ミッションクリティカル業務をオープンシステムで実現するテクニカルマネジメントポイントについて

Techical Management Matters of Mission-critical Application System Implemented with Open System

西　谷　　浩, 勝　俣　英　善

要　約　オープンシステムが注目されて久しいが，価格・信頼性・拡張性の点からミッションクリティカル業務分野への適用はなかなか進まなかった．

しかし，ハードウェア/ソフトウェア技術の進歩によりハードウェアの高性能化・低価格化が進み，信頼性もずいぶん向上し，ミッションクリティカル業務分野への適用も徐々に増えてきている．

本稿では，銀行におけるミッションクリティカル業務システム構築事例をもとに，オープン環境の技術基盤およびマネジメントポイントについて述べる．

Abstract　It is a long time since the open architecture applied to computer systems, however, it took a great deal of time to apply to the mission-critical business application field,because of the cost, reliability and expandability of the open architecture.

However, recently the open architecture has been gradually applied to the mission-critical business application fields. Because of the progress in hardware and/or software technology, and the performance of hardware products has been highly enhanced, and the price of hardware products has been decreased.

This paper describes the technological infrastructures and management considerations on an open architecture environment, introducing a sample mission-critical business application in a centain large bank.

1. はじめに

オープンシステム技術を採用したシステムの導入が始まった初期の段階では，UNIX をオペレーティングシステム（OS）とするサーバ/ワークステーションとリレーショナル・データベース管理システム（RDBMS）を組み合わせることにより，汎用検索・シミュレーション等の特定分野での適用が多かった．

この初期の段階では，UNIX のサーバ/ワークステーションは高価であったこと，各種ツール/ミドルウェアも日本語化等の問題から充実しているとは言い難い状況であったため，大規模で多数の利用者が使用する業務システムよりも，特定部門の高負荷で高速性が要求される業務を補完するためのツール的なシステムが主体であった．

その後，

1）Windows の登場による PC の急速な普及
2）ネットワーク技術の進歩
3）ハードウェアの低価格化，高性能化
4）IEEE[*1] に代表される規格化の進展
5）デファクト・スタンダード（業界標準）の進展
6）ミドルウェア/ツール群の整備

に代表される状況の変化が追い風となり，オフィス・オートメーション（OA）を中心としたパーソナル・コンピュータ（PC）の普及と利用者の拡大，それを相互に接続する WAN（Wide Area Network)/LAN（Local Area Network）といったオープンシステムに不可欠な環境の整備が進み，UNIX サーバ＋Windows クライアントといった組み合わせの業務システムが徐々に浸透し始めたが，ミッションクリティカルな業務分野への適用は見送られてきた．

本稿は，都市銀行においてミッションクリティカル業務をオープン環境で構築した際の技術基盤の構築事例である．この技術基盤はアプリケーションの実行環境や運用環境を支えるためのものであり，プラットフォームやソフトウェアの選択，ソフトウェアの組合せなど，ミッションクリティカル業務のシステム要件を満たす必要がある．そのためには，技術的な観点からのプロジェクトマネジメントが重要となる．

第 2 章で構築したシステムの概観を紹介し，第 3 章で実行基盤構築について，第 4 章ではアプリケーションを支える基盤構築について，第 5 章では運用基盤構築について，第 6 章では開発環境について，各々技術的なマネジメントポイントを述べる．

2. システム概要

構築したシステムは銀行の証券円資金システムであり，開発規模は 5000 人月，開発期間は 2 年の大規模ミッションクリティカル・システムである（図 1)．

3. 実行基盤の構築

オープン環境におけるシステム構築は，まず実行基盤を構成するハードウェア（HW)/ソフトウェア（SW）のコンポーネントの決定から始まる．メインフレームの世界では，HW を提供するベンダーが OS だけでなく実行基盤・運用基盤も提供している．しかし，オープン環境の世界では，OS ベンダーが提供する SW だけでなく，各ベンダーからも実行基盤・運用基盤について種々の SW が提供されている．ミッションクリティカル・システムをオープン環境で構築するに当たっては，どういう観点で SW の選択を行い，どのような SW コンポーネントとするか，が重要である．

3.1 HW コンポーネント

まず最初は HW プラットフォームと OS の選択である．

今回のシステムでは，デスクトップは Windows を搭載した PC 以外に選択の余地はなかった．特に堅牢性の観点から WindowsNT とした．

サーバはいくつかの選択肢がありうるが，ユーザで使用実績のあった Solaris を搭載した Sun 社の大型サーバとした．運用の堅牢性・安全性・セキュリティの観点から，これを事務センターに設置しアプリケーションとデータを集中管理する方式とした．

3.2 SW コンポーネント

アプリケーションを動かす実行基盤はデータベース（DB）とデータコミュニケーション（DC）が両輪である．DB はリレーショナル型が標準である．この分野での代表的なベンダーは数社であり，機能的にも大差はない．DC については標準的なものは存在しない．これはクライアント/サーバシステムといっても，機能をどのよう

