レガシーシステムとオープンシステムを アプリケーション連携するゲートウェイ開発

Development of Gateway Carrying out Cooperation of Applications on Legacy System and Open System

江　藤　　篤

要　約　金融機関において一部の業務系システムのオープンシステムによる再構築は，メインフレームとオープンシステム間でアプリケーション(AP)連携を実現することが必要となる．

本稿は，アプリケーション連携を実現するにあたって，必須要件である

① 通信プロトコルの選択

② APサーバとの連携

を実現させるため基盤ソフトウェアの選定と，ゲートウェイ(GW)サーバの開発事例報告である．通信プロトコルの選択は，それまでのSNAに代わって，TCP/IPでのMQシリーズ接続とし，一方APサーバの通信基盤にはSystem ν［nju:］を採用した．GWサーバには，MQからSystem ν［nju:］へのラッピングの実装を行い，基盤ソフトウェアには，GWサーバのシステム要件実現のために最適と考えられるものとして，TPモニタとしてTUXEDOを，リソースマネジャーにはMRMを選択した．

本事例では，これらの実装における課題とその解決法について報告する．

Abstract　In restructuring some application systems within the financial organizations using open systems technologies, it is necessary to implement the cooperative processing between applications running on a mainframe system（or legacy system）and an open system.

This paper discusses the selection of underlying software that facilitates to realize an essential condition including the selection of communication protocols and the cooperative processing with applications servers, and a case study on the development of a gateway (GW) server, to implement the applications cooperation above mentioned.

In investigating the system requirements, we adopted MQ (Message Queuing) based connection over the TCP/IP protocol suits instead of the SNA protocols used in the past as the communication protocol, and System ν［nju:］as an software infrastructure for the interprocess communication to the applications server, respectively.

In developing the gateway server, we implemented the wrapping from MQ series to System ν［nju:］, and adopted TUXEDO for the TP monitor and MRM for the resource manager as the underlying software, both of which are considered as optimal.

This paper also reports problems encountered in the development process and measures taken to resolve them.

1. はじめに

近年のコンピュータシステムは分散，オープン化が大きな流れとなっている．このような環境に対応し得る内部ネットワークが必要であり，接続形態も，従来はファイル転送やトランザクション転送が主体となっていたが，最近ではCORBAによるオ

ブジェクト分散やMQによるメッセージ転送が求められるようになっている．一方で，基幹業務はレガシーシステムに膨大なソフトウェア資産を残しつつこの環境に対応することが求められている．

このような環境に対応するために，既存システムのアプリケーションとオープン系アプリケーションが連携できるゲートウェイサーバ（以下GWサーバと略す）が求められた．開発にあたり，次の二つの重要な課題があった．一つは通信プロトコルの選択，もうひとつは実装システム基盤の選定である．

アプリケーション連携を，MQシリーズ[*1]とGWサーバのシステム基盤により実現させたが，実装においていくつかの課題に直面した．本稿ではその際の問題点と，その時にとった対応策について報告する．

2. 開発の背景

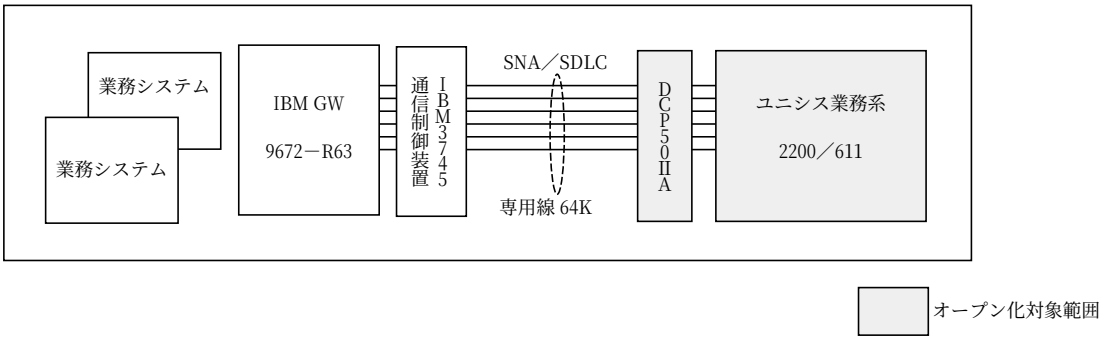

図 1　現行システムのネットワーク構成

図1に示すように，ユニシス業務系のオープン化を受けて，IBM社メインフレームとの接続インタフェースをどう決定するか，さらにオープンシステムの基盤をどう構築するかといった命題を解決しなければならなかった．

接続インタフェースに関しては，現行のSNA[*2]/SDLC[*3]も含め，オープンシステムの主要な標準通信プロトコルとしていくつかの候補を上げ，この中からさらにIBM社メインフレームに親和性のあるものを絞り込むことにした．

オープンシステムの基盤として第1の候補としてあがったのが，新しいテクノロジーであるCORBA[*4]である．CORBAによる分散オブジェクト技術やエージェント指向を取り入れることで，プログラムの部品化やアプリケーション開発の生産性向上，アプリケーションの位置透過性の実現を狙うことが主眼であった．

