表-1**に示すような分類で作成することが多い．

「正常系」の試験では通常の業務処理で行う内容をシナリオとしてまとめて，試験を実施する．正常試験のシナリオでは，通常の業務処理で定常的に行われる内容を試験するため，必要となるすべての項目に対して試験を実施する必要がある．

「異常系－ユーザ」の試験では，「正常系」のシナリオの中でユーザが通常外の操作を行った場合や，不正な値をもつデータを入力した場合を想定している．これらの項目は，ユーザのミスにより通常の業務処理中に操作や入力が可能なために，「正常系」の試験と同様にほとんどの項目に対して試験を実施する必要がある．

「異常系－システム」の試験は，プログラムの不正がないかぎり起こり得ない内容である．そのため，発生した異常処理がシステムに与える影響を考慮すれば，必ずしも実施が必要ではない項目もある．

「障害系」の試験では，ハードウェアの故障などを想定しているため，日常的に起こり得る内容ではない．したがって，障害がシステムに与える影響を考慮すれば必ずしも実施が必要ではない項目もある．ただし，ネットワーク障害やディスク障害など，装置の特性で発生しやすい障害については試験を行う必要がある．

並行処理の観点から試験が必要となるのは，「正常系」と「異常系－システム」である．「正常系」では，複数のクライアントプロセスからの同時アクセスや，データベースのロック中へのアクセスなど，正常な処理だが結合試験では実施できなかったタイミングについて試験を行う．タイミングについては多数の試験項目があるが，必要な項目に対して確実な試験を行わなくてはならない．「異常系－システム」でも，並行処理で問題となりそうなケースを確実に試験することが大切である．

表-2　在庫管理システムにおける重み付けの例

重　み	試験項目
大	・発注による在庫引当て機能 ・倉庫への出庫指示機能
中	・月次集計表
小	・システム管理用の性能レポートの出力機能

(4) 運用試験

運用試験では，ユーザ向けの操作マニュアルや管理者向けの操作マニュアルを基に試験を行う．したがってこの試験では，システム的には実施可能でも試験として実施されない項目は多数存在する．運用試験では，システムを継続的に使っていく中で，操作内容やデータ内容に不正や考慮漏れがないかを確認することが目的である．したがって，プログラムに対する機能的な試験というとらえかたはできない．

4. ビジネスシステム開発現場における試験の網羅性と有効性

単体試験ではホワイトボックス試験を行い論理的に考えられるすべての項目を試験することはすでに述べた．ただし，この試験は並行処理の試験ではない．ここでは，クライアント・サーバプログラムとしての結合試験や総合試験の網羅性を考察してみる．**3**章で述べたようにビジネスシステムの開発現場では，時間と費用の制約の中で最大の効果が出せる試験を行うことを指向している．結果として，論理的に考えられる試験に対する現実の試験の網羅性は完全ではない．一般的にも並行処理のすべての試験を実施することは現実的ではないといわれている[2]．

ビジネスシステム開発現場での試験の方法は，試験項目に対して重要度に関する重み付けを行う方法と考えられる．総合試験のシナリオでは，重要な処理を優先して試験項目が作成される．これには重みの少ないケースにおける不正は，業務処理全体に大きな影響を与えないという暗黙の前提がある．在庫管理システムにおける試験項目の重みの例を**表-2**に示す．たとえば，「注文による在庫の引当て機能」での不具合は，誤った商品が納入されるといったような，ビジネスそのものに影響を与える．それに対して，「システム管理用の性能管理レポートの出力機能」での不具合は，システム管理面だけに影響があり，ビジネスそのものに影響を与えることはない．

試験項目に対して重みを与えたとすると，ビジネスシステム開発現場での試験の方法の実効値としての網

羅性は，

Σ（実施する試験項目×試験項目の重み）／Σ（論理的な試験項目×試験項目の重み）

と考えることができる．この値は実施する試験項目を論理的な試験項目で単純に割り算した値より大きくなると予想できる．ただし，この重みの客観性を評価することは非常に難しい．経験的には，不正が起きるとほかに代替手段のない項目にくらべて不正が起きてもほかの手段で代替可能な処理の重みは10倍以上の差がある．現実にはほかのさまざまな要因が組み合わさって最終的な重みは決定される．ビジネスシステム開発現場では，経験を基にして技術者が試験項目の絞り込みを行っているのが実状である．

また，ビジネスシステムの特徴として，不正の発生を前提としたフェイルセーフ構造をとることを**3.1**節で述べた．この場合，フェイルセーフを実現するためのプログラムに対する試験の重みは非常に高くなるが，それ以外のプログラムの試験の重みは相対的に低くなることには注意が必要である．

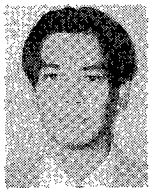
5. おわりに

本稿では，並行処理の範囲を超えて，ビジネスシステム開発現場での試験の考え方を述べた．しかし，クライアント・サーバプログラムの適用分野は，ビジネスシステムだけではない．ほかの分野での試験については別の考え方もあると思われるが，当社ではビジネスシステムの開発を主に行っているため，ここで述べることはできなかった．また，本稿では試験の考え方のみを述べている．具体的な試験の実施方法については，参考文献の1)，3)，4) をはじめたくさんの文献があるので，それらを参考にしていただきたい．

参考文献

1) Myers, G. J.: The Art of Software Testing, John Wiley & Sons, 1979（ソフトウェアテストの技法，近代科学社）．
2) 伊藤栄典，川口　豊，古川善吾，牛島和夫: 順序列テスト基準に基づく並行処理プログラムの充分性評価，情報処理学会論文誌，Vol.36, No.9, pp.2195-2205（Sep. 1995）．
3) Beizer, B.: Software Testig Techniques, Van Nostrand Reinhold, 1990（ソフトウェアテスト技法，日経BP出版センター）．
4) Beizer, B.: Black-Box Testing-Techniques for Functional Testing of Software and Systems, John Wiley& Sons, 1995（実践的プログラムテスト入門，日経BP社）．
5) 藤沼彰久，角田　勝，斎藤直樹，西本　進，沼田　薫: C/Sシステム構築入門，日経BP出版センター（1995）．

（平成9年9月30日受付）



沼田　薫

1964年生．1987年大阪大学基礎工学部情報工学科卒業．同年野村コンピュータシステム（株）（現（株）野村総合研究所）入社．現在，（株）野村総合研究所情報技術本部システム技術部リーダー．主に金融業界，流通業界向けのクライアント・サーバシステムの設計に従事．著書「C/Sシステム構築入門」（共著，日経BP出版センター），訳書「Java & CORBA C/Sプログラミング」（共訳，日経BP社）．