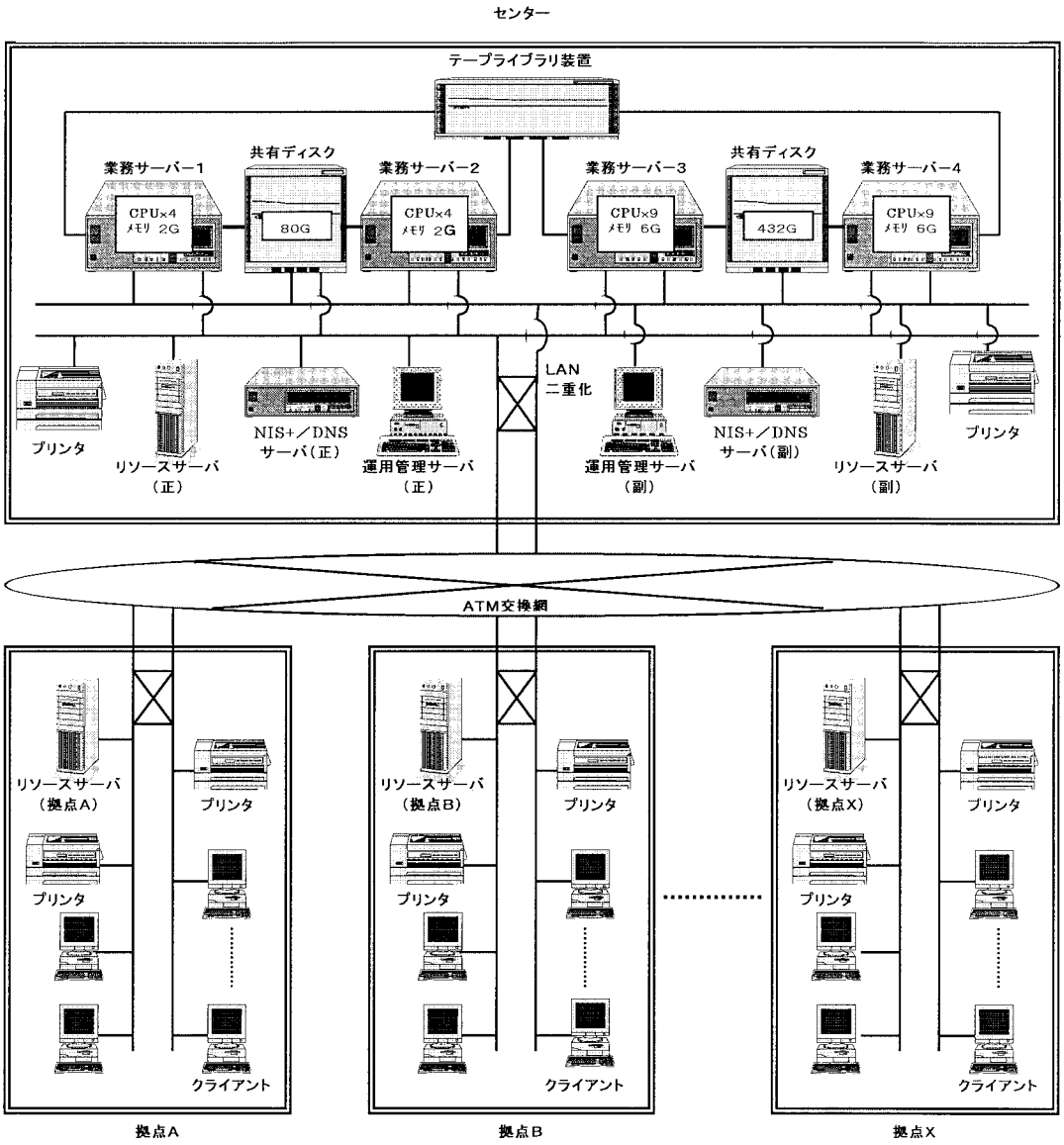

図 1　全体構成概要図

に配置するかにより，使用する DC が異なってくるからである．したがって，DC については機能配置を含め検討する必要がある．

主な検討の項目・結果・理由は，以下の通りである．

1）　アプリケーション構造……三層型（第三世代言語）

二層型はクライアントでプレゼンテーション層とビジネス層すべての機能を記述し，DB に直接アクセスする方法であるが，この場合，DC は一般的にデータベース・ベンダーが提供する通信 SW を利用する．この方式は簡易なクライアントビルダー（VB，PowerBuilder など）でアプリケーションを記述でき，開発は比較的容易である．しかし，複雑な機能を持つ大規模システムでは，性能面・

運用面で重大な問題が発生する危険性が高い．また，PC の世界は技術進歩が早く，全面的にこのプラットフォーム上でソフトウェアを開発するのは投資効果からも疑問である．

三層型はプレゼンテーション層とビジネス層を分離し，それぞれ別のノードに配置する．開発は二層型に比べやや難しくなるが性能チューニングや堅牢な運用に有利であり，企業にとって重要なビジネスロジックをより安定的なプラットフォームで実現できるので，開発したソフトウェア資産の長期保全にも望ましい．三層型の実現方式には LINC[*2] や Forte[*3] のようなクライアントサーバビルダーを使用する場合と，C や COBOL のような第三世代言語を使用する選択肢がある．前者は通常そのビルダーが DC 機能を包含するが，後者は言語自体にそのような機能がないため，いわゆる On-Line Transaction Processing（OLTP）用ソフトウェアを使用するのが一般的である．三層型でのクライアントサーバビルダーは生産性の高さが売り物だが，数百万ステップに達する大規模システムでの実績が少なく，開発および保守要員の確保に課題がある．

