メインフレーム JavaVM と利用形態

Mainframe JavaVM and its Utilization Form

西 本 誠 一

要 約 Java の実行環境は様々なプラットフォームで実現されている．パーソナル・コンピュータやワークステーションのみならず，メインフレームの世界も Java 実行環境は提供されている．Java の利点は Write Once，Run Anywhere とうたわれているとおり，一度 Java プログラムを作成すれば，プラットフォームに依存することなく，どの Java 実行環境下でも実行できるということである．また，さまざまな“Off The Shelf”の Java アプリケーションを利用できる点も Java の魅力のひとつである．本稿では，CS シリーズにおける JavaVM（Java 仮想マシン）の現状，利用形態について論じる．

Abstract Java run-time environment is implemented on various platforms and provided for not only personal computer and a workstation but also the world of a mainframe. The advantage of Java exists in that a program written in the Java programming language can run on various platforms without re-compling as well-known as“Write Once, Run Anywhere”. Java applications of“Off The Shelf”are available to Java is also one of charming points of Java. This paper discusses the current status and utilization form of JavaVM(Java virtual machine) in CS Series.

1. はじめに

CS シリーズにはメインフレーム・パーティションと Windows パーティションが混在している．メインフレーム・パーティションは MCP または OS 2200 を搭載している．MCP および OS 2200 双方の環境で共に JDK1.3 に準拠した Java 実行環境を提供している．従って，MCP と OS 2200 環境の Java 実行環境を区別して論ずる必要はないが，JDBC の実現方法やパフォーマンスの改善方法などの実装には相違がある．MCP 環境では Java クラスファイルを MCP のネイティブコードにコンパイルすることでパフォーマンスの改善を図っている．一方 OS 2200 環境では JIT の実装によりパフォーマンスの改善を図っているのがその例である．しかしながら，本稿の主たる目的はそれぞれの実装の詳細を紹介することではなく，メインフレーム（CS シリーズ：以降，メインフレームは CS シリーズを指す）での Java 環境の今日を概括し，さらには次世代におけるメインフレームと Java の係り方を議論することである．したがって，MCP と OS 2200 を区別して述べず，MCP 環境を搭載した CS シリーズを例に論じることにする．

2. Java の特徴

Java の特徴は，Write Once，Run Anywhere とうたわれているように，一度書かれた Java アプリケーションは，たとえ実行するプラットフォームが変わったとしても再コンパイルすることなく実行できることである．これは，Java ソースをコンパイルした結果をすべてのプラットフォームで共通なバイトコード（Java クラス）と

呼ばれる中間コードで持ち，Java 実行環境がクラスファイルに書かれたバイトコードをネイティブなマシン・コードに翻訳することで実現している．プラットフォーム毎のハードウェアやオペレーティング・システムの違いは Java 実行環境（Java 仮想マシン）が吸収している（図 1）．プログラムを実行するプラットフォームに依存せず，同じプログラムを再コンパイルすることなく実行できるということは，それまでのコンピュータの常識を覆すほどのできごとである．以前からもオープンな開発環境として C 言語がもてはやされていた．C 言語が出現したときも，プラットフォームが変わってもプログラムを書き換えることは不要と言われた時期があった．しかし，その場合でも，include ファイルなどでプラットフォーム依存の部分を補完しているため，必ず実行するプラットフォーム上でプログラムを再コンパイルする必要があった．Java は，このような移行作業からユーザを解放させた．また，Java の特徴として，強力なセキュリティやネットワーク対応，分散処理機能と行った機能があげられる．インターネットとの親和性も他の言語にはないものをもっている．もちろん，国際化（I 18 N）に対応しているのも特徴のひとつである．