したがって，次に示すIBM社メインフレームとの接続プロトコルに関しても，CORBA接続が第1候補に上がった（以下，要求度合いの高い順）．

① CORBA接続
② MQシリーズ　ON TCP/IP
③ SNA ON TCP/IP
④ TCP/IPによるLAN接続
⑤ SNAによるLAN接続

⑥　SNA/SDLC（現行の接続方式）

IBM 社とはまず CORBA 接続の可能性を検討したが，結果的に CORBA 製品のリリース時期がすり合わないなど，時期早々という理由から CORBA 接続は見送られた．代わりに IBM 社から提案されたのが MQ シリーズ ON　TCP/IP である．MQ シリーズは各種オープンプラットフォームにも提供されている．MVS[*5] 版の MQ シリーズは既に接続実績があるということであったが，唯一 MQ シリーズによる通信のパフォーマンスが懸念された．

パフォーマンスに関しては，IBM 社から提示された 1 ローカルキュー[*6] あたりの性能目標値があったので，実測による性能見積もりをしてその裏付けをとった．性能見積もりの方法は，実験マシンでの実測値を基礎データとし，本番機と実験機との TPC-C[*7] 値で換算した．

M＝m×本番機 TPC-C 値/実験機 TPC-C 値

M：本番機での性能値　　m：実験機での性能値

（注）この性能値は MQ アプリケーション処理を含んだ値である．

見積もり結果は，充分に性能目標を達成できるものであった．

3. レガシーシステムとオープンシステムの AP 連携の必要性

IBM 社メインフレームとの CORBA 連携は断念したものの，ユニシス AP サーバの基盤として，弊社製品 System ν［nju：］[*8] を採用した．その理由としては，System ν［nju：］の ORB[*9] 機能が AP サーバ間の通信機能に最適であると考えられたからである．ORB の通信プロトコルには IIOP[*10] が実装されており，これにより AP サーバ間の通信を行うことができる．さらに，AP 間でのメッセージ連携を ORB で実装することで，オブジェクトの位置透過性やロードバランス，リカバリなどが実現され，フレキシブルで高パフォーマンス，かつ信頼性の高い通信が可能となる．

ユニシス業務系の通信基盤として System ν［nju：］が採用され，IBM 社メインフレームとの通信インタフェースに MQ シリーズが決定したことで，ユニシス業務系と IBM 社メインフレーム間の AP 連携に，GW サーバの構築が必要となった．

アクセスハブの 1 局化という観点から，GW サーバを置かずにユニシス業務系の各 AP サーバが直接 MQ シリーズで接続する形態も考えられたが，表 1 に示すように比較検討した結果，GW サーバ方式の方が，MQ の運行を集中管理ができる点や，システムの柔軟性に優れているという点などから，GW サーバをアクセスハブとしてフロントに置くこととした．

表 1　接続方式比較

接続方式	集中管理	柔軟性/拡張性
IBM 社 GW 直結方式	①IBM 社　GW への負担が大 ②各 AP サーバ毎に MQ の運行管理が必要．	①AP の追加や変更に難がある． ②各 AP サーバ毎にネットワーク変更や各種プロトコル対応が必要．
GW サーバ方式	①MQ シリーズに関する運行管理を集中して行うことができる．	①AP の追加や変更への柔軟性を与える． ②ネットワーク変更や各種プロトコル対応が容易である．

GW サーバの位置付けを図 2 に記述する．必然的に GW サーバをフロントに置く以上，GW サーバのスループットを最大にすることは必須要件となった．

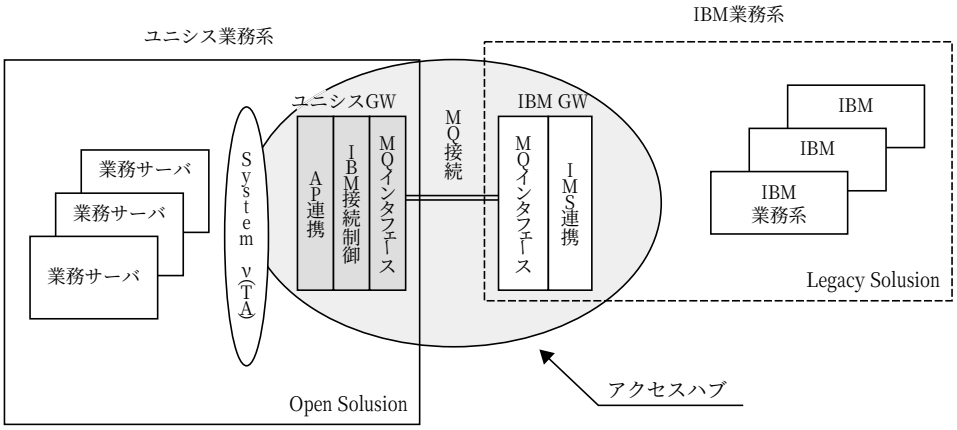

図 2　GW サーバの位置付け

4.　GW サーバ・システム要件実現のための課題整理

4.1　GW サーバ機能要件

ミッションクリティカルなシステムにおける GW サーバとしては，一般的には以下のようなシステム要件が挙げられる．

① ノンストップ（24 時間 365 日連続稼働）
② 高可用性
③ 障害リカバリ
④ 運行監視
⑤ データの保全
⑥ 高パフォーマンス

GW サーバには，多種多様な通信プロトコルに対応し，かつ各種 AP インタフェースとの組み合わせが可能となるという利点もある反面，フロントに GW サーバを置くことによるターンアラウンドタイムのロスは無視できない．高パフォーマンスや，フェイルオーバなどの障害リカバリ機能は GW サーバの最も重要な要件である．