サーバをビジネスサイトではなくセンターに集中配置すること，要求されているデータ量と処理能力，および障害対応，開発規模を考慮すると，二層型およびクライアントサーバビルダーを使用した三層型では開発は困難である．

2） データコミュニケーション……System ν［nju:］

On-Line Transaction Processing（OLTP）用の通信 SW にはデファクトスタンダードが確立していないため，個別事情で選択していく必要がある．今回はベンダーのサポート力を重視した．ミッションクリティカルな業務では障害発生に対し最速の対応が求められるが，特に外国製の製品は簡単なバグの対応にも数ヶ月を要するケースが珍しくなく，弊社が責任を持って対応できる自社製品である System ν［nju:］とした．

3） データベース管理システム（RDBMS）……ORACLE

DB としてはデファクトスタンダードであること，弊社で多数の実績があること，から ORACLE とした．

4） サーバ開発言語……C

メインフレームの世界では COBOL が主流であり，今回のような事務系システムの開発に適するが，オープン環境では圧倒的に C 言語が普及している．開発要員の確保，各種プロダクトとの親和性からサーバの開発言語は C 言語とした．

5） クライアント開発言語……VisualBasic（VB）

クライアントについては，VisualBasic（VB）とした．VB はバージョンアップの都度互換性が問題となり生産性が良いとも言えない．しかし，マイクロソフト社の製品以外は汎用性，将来性に不安があり，またピーク時には数百人規模のプログラマが必要であり， Java 等最新のインターネット技術はまだ実績がなく，選択肢が限られていた．