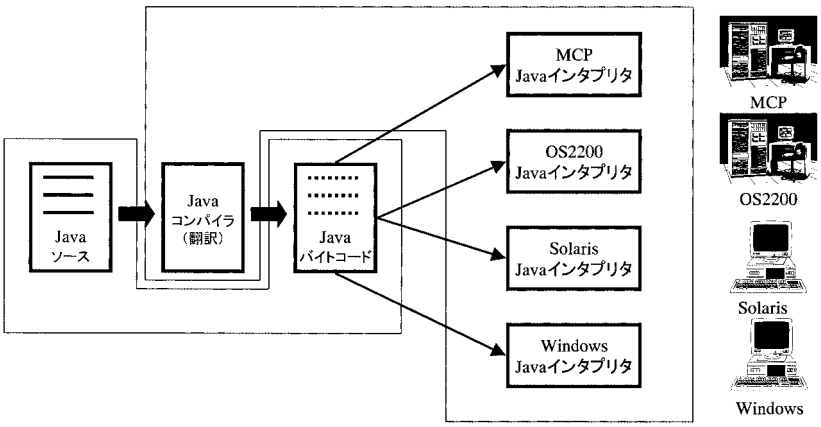

図 1　Write Once, Run Anywhere

3. JavaVM の意義

メインフレーム上で JavaVM を実現させるための意義を考えてみる．メインフレーム上での一般的なアプリケーション開発言語としてあげられるものは，事務処理計算用の COBOL や科学技術計算用の Fortran がある．また，C 言語も利用できるが，メインフレーム上で圧倒的なシェアを持つ言語は，やはり COBOL である．近年のインターネットの普及により，メインフレームでもインターネット上での商取引である B-to-B（Business to Business：企業間電子商取引）や B-to-C（Business to Consumer：消費者向け電子商取引）の必要性が出てきた．この分野で一歩も二歩も先行している言語が Java である．Java はインターネットと密に融合しているプログラム言語である．ここでメインフレーム上で XML を利用する必要が出てきた場合を想定してみる．XML パーサや XML ジェネレータを COBOL で新たに自己開発するアプローチもある．しかしすでに多くの XML 関連のツールやアプリケーションが溢れて

いる．そういったものをメインフレーム上のアプリケーションから利用できれば既存資産を利用した素早い業務への展開に有利となる．この場合，既存の COBOL で書かれたアプリケーションからそれらのツールを直接利用できるわけではないが，メインフレーム上で Java が実行できれば，そういったツールを手軽に利用できることになる．ここで取り上げた XML パーサや XML ジェネレータの利用のみならず，既存アプリケーションのモダナイズ，RMI（Remote Method Invocation）を利用した他システムとの連携などにもメインフレーム上の Java 実行環境は重要である．

4. CS シリーズ JavaVM

ところで，一般的にメインフレーム Java 実行環境の課題として Windows/UNIX 環境と比較したときのコスト・パフォーマンスが懸念される．メインフレームでは独自プロセッサ技術を使用して，特にミッションクリティカルな業務への適用を前提とした耐障害性，可用性，信頼性に多大な投資をしているためである．また，CPU 能力，入出力，メモリ量やキャッシュ・メモリ量などの資源がトランザクション処理などの COBOL アプリケーションに最適となるようにバランスよく設計されている．従って， 相対的に高い CPU 能力が必要な Java 環境にはコスト的に不利な場合がある．

またデータベース構造の課題として，MCP 環境の DMSII データベースや OS 2200 環境の RDMS データベースのアクセスがある．これらは SQL データベースではない．そのため，Java アプリケーションから直接 DMSII データベースや RDMS データベースをアクセスするためにはタイプ 3 の JDBC ドライバを使う複雑さが出てくる．

このため CS シリーズの Java 環境ではミッションクリティカル業務への適応性を持ちながら，広く Java アプリケーションが利用できるコストパフォーマンスのための仕組みや，より高い生産性が得られるように Java 環境を使い分ける仕組みを持っている．以降ではこれらの仕組みについて述べる．