今回の GW サーバの開発では，図 2 に示すように，IBM 業務系とユニシス業務系の間の AP 連携を行うために，主な機能として以下のものが求められた．

① MQ シリーズとのインタフェース
② AP サーバとの連携機能
③ 障害監視と障害リカバリ
④ 連続稼働
⑤ 高パフォーマンス

4.2　システム要件実現のための課題

オープンシステムで上記機能を実装するためには，下記の課題があった．

■基盤ソフトウェアの選定

■MQ から System ν［nju：］へのラッピングの実装方式

1） 基盤ソフトウェア選定

GW サーバの機能要件実現のために，表 2 に記述されている基盤ソフトウェアの選定が必要となった．

表 2　各機能と必要とされる基盤ソフトエウェアの対応関係

機　　能	必要とされる基盤ソフトウェア
MQ シリーズとのインタフェース	・MQ シリーズ
AP サーバとの連携機能	・System ν［nju：］
障害監視と障害リカバリ	・HA 製品　・統合システム管理　・ジョブ管理
連続稼働	・ジョブ管理

2） MQ シリーズから System ν［nju：］へのラッピングの実装方式

MQ シリーズで受けたメッセージをどの System ν［nju：］へ渡すのかという実装方式（図 3 参照）についてはこれまで事例がなく，今回始めて検討することになった．

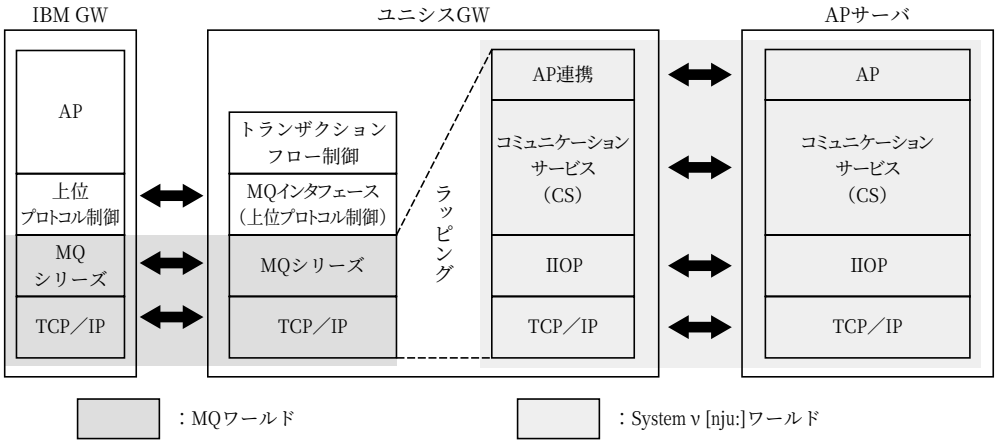

図 3　MQ シリーズから System ν［nju：］へのラッピングのイメージ

5.　GW サーバ構築の技術的アプローチ

5.1　MQ シリーズとのインタフェース

MQ シリーズとのメッセージ受け渡し処理の中で使用される基盤インタフェースを実装するためには，TP モニタや RM（リソースマネジャー）が必要となる．

基盤ソフトウェアの中でも TP モニタの選定は，最も難航したところである．IBM 社 GW との接続インタフェースでは MQ シリーズ接続だけでなく，MQ シリーズの上位に企業内接続手順を実装する必要があった．これはトランザクション転送の手順である．弊社がミッションクリティカルな分野で開発実績のある TUXEDO[*11] を推した．

さらに，TUXEDO 補完ミドル製品として TSAS[*12] や，回復型共有メモリの RM

としてMRM[*13]を弊社は持っており，納期を順守するために現実的な選択としてTUXEDO，TSAS，MRMを選択した．

図4にMQインタフェースのTUXEDO実装図を示す．MQシリーズとのメッセージ受け渡しの実装方法として，MQシリーズのトランザクション管理を，TUXEDOが持つXA[*14]準拠の2フェーズコミット機能に統合することも考えられたが，当時リリースされたばかりのSolaris版MQシリーズ5.0のフィジビリティが不明であったことと，上位のプロトコル制御が通番管理機能を装備していることもあって，あえてXA準拠の2フェーズコミット機能ではなく，MQシリーズ，TUXEDOそれぞれのコミット機能を個別に使用することにした．

MQインタフェース処理の設計上のポイントとしては，欠落があってはいけないが，重複はありえるということを前提とした．重複データは，双方の上位プロトコル制御の通番管理機能により自動的に読み捨てることにした．