4. アプリケーション基盤の構築

コンポーネントを決定したとしても，それだけではシステムを構築できない．イン

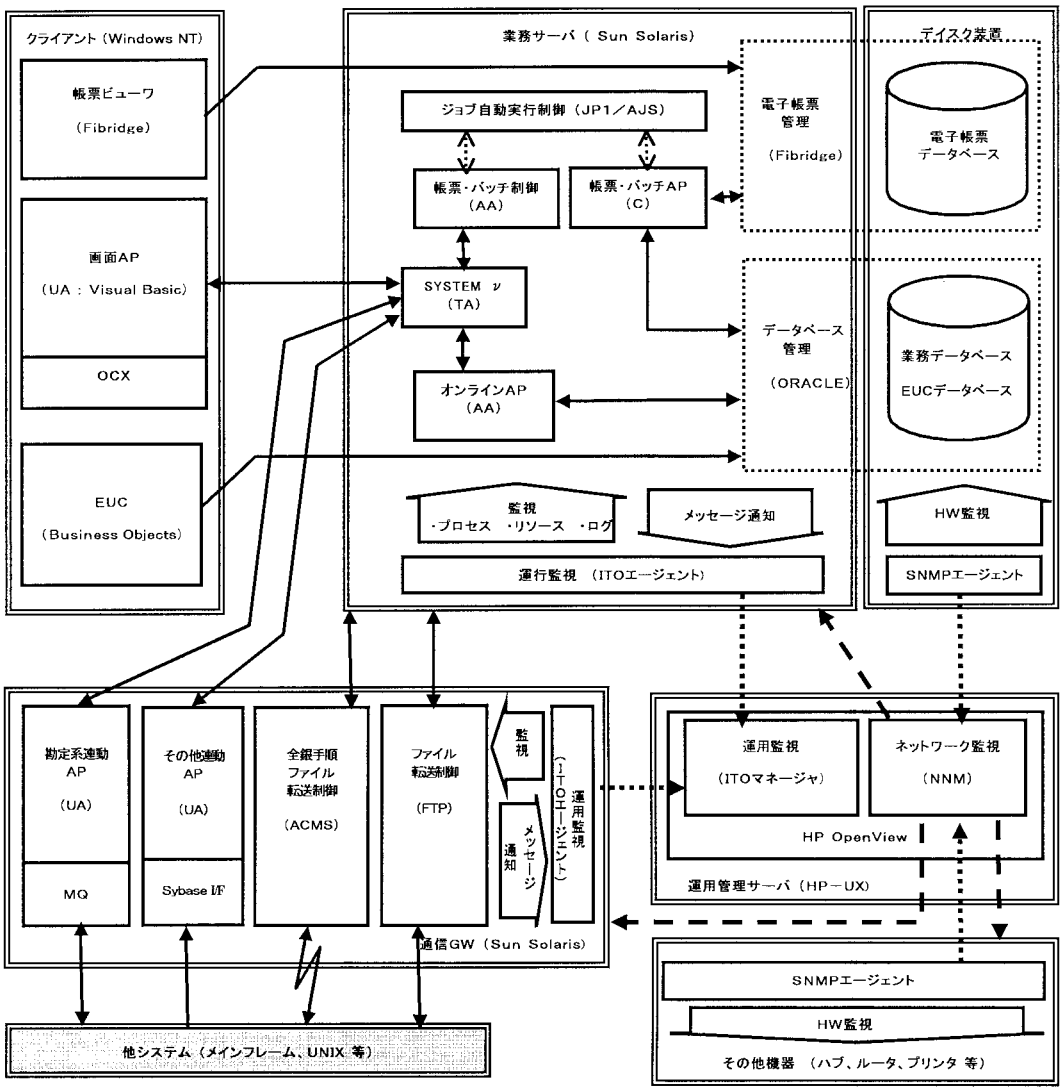

図 2　アプリケーション基盤全体概要図

タフェースの統一性がないし，機能的にも不十分である．また，大規模システム開発においてプログラム開発の生産性を高め，性能と品質を向上させるためには標準化を避けることはできない．特にシステム設計の観点から言えば，求められる処理方式を定型化し，それを実現するコア機能を共通化することが必須となる．これはメインフレーム環境でも同様であるが，コンポーネントが異なるベンダーの SW で構成されるオープン環境では，一層重要である．同時にそのような共通化だけでなく，オープン環境ならではの最新情報処理テクノロジーや向上著しいエンドユーザ情報リテラシーを上手に活用することがオープンシステムによるシステム構築を成功させる鍵となる（図 2）．

4.1　オンライン処理

PC 画面からのデータエントリや照会，あるいは外部システムとのリアルタイムの

データ交換といったオンライン処理はSYSTEM ν［nju:］をベースにしたOLTPシステムとした．OLTPに求められるシステム要件を以下に示すが，ここではそれぞれにどのような実現手段をとったかを述べる．

1）異機種複数ノードにあるプロセス間の通信手段
2）要求された処理をネットワーク上の適切なプロセスに振分けるメカニズム
3）同じ処理を複数並列に実行することができる多重処理メカニズム
4）障害回復単位であるトランザクション管理
5）要求内容や処理結果のロギング，リカバリ機能

1)〜3）の要件についてはSYSTEM ν［nju:］が提供する機能を基本的に利用している．通信方式は同期型のRemote Procedure Call（RPC）であり，振分けはCORBA[*4]が定義するORB[*5]に拠る．簡潔に述べると，クライアントプログラムはサーバプログラムをサブルーチンコールで呼び出すが，その名前解決は実行時にネットワーク上で動的に決定される，というものである．

SYSTEM ν［nju:］の機能をそのまま利用しているとはいえ，ORBの概念は一般プログラマには難しく，SYSTEM ν［nju:］の開発言語であるC＋＋は充分な要員を確保できる見込みがないため，インタフェースを自動生成するツールを提供し簡略化している．また，異機種間で異なるコード体系やデータ配列の対応もこの機構の中で組み込んだ．

多重処理はプログラムをマルチスレッド[*6]で実行することによって実現している．SYSTEM ν［nju:］では処理を要求するクライアント側のプログラムをUser Agent（UA），要求された処理を実行するプログラムをApplication Agent（AA）と呼ぶが，AAは常駐型のマルチスレッドプログラムである．Cでリエントラント型のプログラムを作成するのは難しいことではない．しかし，システムライブラリの使用方法や動的メモリの割当て/解放などに注意が必要なため，そのような部分に共通APIを提供することで一般のプログラマでもマルチスレッドコーディングができるように工夫した．

4)〜5）のトランザクション管理とロギング機能はSYSTEM ν［nju:］に拠らず，独自の方法を考案した．一般的にデータの回復はRDBMSのロールバック機能に拠らざるをえない．そこで，まずAAにおける一回の処理（スレッド）がトランザクション単位になるようにオンラインプログラムを設計した．次にスレッド毎にRDBMSとの通信セッションを割当て，業務プログラムの処理結果によってDB更新のコミット/ロールバックを決定する共通APIを準備することで，簡易的にトランザクション管理を実現している．

ロギング機能もまた共通APIの中に組み込んで入力メッセージと処理結果を保存するようにした．このログにより監査証跡や障害追求に備えるだけでなく，入力トランザクションからのベーシックリカバリ機能も実現している．

4.2 外部システム接続

外部システムとの接続には専用のサーバを用意し，窓口を一本化することで，経路が複雑にならないよう考慮している．外部システムとのリアルタイム接続にはゲートウェイを開発し対応した．ゲートウェイでは，外部システムとの接続は米国IBM社

の MQ を採用し，システム内の通信方式である SYSTEM ν［nju:］へのプロトコルに変換を行っている．

一方，ファイルによるデータ交換は FTP をベースにしたファイル転送方式を独自に開発している．当初は市販のファイル転送ツールの利用も検討したが，相手側には同様のツールが必要など実状にそぐわないため，標準的な FTP に基本的な運行制御機能を持たせることで対応している．これにより運用に手間のかかる MT によるデータ交換は極力廃止した．

4.3 帳票システムの合理化

ネットワーク化が進展した今日のオフィス環境でも，帳票が果たす役割は依然として重要である．旧来のシステムでは大量の帳票を夜間バッチでセンターに出力し，それを社内便で翌日にエンドユーザのもとに届ける方法がとられていた．その結果，オフィスは鮮度の低い情報と紙の洪水に悩まされ，センターでの帳票仕分け，発送作業が運行部門の大きな負担になっていた．