4.1 JCODE コンパイラ

JCODE コンパイラは，CS シリーズ（MCP）での Java 高速化技術である．これは，Java クラス（バイトコード）を読み込み，MCP 環境で実行可能な C 言語で記述されたネイティブコードを生成するコンパイラである．CS シリーズ（MCP）では独自の Java の開発環境をあえて提供していない．Java の開発はすべてパーソナル・コンピュータやワークステーション上で行うことを前提にしている．これは世の中に広く出回っている優れた統合開発環境ツールの利用を考えてのことである．パーソナル・コンピュータやワークステーションでプログラミングした Java プログラムは，javac によりコンパイルすると Java クラスが出来上がる．この Java クラスはバイトコードと呼ばれる中間コードを含んだファイルである．このバイトコードを実行時に Java 実行環境がプラットフォーム毎にネイティブなマシンコードに翻訳し，Java アプリケーションが実行されるわけである．つまり，Java はクラスを生成するときはコンパイラ言語であるが，実行時にはインタプリタ言語なのである．JCODE コンパイラは，インタプリタによるこの翻訳作業を実行前に先取りする．したがって，Java アプリケーションを実行する前に Java クラスを MCP 環境にコピーし，JCODE コンパイラを使って Java クラスファイルをネイティブコードにコンパイルしたものを Java

実行環境にあらかじめ結合しておくことができる．これらの作業は JNIBUILD ユーティリティと呼ばれる MCP 環境の Java 実行環境で，JNI（Java Native Interface）を利用するためのユーティリティを用いて行うことができる．コンパイルしたネイティブコードは，Java アプリケーションからは JNI を通じて Java ネイティブメソッドとして実行される（図 2）．JCODE コンパイラを使えば，インタプリタに比べ，効率が大幅に改善される．