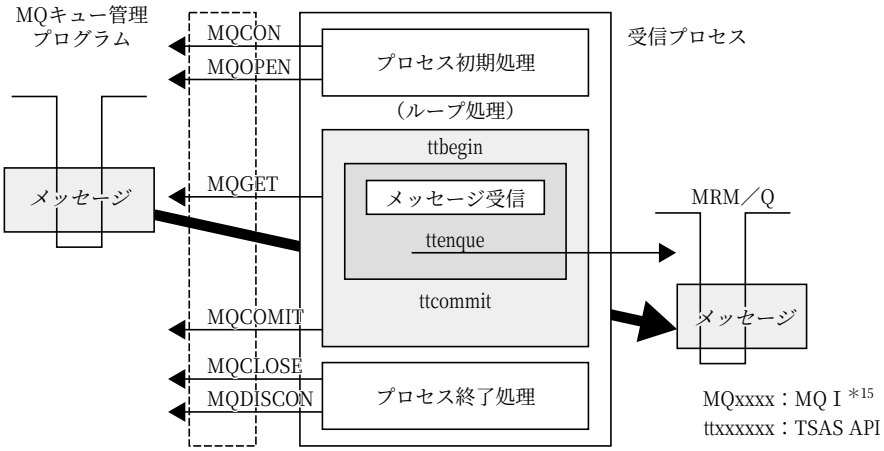

図 4　MQインタフェースのTUXEDO実装（受信処理の例）

5.2　APサーバとの連携機能

5.2.1　MQからSystem ν［nju:］へのラッピング

MQからSystem ν［nju:］へのラッピングは，図5に示すようにMRM/Qを媒体として行うことにした．MRM/Qとは，共有メモリRMであるMRMを媒体とした回復型メッセージキューイング機構である．メッセージの受け渡しを行うだけでなく，メッセージ保存機能や再送処理を目的とした再読み込み処理などが可能である．

MQシリーズからSystem ν［nju:］に直結できない最大の理由は，MQインタフェースの中の上位プロトコル制御を実装する上での制約である．IBM社GWとのメッセージの送達保証をGWサーバで行うためには，そのメッセージを格納するリソースマネジャーが必要となる．

5.2.2　APインタフェース

APインタフェースでは，System ν［nju:］のORBによる通信機能を使用した．トランザクションフロー毎にIDL[*16]を記述し，このIDLでAPサーバとのインタフェースが決定される．クライアント側のプロセスをUA[*17]，サーバ側のプロセスを

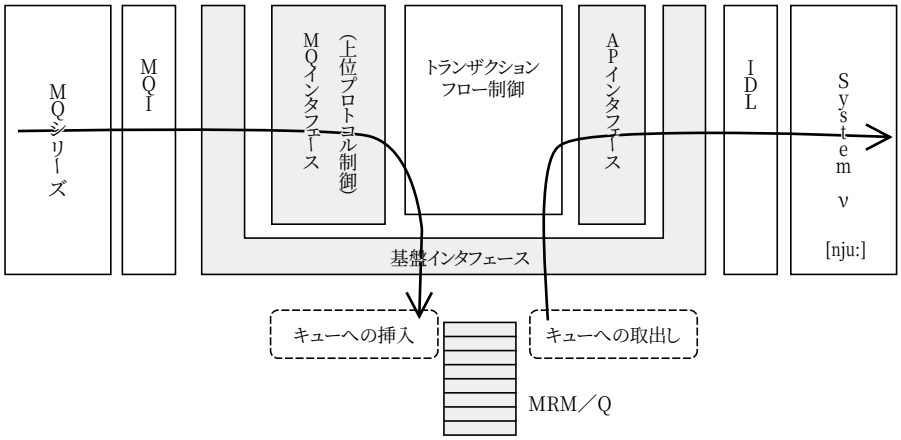

図 5　MQ シリーズから System ν［nju：］へのラッピング（INPUT の例）

AA[*18] で実装した．

1）　クライアントプロセス（UA）の実装

GW サーバ側から，AP サーバ側にメッセージを送信するプロセスは，System ν［nju：］の UA で実装した（図 6）．さらに，MRM/Q からメッセージを取り出すため，同時に TUXEDO クライアントでもなければならない．同一プロセスで System ν［nju：］と TUXEDO を実装した例はこれまでにないということであったが，プロトタイプを開発して実証した．

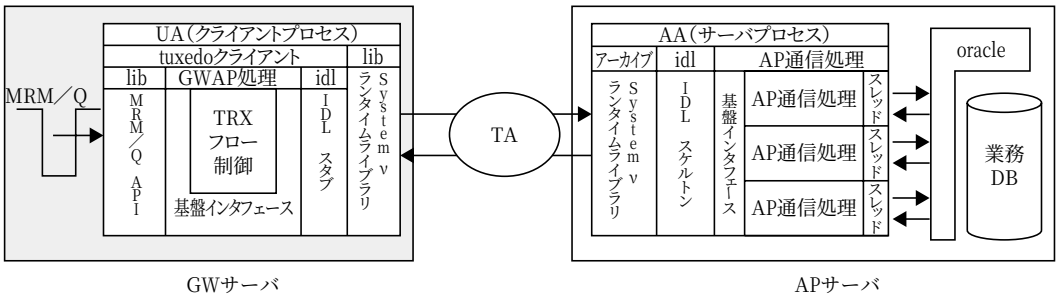

図 6　UA 処理

2）　サーバプロセス（AA）の実装

AP サーバ側からメッセージを受信するプロセスは，System ν［nju：］の AA で実装した（図 7）．AP サーバから受け取ったメッセージを MRM/Q に書き込むため，UA と同様，同時に TUXEDO クライアントでなければならない．AA の場合は，UA がシングルスレッドで動作するのとは違い，マルチスレッド環境での動作が前提となる．TUXEDO（当時 6.3 J）はシングルスレッド環境下でしか動作しなかったため，同時に 1 スレッドしか動作できないようにする制限を入れる必要があった．