しかし，情報処理の即時性が益々重要になってきた今日では帳票もそれが必要な時に直ちにエンドユーザの手元に出力されることが要求される．一方で運行部門の合理化，スリム化は経営の重要課題である．

これらの矛盾を解決するために，今回のシステムでは川鉄情報システム社の Fibridge を採用し，帳票をその出力要求時点で電子化する仕組みとした．帳票はサーバ上の格納庫に自動的に整理・分類されるため，エンドユーザは必要な時にいつでも格納庫から取出して紙に印刷でき，簡易操作の帳票参照ツールを利用しデスクトップ上で仕事を済ませることもできる．また帳票から任意にデータを切り出して EXCEL 等のデスクトップツールで再利用することも自在である．なによりも，大量の紙を人手で仕分けする必要がなくなったため，運行部門の負担を大幅に軽減することができた．

4.4 エンドユーザコンピューテイング

今日的なシステムにおいては，いわゆるエンドユーザコンピューテイング （EUC）によるアドホックなデータ検索・加工ニーズに対する解決手段を用意しておくことが要請される．この機能がないと非定型的なデータ処理に対するシステム開発要求が際限なく発生し，保守コストが肥大するか，システムとしての役割を果たさなくなる．

今回の開発では，RDB 検索ツールとして BusinessObjects 社の BusinessObjects を利用している．ツールの選択に際しては，DB に格納されたデータに対して RDBMS のスキーマと別にエンドユーザに分かり易いビューを定義できること，条件指定などオペレーションの操作性に優れていること，検索結果を EXCEL などの EUC フロントエンドにシームレスに連携できること，を重視した．

4.5 クライアント基盤

画面上のユーザオペレーションでは，サーバ上のマスタで持っている名称，属性などの情報を必要とすることが多い．そのような場合に都度サーバプログラムを呼び出して照会するのでは，効率が悪く快適な操作性が得られず，またプログラムの本数も増えてしまう．そこでクライアントのメモリ上にマスタ情報のキャッシュを貼付け，そこを参照する API によりクライアントプログラム内で処理が完結できるようにし

た．当然サーバ側でマスタ情報に更新が発生すれば，それをリアルタイムに近いかたちでキャッシュに反映させる．また，プログラミングの生産性を高めるため市販のコンポーネントソフトウェア（OCX 等）を積極的に利用した．

4.6 セキュリティ強化

グローバルスタンダートなセキュリティレベル達成を新システムの重要施策として，セキュリティポリシーの具体化に取り組んだ．その骨子は

・利用者認証，特にパスワードによる本人識別の厳格な運用

・利用者の業務職掌・職務権限に基づくシステムやデータの利用制限

・利用状況のモニタリング等，アクセスの監視強化

であり，それはアプリケーションの利用者であるエンドユーザ部門だけでなく，システムの保守を担当する運行部門にも等しく求められるクライテリアである．

エンドユーザ認証は Windows 認証の上に，今回の開発システムとして独立したセキュリティ空間を構築し，安全性を高めた．そのために banyan 社の SteetTalk ディレクトリサービス SW を利用し，厳格なパスワード運用基準や利用基準をクリアしている．業務職掌はユーザ毎に属性登録し，それに応じてメニューの内容を変えることで対応した．一方，運行部門のセキュリティ強化には運用管理サーバの OS である HP 社の HP-UX の高信頼性モードを採用した上で，重要なアクセス情報は暗号化するなどの工夫を行なっている．

5. 運用基盤の構築

運用基盤の範囲は広いが，中心は監視と自動化である．オープン環境では一般的に機能別，ロケーション別に複数のサーバ，プリンタなどの周辺装置が分散配置される．特に大規模ユーザになれば，稼働する本番システム自体が多数存在し，運用管理コスト削減の観点からも，システムを別々に管理するのではなく，全体システムを 1 個所で集中監視するニーズが高い（シングルポイントオペレーション）．また，システムの安定稼働の観点からは，人間の判断・検証が必要な各種設定・登録等の運用操作を除き，人的関与を極力排除し自動化することが必要である．自動化のポイントは，ジョブの自動実行制御と，High　Availability（高可用性：HA）システムとして障害時の対応を HW/SW 的に自動化することである．オープン環境でのミッションクリティカル・システムの構築にあたって重要なポイントの一つは，この HA システム構築により高信頼性を確保することにある．

5.1 集中監視とジョブ自動実行制御

集中監視とジョブ自動実行制御については，オープン環境において既に一般的に実現されており，特に問題はない．

今回の開発ではユーザで使用実績のある，ヒューレット・パッカード（HP）社の OpenView IT/オペレーション（ITO）と NetWork Node Manager（NNM）を採用した．その主たる機能はイベント監視とメッセージ通知で，この種類のソフトウェアとしては標準的なもので実績もある．監視の仕組みとしては運用管理サーバからの監視を行うものと，個々の機器内部での監視結果が運用管理サーバへ通知されるもの，とに大きく二つに分けられる．