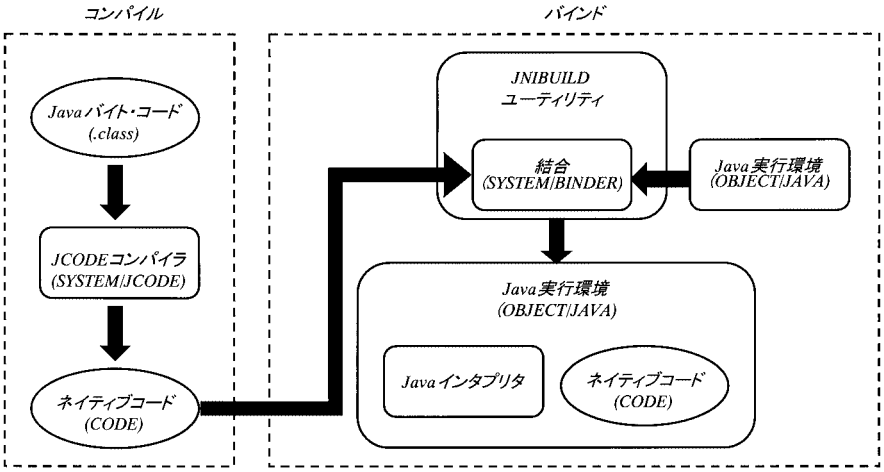

図 2　JCODE コンパイラ

4.2 Offload JVM

Offload　JVM は CS シリーズ（MCP）でのもうひとつの Java 高速化技術である．CS シリーズ（MCP）には独自のメインフレーム CPU で構成するシステムと，独自のメインフレーム CPU を持たずに Windows 上でメインフレーム CPU をエミュレーションするシステム（図 3）がある．この Offload JVM は主に後者のエミュレーションシステム用に考案した技術である．Java 実行環境は Intel プロセッサ上で動作する Windows 環境にも存在する．Offload　JVM は，MCP 上の Java 実行環境を図 4 に示すように Intel 上の Windows 環境にオフロードする技術である．これは COM（Component Object Model）を利用することにより実現可能である．COM は，実行プログラムや DLL 間の連携を，オブジェクト指向技術を使ってインプリメントするものである．MCP 環境上の Java 実行環境から Windows 環境の Java 実行環境を LRPC（Lightweight Remote Procedure Call）でシームレスに利用することにより，あたかも MCP 上に Java 実行環境が存在するかのようにみせかけるのである．COM では同じコンピュータ上のプロセス間通信に LRPC という通信機構を利用する．LRPC は，RPC（Remote Procedure Call）の小型版で，プロセス間通信を効率化するために設計されている．MCP 環境の資源を利用するために開発された Java パッケージ（com.unisys.mcp）も，MCP Service のスタブを JNI（Java Native Interface）を使って MCP ネイティブ・メソッドとして呼び出すことで MCP 環境の MCP Service Library を利用することができる．つまりは，CS シリーズを利用すれば従来のホスト資産を Java

の最新技術を使って効率的に利用できるようになる．なお，この技術は将来実装する計画である．

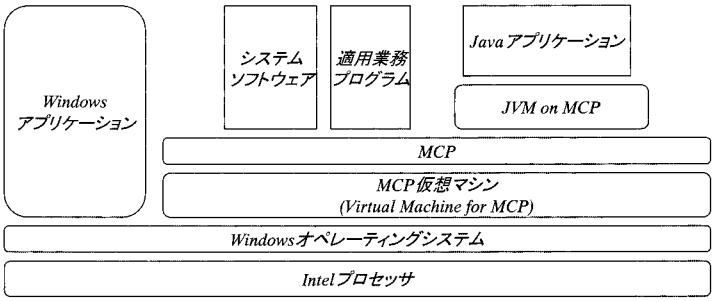

図 3　CS シリーズの実行環境

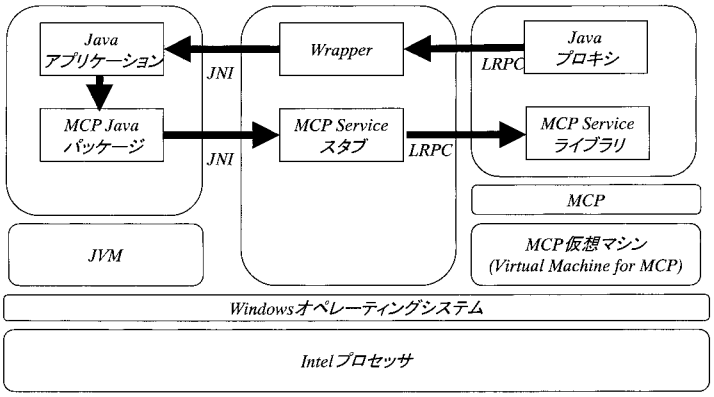

図 4　Offload JVM の概念

4.3　JDMI API

CS シリーズ（MCP）上の Java 実行環境において必要不可欠なものは，やはり DMSII データベースのアクセスである．DMSII データベースは，MCP 上のネットワーク型のデータ管理システムである．Java では，データベースにアクセスする手段として JDBC（Java Database Connectivity）を提供している．JDBC は，リレーショナル・データベースで広く使われている SQL データベースに対するアクセスを提供するインタフェースである．Java アプリケーションから SQL データベースにアクセスする方法は JDBC を利用することが一般的である．DMSII データベースは SQL データベースではないが，DMSII データベースに対して SQL データベースのふりをさせることによって JDBC を利用することができる．DMSII データベースを SQL データベースとして取り扱うことができるプロダクトに Application DataAccess ドライバや INFOAccess ODBC ドライバがある．Application DataAccess は，MCP アプリケーションから Windows 上の SQL データベースにアクセスを可能にするプロダクトである．INFOAccess ODBC は，Windows アプリケーションから ODBC