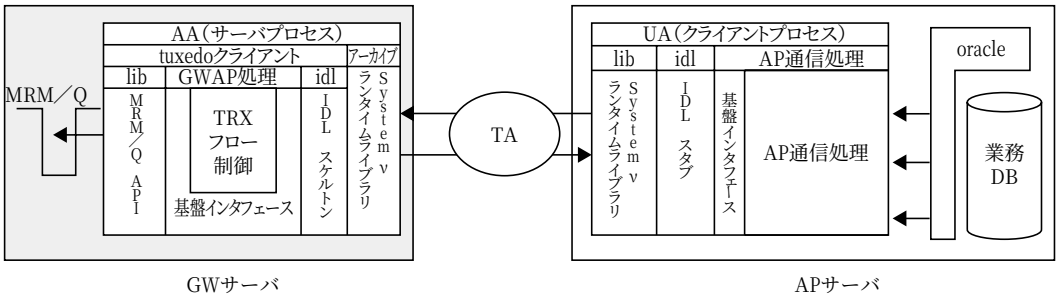

図 7　AA 処理

5.3　障害監視とリカバリ

基盤ソフトウェアの導入を前提に，表 3 に示すような障害監視機能や障害時のリカバリ機能を実現する実装方式を検討した．

表 3　障害監視機能や障害時のリカバリ機能

障害の種類	基盤ソフトウェア	実装方式
ノード障害	FirstWatch[*19]	FirstWatch のフェイルオーバ機能を使い，GW サーバ障害時には待機系の GW サーバに切り替え，業務を続行する．
ネットワーク障害	MQ シリーズ， System ν［nju：］ OpenView[*20]	MQ シリーズや System ν［nju：］で障害検知を行い，障害時は GW アプリケーションが障害イベントを OpenView の統合監視コンソールに表示すると同時に，メッセージの再送処理などを行う．
IBM 障害	MQ シリーズ	MQ シリーズや MQ インタフェースで障害検知を行い，基本的には MQ ネットワーク障害と同様の処理を行う．
AP サーバ障害	System ν［nju：］	System ν［nju：］や AP インタフェースで障害検知を行い，基本的にはネットワーク障害と同様の処理を行う．
プロダクト障害	OpenView	OpenView の性能管理機能を使用し，MQ，TUXEDO，MRM などのプロダクト障害時には障害イベントを OpenView の統合監視コンソールに表示すると同時に，定型のリカバリ手順を実行する．
イベント監視	OpenView	GW サーバ上で発生する障害イベントや運用イベントを OpenView の統合監視コンソールに表示する．
キュー障害		GW サーバ内の業務処理の障害は，キュー障害とみなされ当該キューが自動閉塞される．その際，障害リカバリ操作のためのコマンド操作画面や状況照会画面を提供する．
ベーシック リカバリ		MRM のリカバリシステムに致命的な障害が発生した場合は，AP サーバのデータベースからリカバリ情報取りこみ，MRM を復元する．

5.4　連　続　稼　働

24 時間 365 日連続稼働を実現するためには，システムを停止せずに業務毎の通番初期化やデータのバックアップなどのデイリー更新処理が必要である．GW サーバでは，JP1[*21] の定時起動処理を使用し，この操作を無人運転で行う．この際，業務毎のデイリー更新処理中は，IBM 社 GW や AP サーバからの誤動作に対して，防御機能が働く仕掛けを用意した．

5.5 パフォーマンス

GW サーバ内のパス構成は，送受信とも業務毎に 4 系列設定され，最大 4 多重の並行処理を可能とした（図 8）．また，一つのトランザクションフロー処理でも，IO 回数の削減や，多重処理におけるリソース排他がおきないよう，考慮されている．さらに，MRM のバックグランド処理では 4 多重分のトランザクションログを一括更新するなどして，IO 回数の削減をはかっている．パフォーマンスは性能目標値を満足するものであった．

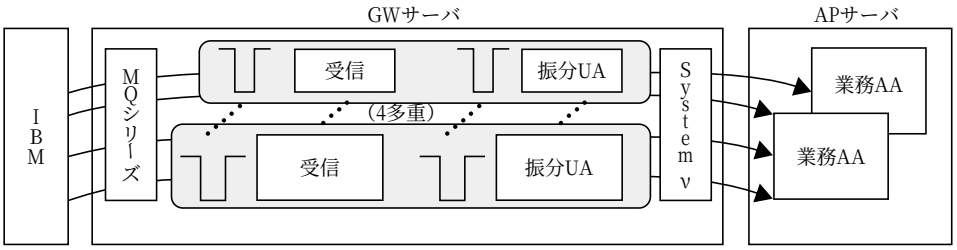

図 8　トランザクションフローの多重処理（1 方向受信の例）

5.6 基盤ソフトウェアの選定結果

検討の結果，表 4 に示すとおり，下記の基盤ソフトウェアを決定した．

表 4　選択された基盤ソフトウェア

インフラの種類	選択された製品	適用される機能要件
通信ソフトウェア	MQ シリーズ（IBM 社製品）	MQ シリーズとのインタフェース
CORBA	System ν［nju：］（当社製品）	AP サーバとの連携機能
HA 製品	FirstWatch（VERITAS 社製品）	障害監視，障害リカバリ
統合システム管理	OpenView（HP 社製品）	障害監視，障害リカバリ
ジョブ管理	JP 1（日立社製品）	連続稼働 障害監視，障害リカバリ
TP モニタ	TUXEDO（BEA 社製品）	MQ シリーズとのインタフェース AP サーバとの連携機能 障害リカバリ
TUXEDO 補完ミドル	TSAS（当社製品）	
リソースマネジャー	MRM（当社製品）	