前者は運用管理サーバ上のNNMを使用して行われる．NNMはPingコマンドの結果からハードウェアの状態を判断し，正常な応答が無い場合は異常として，運用管理サーバ上に異常のメッセージを出力するとともに，GUIで該当機器が異常として表示する．

後者は個々の機器での監視は管理対象ハードウェアの内部で行われ，障害や障害に至る可能性がある事象を検知した場合はITOを通して運用管理サーバ上に異常のメッセージが出力されるとともに，GUIで該当機器が異常として表示される．また，運用関連の情報は他の分散システムを含めたシステム横断的な統合監視システムへ転送され，集中監視がおこなわれている．

管理対象ハードウェアの内部で行われる監視は以下の通りである．

1） ログ監視……OSや各種ミドルウェアから出力されるログの内容を監視する．
2） プロセス監視……アプリケーションを除くサーバ上で必須の常駐プロセスの存在を監視する．
3） リソース監視……サーバ上のリソース（ディスク等）の使用状況を監視する．
4） HW監視……機器内部のコンポーネントの障害を監視する．

ジョブの自動運行機能については，株式会社日立製作所のJP 1を採用し，ITOとの連携機能を提供している．日次・月次といった定例的なジョブだけでなく随時的な処理やオンライン処理などの常駐プログラムも全てJP 1のジョブネットワークとして登録し，起動終了制御や実行状態の管理が統一された方法でコントロールできるようにしている．また，今回のシステムでは，ライブラリ型MT装置を利用しバックアップの自動化が実現されており，MTの装填/付け替え等の運用負荷の軽減がはかられている．

5.2 HAシステム

HAシステムとは「ハードウェアに冗長性を持たせて構成し，SWにより障害の監視/障害時の操作を自動化させることにより稼働率を高めたシステム」である．一般的にシステムは2台以上のサーバで構成され，1台のサーバで行っているサービスの継続が不可能になった場合，他のサーバが自動的にサービスを引き継ぐ事でシステムの稼働率を高めている．しかし，サービスを引き継ぐための切り替え（フェイルオーバ）のために，ダウンタイムが発生する．したがって，フェイルオーバの機会を少なくすること，フェイルオーバ時のダウンタイムを短時間にすること，がポイントとなる．各サーバ内のディスク/ネットワークインタフェース等を，物理的経路を含めてコンポーネントに冗長性をもたせて構成する事により，フェイルオーバの発生を極力回避する事が可能となる．そのため，個々のハードウェアについて以下の構成とした．

1） 業務サーバは，同じ構成を持つ2台のサーバを1組とし，ホットスタンバイ可能なHigh Availability（高可用性：HA）システムとする．
2） 業務サーバに接続される共有ディスクは，HWでRAID-5[*7]機能をもつ装置2台を一組とし，SWによるミラーリング（RAID-1）を行い，実質的には四重化する．
3） 各サーバのローカル・ディスクはSWによるミラーリングを行う．
4） 運用管理サーバ，NIS＋[*8]/DNS[*9]サーバ，リソースサーバは，同様の構成・

機能を有するHWを1台以上バックアップ機として用意し，主たるサーバの障害時に切替え可能な構成とする．

5) ネットワーク機器は，同様の構成・機能を有する2台で一つの経路をサポートし，障害時には自動的回避可能な構成とする．

フェイルオーバ時のダウンタイムを時間短縮するためにジャーナリングファイルシステムを採用している．これにより，フェイルオーバが発生した際にシステム的に行われるファイルシステムのチェック時間が144秒から12秒と大幅に短縮されている．また，フェイルオーバ時には障害が発生したサーバのサービスを引き継ぐために各種の常駐プログラムが順次起動されるが，フェイルオーバ時には起動のみを行い，正常に起動されたかのチェック/リトライをJP1/AJSに任せることにより，非同期に起動が行われるような考慮を行い時間短縮をはかっている．

なお，障害監視/障害時操作を自動化するHA管理SWはVERITUS社のFirstWatch，ジャーナリングファイルシステムSWは同じVERITAS社のVxFSをを採用している．

6. 開発基盤の構築

一般的な開発基盤も検討したが，大規模ミッションクリティカル・システムでの実績，採用した実行基盤に適したツールがなく個別に作り込みを行っている．ただし，開発基盤の中心となるリポジトリは，弊社パッケージSWであるSIATOLを採用したことから，開発基盤の中心となるリポジトリは，SIATOLリポジトリを改造して使用した．オープン環境ということで，自由度が高く，標準化および環境設定が重要ポイントとなる．ここでは製造工程での代表例を述べる．

6.1 プログラム規約

品質および開発・保守の生産性を確保するためには，プログラム規約を作成し遵守することが必須である．特にオープン環境においては，C言語に代表されるように，同一機能処理に対し様々な記述形式が可能であるなど，自由度が非常に高い．したがって，規約の整備だけでなく，如何に遵守させるかも重要となる．