による DMSII データベースアクセスを可能にするプロダクトである．現在，DMSII データベースにアクセスするための JDBC ドライバは，すべて Application DataAccess ドライバや INFOAccess ODBC ドライバを利用している（図 5）．そのため，データベースの更新などの必要がある場合には DMSII データベースを SQL データベースとして扱うために Microsoft SQL Server を利用する．SQL 文の構文検査を Microsoft SQL Server 側で一貫して処理する思想からこのような形態をとっている．この場合，ネットワークを経由することになり，パフォーマンスが少し悪くなってしまう．そこで登場した技術が JDMI API（Java DMINTERPRETER API）である．これは JDBC の代わりに Java アプリケーションから DMSII データベースに直接アクセスする API を提供する．ただし，MCP に特化した DMSII データベースにアクセスする API をクラスとして提供するため，Windows 環境の Java 実行環境では利用することはできない．JDMI API を利用した Java アプリケーションは MCP 環境以外のプラットフォームでは実行できることができなくなることに注意する必要がある．既存の DMSII データベースを使用したアプリケーションを Java に書き換える場合や，パフォーマンスを重視する場合には JDMI API が向いている．パフォーマンスを問わない場合，他のデータベースに移行する可能性がある場合，DMSII データベースの使い方に詳しくない場合などには JDBC を利用するべきである．

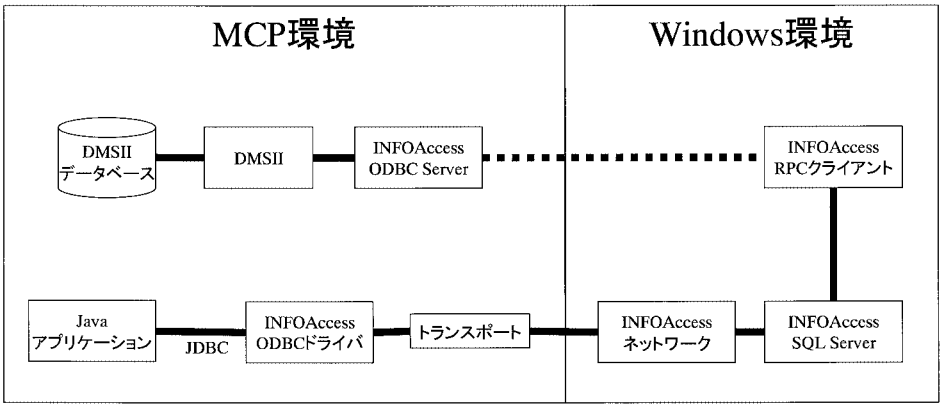

図 5　データベース・アクセス経路

5. CS シリーズ JavaVM の利用形態

CS シリーズの JavaVM の利用形態として，J2EE コネクタ，Servlet/JavaServer Page，既存プログラムとの融合について，以下に述べる．

5.1 J2EE コネクタ

J2EE コネクタは，J 2 EE コネクタ・アーキテクチャ（JCA :J2EE Connector Architecture）に基づくコネクタの総称である．JCA は，J2EE プラットフォームを既存システム（EIS : Enterprise Information System）に接続するための標準を定めたものである．EIS の例としては，従来からのメインフレームのトランザクション処理システムや ERP（Enterprise Resource Planning）などがある．

標準化の意図は，機種や処理システムが異なる毎にそれぞれのコネクタを開発する

ことの弊を避けることにある．接続元がm種類，接続先がn種類であると，m＊nの組み合わせが存在するが，JCAの採用によりm＋nの組み合わせに単純化することができる（図6）．このようなアダプタをリソース・アダプタと称し，EISベンダ（CSシリーズでは当社）がこの標準リソース・アダプタを提供する．この標準リソース・アダプタをJCAをサポートするアプリケーション・サーバにプラグインすることにより，色々な種類からなるEIS，アプリケーション・サーバおよびJ2EEアプリケーション間のスムーズな接続・連携を図ることができる．