6. GW サーバの概要と特徴

6.1 GW サーバの概要

G/W アプリケーションは，図 9 のようなシステム構成とした．

■MQ インタフェース

MQ インタフェースは，MQI を使用した MQ 制御と上位プロトコル制御を行う．

■AP インタフェース

AP インタフェースは，System ν［nju：］の ORB 機能を使用し，AP サー

バとのメッセージの送受信を行う．

■GW 運用管理

GW 運用管理は，MQ の運行監視や MQ の構成管理機能，MRM キューの状況監視やキュー操作コマンドなど，GW サーバの運行監視機能を GUI で提供する．さらに，統合監視コンソール機能として OpenView の ITO[*22] エージェントへのイベント連携機能，GW ジョブ監視機能では各種 GW 運行シェルを JP1 のジョブネットとして実行する．

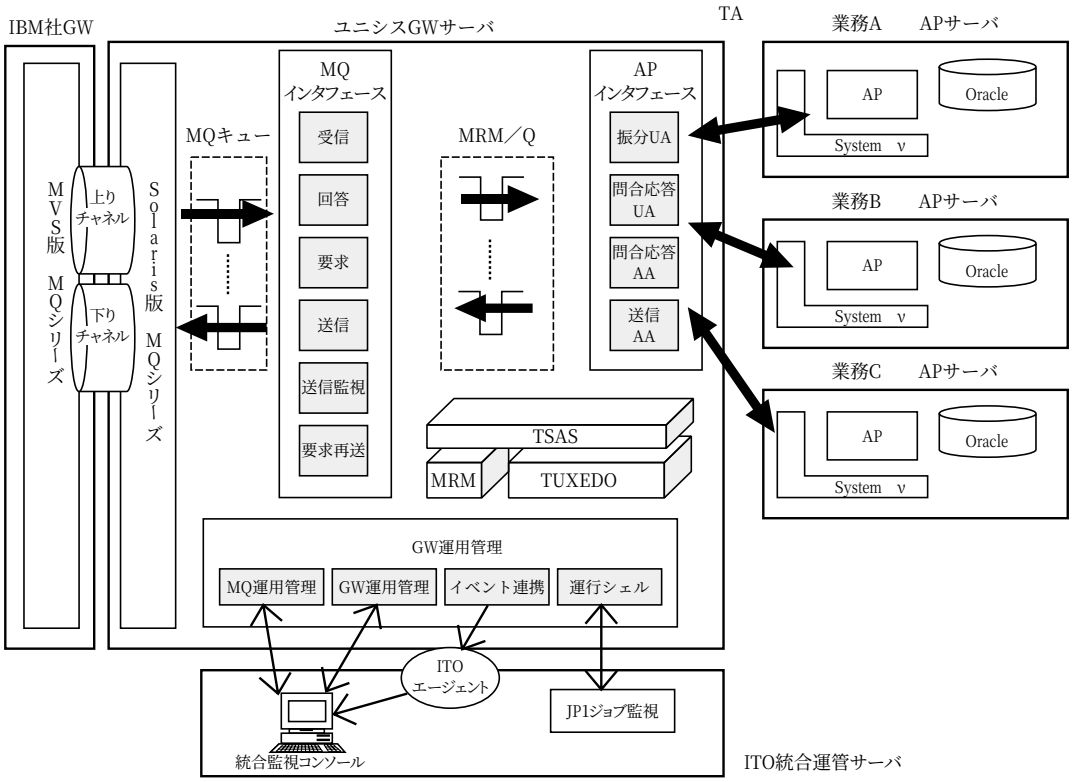

図 9　IBM 接続 GW サーバ機能概要図

6.2　GW サーバの特徴

この GW サーバの主な特徴として，以下のような点が挙げられる．

1）　フレキシブルなネットワーク・アクセスハブ

図 10 のように，MQ インタフェース，AP インタフェースとも，MQI や System V［nju：］関数を直接使用することはしないで，GW 基盤インタフェースを間にはさんだ．この GW 基盤インタフェースによって，電文種別単位に IBM 社 GW と AP サーバ間のトランザクションフローが決められる．

キュー構成の変更や追加，あるいは新規サービスの追加などは，GW 基盤インタフェースの MRM 構成定義変更で行うことができ，また，AP サーバ間との新規インタフェースの追加には，GW 基盤インタフェースの通信アダプタが自動生