規約として特に注意が必要なのはC言語・SQL言語のプログラム規約である．どちらもコーディングに個人差がでるだけでなく，障害対応・効率について大きな違いがでてしまう．個人的な差が出ないように規約を充実することが必要である．さらに，メモリ操作の部分を共通関数化してC言語特有のメモリリークを防止し，インデックスを経由したアクセスを行い，SQL処理での効率低下を防止するなど，問題発生を事前に防ぐための規約も盛り込んでおく必要がある．

遵守の手順としては

1) 規約およびサンプルプログラムを作成する．

2) 開発メンバへの教育，代表プログラムおよび障害記録を分析し注意事項などをフィードバックする．

これが一般的な流れである．しかしこの流れは，教育・注意という一方向だけの指導であり，これだけで規約を遵守することはできない．これに加え，プログラムチェッカ（lint）や独自の規約チェックプログラムによる機械的なチェック，および機械

的チェック不可部分についてのチェックリストによる確認を義務づけ，開発要員自身に遵守を意識付ける必要がある．さらに特別な体制を敷き第三者による重要部分についての検証を行うべきである．後工程でのテスト負荷・手戻り負荷を考えると，規約内容を充実させ，徹底的に遵守させることの効果は大きい．

6.2 プログラミング・単体テスト環境

メインフレームにおける開発では，単体テスト/結合テスト/システムテストと各工程に対し一つの環境を設定し，その環境内で整合性を取りつつテストを行っている．オープン環境では自由度が高く，目的に応じた複数環境の設定が可能である．特にプログラム開発，単体テストにおいて複数環境を設定することにより，生産性の向上が可能である．しかし，リソース面の管理強化が重要となる．

まず最初のポイントは，SW ライセンス数の見積もりといかに有効に活用するかである．見積りについては，各サブシステム毎のピーク時期，工程毎必要となる SW，個人使用かグループ使用か，などの見極めを行い調達計画を立てる事が必要である．大規模開発においては要員の異動（着任/離任）が度々発生する．そのためライセンス数が不足したり，インストールなどの環境設定負荷が増大する．ライセンス数を管理するだけでなく要員計画・開発状況と連動してのライセンス数の見直し，および環境設定負荷軽減のための移動プロファイル[*10] の採用/VB 開発コーナなど目的別の座席を設定し，要員と環境との分離を図る必要がある．

単体テストはモジュール毎のテスト環境とする事により，環境の問題による障害を防ぐことができる．特にこの工程は要員数がピークであり効果は大きい．大規模開発ではモジュール数が多くなるため，環境設定のパターン化・自動化を行い，設定負荷を減少させること，および消去の管理が必要となる．重要モジュールの環境を残存させる，あるいは復活可能とすることにより後工程での活用も可能となる．

6.3 開発支援ツール

大規模開発において，プログラムを構成するソースをどのように管理するかは一つの大きな課題である．開発に多数のプログラマが介在するため，「でき上がったプログラムがどのソースをコンパイルして作られたのか解らない」などのトラブルが多々発生する．今回の開発においてもそのソースファイル数が，PC 系で約 1000 ファイル，UNIX 系で 6000 ファイルにまで及ぶため，ソース管理が必須であった．

市販されている開発支援ツールの検討も行ったが，PC/UNIX を同じインタフェースで管理する物がなく，独自のツールを開発し使用している．ツールでは，以下の仕掛けを内部的に使用し，ソースのバージョン管理も同時に行っている．

1） SCCS（Source Code Control System）
　UNIX の標準的なソースファイルバージョン管理システムで，コマンドとして提供されている．

2） VSS（Visual Source Safe）
　VisualBasic にバンドルされているソースファイルバージョン管理システムである．

また，このツールでは，ソースの登録時に自動的にコンパイルを行う機能，前のバージョンとの差分情報を自動プリントアウトする機能などにより，開発担当者の負荷

軽減，および品質維持に貢献している．

この他にも，以下のような独自の管理ツールを作成・運用する事により開発をサポートしている．

1) プログラム製造環境作成ツール

環境作成依頼書（EXCEL シート）に基づき，プログラム製造環境を作成し，雛形ソースファイルを作成する．また，作成した環境は，開発担当者のみのセキュリティ設定を行う．

UNIX の環境作成依頼の場合は，同時に単体テスト環境用 ORACLE へのユーザ登録も行う．

2) プログラム構成管理ツール

本番環境へのリリース対象プログラム（実行ファイル）を解析し，以下の情報を取得する．

① プログラムを構成するソースファイル名およびそのバージョン情報
② 使用している共通関数名
③ 使用している ORACLE テーブル・項目名
④ 使用している区分値

取得した情報は，ACCESS にて管理し，各種インパクトレポートに活用する．例えば，ORACLE の項目を変更した場合，影響を受けるプログラム調査など．

3) プログラムリリース管理ツール

プログラムを本番環境へリリースする際，上記 2）の構成管理ツールとリンクし，プログラムリリース時に更新されたソース情報，また更新されたプログラム設計書情報などを，リリース履歴として管理する．

これらツール群は，開発開始時に全て揃っていたわけではなく，その都度の必要性によって，補完された．本来であれば，要員管理・スケジュール管理など，プロジェクト管理との統合的な管理が望ましい姿であるが，増改築のなかで実現には至っていない．オープンシステムでの大規模開発をスムースに行うためには，これらツールを含む統合的な管理手法が必須であると考える．