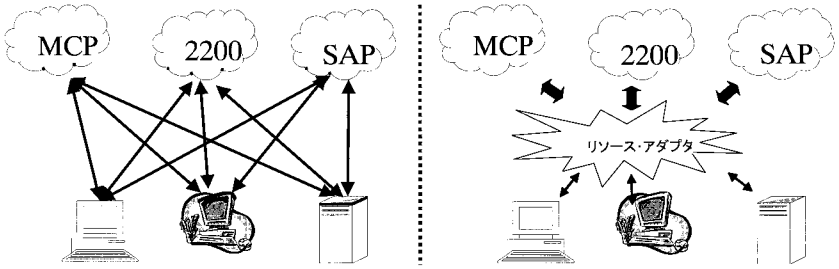

図 6　JCAによる単純化

5.2　Servlet/JavaServer Page

メインフレーム上のアプリケーションをWeb化する手段としてServlet/JavaServer Pageを利用できる．これにより，既存のビジネス・ロジックからプレゼンテーション部分を切り離すことができる．Servlet/JavaServer Pageは，Web Transaction Serverが標準でサポートしている．Web Transaction Serverは，MCP環境で稼働するWebサーバである．Servletは，Web Transaction Serverの起動時にロードしておくこともできるし，オンデマンドで必要に応じてロードすることもできる．オンデマンドでロードされた場合でも，一度ロードされるとServletエンジンを停止するまで再びロードする必要はない．これはMCP上でJava実行環境を一度だけ構築すれば他のServletも同じJava実行環境を共有することができることを意味する．一方，Javaアプリケーションは，Javaアプリケーションごとに Java実行環境を構築する必要がある．このためメインフレーム上のJavaとしてはJavaアプリケーションに比べてJava　Servletや JavaServer　Pageの方が効率がよいことになる．JavaServer Pageはオンデマンドで要求されるためWeb Transaction Serverの起動時にロードすることはできないが，一度コンパイルされるとクラスが変更されるまで再コンパイルを必要としない仕組みが取り入れられている．JavaServer Pageは，ロジックの変更が発生してもJava　Servletのようにプログラマがコンパイルしてクラスを生成する必要はない．Web　Transaction　Serverが自動的に更新が発生したJavaServer　Pageをコンパイルし，クラスをロードする．こういった面でもJavaServer　Pageは Java　Servletに比べて保守性，柔軟性に優れており，HTMLコンテンツの中に JavaServer　Pageを記述できるため，可読性にも優れている．アプリケーションのWeb化にとって JavaServer　Pageはプレゼンテーション部に最適な技術であると言える．

5.3 既存プログラムとの融合

メインフレームには長年にわたる基幹システムとしての実績と既存資産がある．その実績に基づく堅牢性を備えた Java プラットフォームを提供することができる．また，既存資産を Java から利用できるのもメインフレーム JavaVM の強みのひとつである．独立系ソフトウェアベンダが提供する各種ソフトウェアや開発ツールを利用して，新世界（Java）と従来社会（既存プログラム）の融合に JavaVM を利用することができる．既存プログラムのモダナイズにプレゼンテーション部として Java アプレットを用い，メインフレーム上の JavaVM には既存プログラムへのラッパーメソッドを用意し，アプレットからは RMI（Remote Method Invocation）を使ってラッパーメソッドを呼び出し，JNI やソケットを通じて既存プログラムにアクセスすることもできるだろうし，Java アプレットではなく Java Servlet や JavaServer Page をプレゼンテーション部に利用することもできる（図 7）．RMI で呼び出すメソッドを他システムの Java 実行環境に置き，既存プログラムを他システムと連携させることもできる（図 8）．XML パーサ（XML 4 J）を利用し，XML 文書を介しての他システムとのデータ交換のオープン化（図 9）などにメインフレーム JavaVM を利用することができる．