成される．

このように，GW サーバは通信プロトコルを意識することなく，パスの追加，変更や新規サービスの追加などに，柔軟な対応ができるネットワーク・アクセスハブといえる．

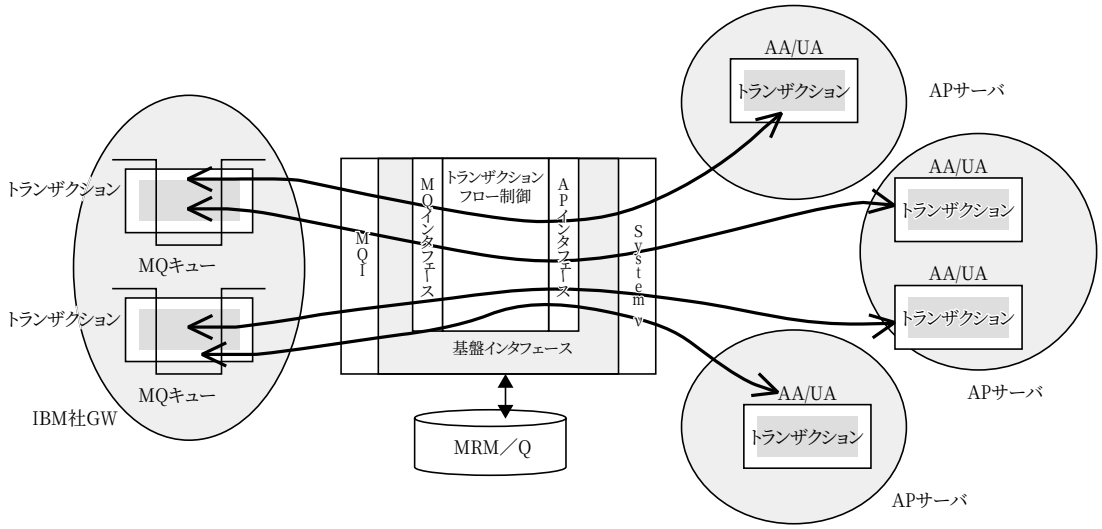

図 10　トランザクション・フロー制御

2)　高信頼性

図 11 のように，サーバ障害時は，FirstWatch のフェイルオーバ機能で，待機系のサーバに自動的に切り替わる．GW サーバ内のデータは，100% 保証されるリカバリ機能を持つ．各種障害時，データは MRM のリカバリ機能により復旧されるが，MRM が致命的な障害を起こした時でも，AP サーバのデータベースからリカバリー情報を取得して，MRM を復元させる機能を持つ．

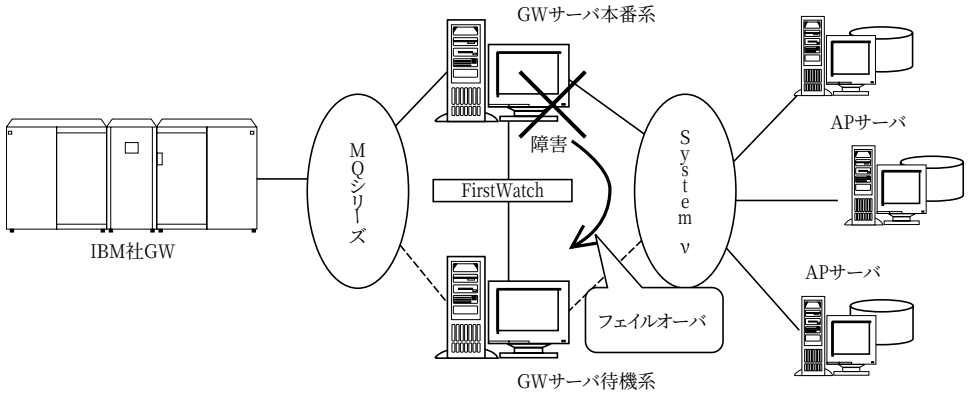

図 11　フェイルオーバ機能

3） GW サーバの独立稼働/連続運転

GW サーバは，IBM 社 GW や AP サーバの障害時，障害マシンを切り離し，アベイラブルな IBM 社 GW や AP サーバだけで，運行を継続することが可能である．回復後，支障なくそれまで GW サーバに蓄積されたデータの送受信処理を再開する．連続運転にも対応可能である．

4） ITO 統合監視コンソールとの親和性

MQ シリーズの運用管理機能と GW 内運用管理機能を，OpenView の VIEW 環境上に GUI で実現．ITO 統合監視コンソールからの操作が可能である．

7. GW サーバ構築の考察

GW サーバの論理構造は，図 12 のようなレイヤー構成になっており，トランザクションフロー制御は，GW サーバの中核に位置付けられている．今回の通信インタフェースには，MQ シリーズ，System ν［nju:］の ORB が実装されたが，可能性としてフロントとバックエンドには，SNA/TUXEDO など様々な組み合わせも考えられる．

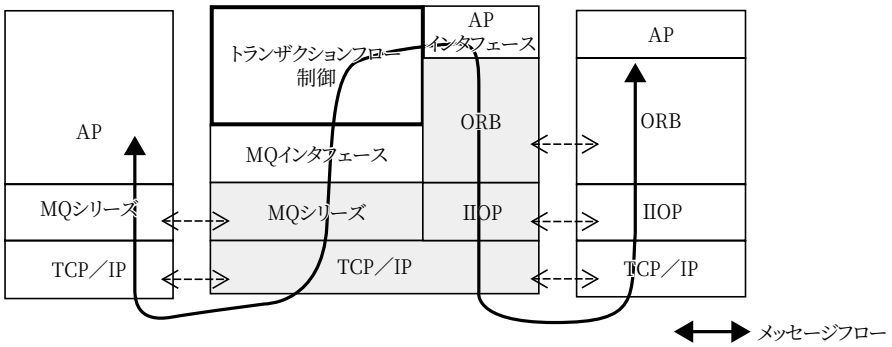

図 12　GW サーバの理論構造

汎用的な GW サーバの実装であれば，図 12 のトランザクションフロー制御は，各通信インタフェースから独立した構造にすべきであった．これには，プロトコル制御を隠蔽するような汎用通信アダプタがあれば可能であったと思われるが，コスト，期間などの制約から手をつけられなかったところである．