7. おわりに

オープン環境でのミッションクリティカル・システムの構築事例について述べたが，事例で示したように，UNIX サーバを用いたミッションクリティカル・システムの構築は充分可能であり，今後，ますますオープン環境での導入が増えてくると考えられる．

さらに，HW ベンダからは WindowsNT サーバで拡張性・可用性に優れた機種が市場に投入されてきており，今後は NT サーバ環境においてのミッションクリティカル・システムの構築へと続いていくものと思われる．

しかし，システム構築にあたり，オープン環境が故に必要となる作業，オープン環境が故に発生する問題が存在している．また，メインフレームで蓄積してきたノウハウが活かされておらず，余分な作業が必要になったり，余分な問題が発生していることも事実である．詳細は割愛しているが，事例システムの構築にあたっても各 HW/

SW ベンダとの折衝を重ね幾多の壁を乗り越えてきている．

ミッションクリティカル・システムの構築にあたっては，ノウハウの流布，問題発生箇所の事前予測，問題発生時の迅速な対応を行うための技術面でのプロジェクトマネジメントが非常に重要であり，本稿が今後，オープン環境でのミッションクリティカル・システムの導入・構築に携わる読者の御参考になれば幸いである．

＊ 1　IEEE：
（米国）電気電子学会」の略．1963 年に創設され，会員数 32 万人以上をほこるエレクトロニクス関係で世界最大の学会．

＊ 2　LINC：
日本ユニシス(株)が提供する，オンライン・トランザクション処理を行う基幹業務アプリケーションを開発，運用する為の支援環境．大規模データベース・システムはもとより，小人数で運営する事業所クラスのシステム，クライアント・サーバ処理の開発，イントラネットへの接続も可能にする統合開発環境．

＊ 3　Forte：
Forte Software Inc が開発したクライアント・サーバ型ミッションクリティカル大規模アプリケーション開発・統合環境．

＊ 4　CORBA：
Common Object Request Broker Architecture の略．ブジェクト指向技術の標準化，普及をすすめるため，1989 年に設立された業界団体の Object Management Group（OMG）が規定した分散オブジェクト技術の仕様．異機種分散環境上のオブジェクト（プログラム部品）の間でメッセージを交換するためのソフトウェア（ORB と呼ばれる）の仕様を定めている．

＊ 5　ORB：
Object Request Broker の略．マシン上に分散して存在するオブジェクト（プログラム部品）間で，データや処理要求などのメッセージをやりとりする際に用いられる仲介ソフトウェア．

＊ 6　マルチスレッド：
一つのアプリケーションがスレッドと呼ばれる処理単位を複数生成し，並行して複数の処理を行うこと．いわばアプリケーション内でのマルチタスク処理．マルチタスクと同じように，CPU の処理時間を非常に短い単位に分割し，複数のスレッドに順番に割り当てることによって，複数の処理を同時に行っているようにみせている．

＊ 7　RAID：
Redundant Arrays of Inexpensive Disks の略．複数のハードディスクをまとめて 1 台のハードディスクとして管理する技術で，データを分散して記録するため，高速化や安全性の向上がはかられる．専用の HW を使う方法と SW で実現する方法があり，高速性や安全性のレベルにより，RAID-0 から RAID-5 まで六つのレベルがある．
RAID-1 はミラーリングと呼ばれ 2 台のディスクにまったく同じデータを同時に書きこむ方式．片方が破損しても，もう一方からデータを読み出せるのでシステムは問題無く稼働しつづけることができる．RAID-5 はデータからパリティと呼ばれる誤り訂正符号を生成し，データとともに分散して記録する方式．データだけでなくパリティも分散する方式で現在最も普及している方式である．

＊ 8　NIS＋：
Network Information Service，Sun Microsystems 社が開発した，ネットワーク上の複数のコンピュータ間でユーザ情報を共有するシステム．

＊ 9　DNS：
Domain Name Service，IP アドレスをもとにホスト名を求めたり，その逆を求めたりすることができる．

＊10　移動プロファイル：
クライアントを共用端末として利用する場合に使われる機能．WindowsNT にログオンした場合，利用者毎の個別の情報（プロファイル）が参照され個人の環境設定が行われる．このプロファイルをサーバ側で一元管理することによりログイン時にサーバ上のプロファイルを参照し個人毎の環境設定が可能となる．

執筆者紹介　西　谷　　浩（Hiroshi Nishitani）

1984年東京大学経済学部経済学科卒業．同年日本ユニシス(株)入社．シリーズ2200での金融システムの開発/導入に従事．1990年より，金融機関向けにオープンシステムでの業務アプリケーション開発/利用技術支援/システム基盤の構築に従事，現在に至る．第2開発センター開発1室に所属．

勝　俣　英　善（Hideyoshi Katsumata）

1987年日本ユニシス(株)入社，Aシリーズでの金融機関向け国際系パッケージの開発/導入に従事．1990年より，金融機関向けにオープンシステムでの業務アプリケーション開発/通信パッケージ開発/利用技術支援/システム基盤の構築に従事，現在に至る．第2開発センター開発推進室に所属．