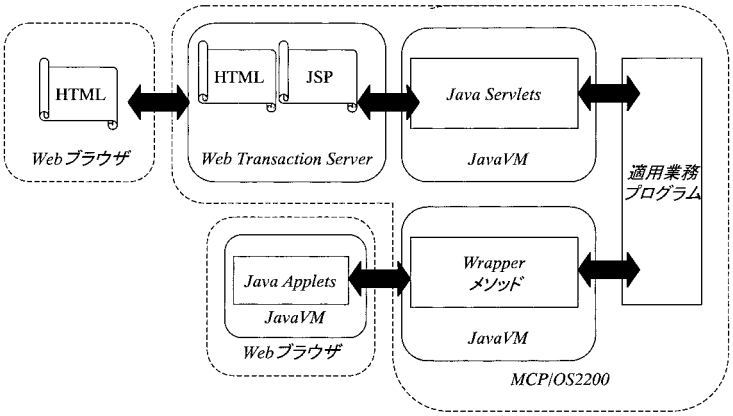

図 7　既存プログラムのモダナイズ

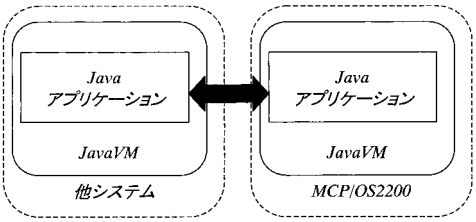

図 8　他システムとの連携

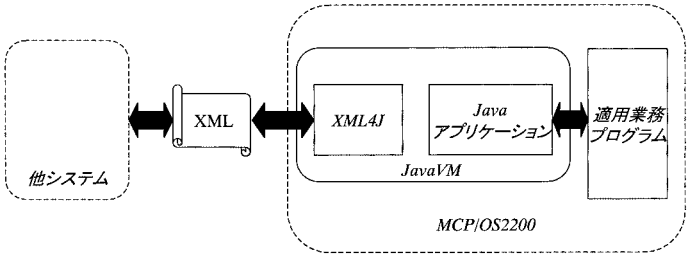

図 9　他システムとのデータ交換

6. お わ り に

これまでメインフレーム JavaVM の現状と利用形態について述べてきた．当社のメインフレームも，CS シリーズで採用されたように独自プロセッサからエミュレーション・マシンに変化をとげてきている．これもまた，ひとつの仮想マシン（Virtual Machine for MCP)である．昨年，マイクロプロセッサ関連の発表が行われる MICRO-PROCESSOR　FORUM において，エミュレーション技術が独自プロセッサの性能を上回る事実も発表されている[1]．CS シリーズのようなエミュレーション技術を利用した機種は，独自プロセッサを搭載した機種に比べてコストパフォーマンス面で有利である．また，CS シリーズで導入されたエミュレーション技術の台頭を考えると，Offload　JVM を使って Java 実行環境を Java に適した環境での実行を可能にする CS シリーズは，Java とメインフレームの融合の架け橋になる機種であると言える．Offload　JVM 技術を実装した CS シリーズは，最新の Java 技術を利用できるメインフレームとして，将来に渡って広く利用できるだろう．

参考文献　［1］「MICROPROCESSOR FORUM 2001 レポート」
http : //www.watch.impress.co.jp/pc/docs/article/20011019/mpf 04.htm
［2］ J.Cadman「COM/DCOM 実践プログラミング」(株)アスキー
［3］「アスキーデジタル用語辞典」(株)アスキー
［4］「JVM on MCP 解説書」Unisys Cooperation
［5］「Java サーブレット API 解説書」Unisys Cooperation

執筆者紹介　西　本　誠　一（Seiichi Nishimoto）
1982 年神奈川大学工学部工業経営学科卒業．1986 年日本ユニシス(株)入社．HMP　NX/A シリーズの基本ソフトウェアの受入れ・保守業務に従事．現在，プロダクトサービス部サービス技術推進室に所属し，オープンソースを基盤としたサポートサービスビジネスに従事．