8. お わ り に

GW サーバの開発を終えて今回のプロジェクトを評価してみると，機能要件はもちろんパフォーマンス性能の点でも，当初見積もった目標値をクリアすることができたと考える．MQ シリーズによる接続テストは，様々なトラブルにも遭遇したが，ほぼ予定どおりに本番稼働を迎えることができた．

この GW サーバはフロントエンドに MQ シリーズを，バックエンドに System ν［nju:］を実装したが，言いかえれば，MQ インタフェースを System ν［nju:］ワールドへラッピングしたものともいえる．

今回はMQだけであったが，フロントにはMQ以外にも様々な通信インタフェースがありえるし，バックエンドは将来的に，他のCORBA製品も考えられるわけで，今後のゲートウェイ開発には，そうした多種多様なインタフェースの組み合わせが予想される．今後の方向性としては，CORBA連携を企業内ネットワークの標準プロトコルにという展開も充分考えられるところである．このGWサーバの開発が，その第1歩となれば幸いである．

＊1　IBM社が提供するキューマネジャー．
＊2　System Network Architectureの略．システムネットワーク体系．OSIに似た7層アーキテクチャを用いているが，1対1には対応しない．IBM社プラットフォームにのみ適用されるネットワーク機能．
＊3　Synchronous Data Link Controlの略．IBM社が制定したプロトコルで，SNAのデータリンク制御層のプロトコル．
＊4　Common ORB Architectureの略．共通オブジェクト・リクエスト・ブローカ．異質な複数のシステム間で，オブジェクトの移植性と相互運用を提供する．
＊5　Multiple Virtual Systemの略．1973年に発表されたIBM社のオペレーティングシステム．
＊6　キューマネジャー上に実在するキュー．全てのキューは最終的にはローカルキューに対応づけられる．
＊7　Transaction Processing Performance Council benchmarks-Cの略．TPCは，異なるベンダーの製品を公平に評価するために，トランザクション処理のベンチマークを定義したもの．TPC-Cはその中でもA，Bに比べて，比較的バリエーションがあって，処理が重たいアプリケーションをモデルにしたもの．
＊8　ユニシスのCORBA製品．
＊9　ObjectRequestBrokerの略．ORBはネットワーク間でオブジェクト通信を行う機構のことを指し，リモートオブジェクトの呼び出しを行うためには，クライアント，サーバ両サイドでORBを使用しなければならない．
＊10　Internet inter ORB Protocolの略で，OMG（Object Management Group）がCORBA 2.0の中で規定したORB同士を相互に接続するためのプロトコル．
＊11　BEA社が提供するTPモニターである．
＊12　日本ユニシスのTUXEDO補完ミドルウェア．
＊13　日本ユニシスの回復型共有メモリリソースマネジャー．
＊14　X/Open DTPモデルでの，リソースマネジャーとトランザクションマネジャー間のインタフェースを定めたもの．
＊15　Message Queue Interfaceの略．MQシリーズがアプリケーションに提供するAPIである．
＊16　リモートオブジェクトのインタフェースを特定言語に依存しない形式で記述するための言語である．
＊17　User Agentの略．System *v*［nju:］のクライアントプロセス．
＊18　Application Agentの略．System *v*［nju:］のサーバプロセス．
＊19　ハイアベラビリティ・システムを構築するためのフェイルオーバー管理ソフトウェア．
＊20　HP社が提供する統合ネットワークシステム運用管理ソフトウェア．
＊21　日立製作所が提供する統合システム運用管理ソフトウェア．
＊22　HP OpenViewの製品群のひとつで，ネットワーク上にあるUNIXシステムやWindows NTシステムなどの各種OS，及びその上で動作するアプリケーション，データベース等のソフトウェアの動作を中央から監視し，障害検知し対応する統合監視をおこなう製品である．

参考文献　［1］ D. T. Dewire,“多階層クライアントサーバコンピューティング”日刊工業新聞社 1998年1月31日.
［2］ 萩本順三, 福村真奈美, 不破康人　共著,“最新オブジェクト指向技術応用実践”, 1998年4月17日　第1版第2刷.
［3］ プロトコルハンドブック編集委員会編,“新プロトコルハンドブック”朝日新聞社 1996年7月25日　第4刷.

執筆者紹介 江　藤　　篤（Atsushi Etoh）

1950 年生．1972 年国立大分工業高等専門学校電気工学科卒業．同年日本ユニシス(株)入社．第 3 開発センター開発 3 室 3 G．金融外接パッケージの開発に従事．銀行 ANSER や ODEX-II，東証約定直結，NOBAS（Solaris 版銀行 ANSER）などの開発に参加．現在，OnNetSolurion の通信イネーブラ EnConne の開発に従事．

E-Mail：Atsushi.Etoh@unisys.co.jp