基幹システムのライトサイジング

株式会社 ニチリン

■ 執筆者Ｐｒｏｆｉｌｅ ■

西　山　哲　生

1996 年　(株)ニチリン入社
システム部システム課
システム開発業務担当
1999 年　～現在
情報管理部情報管理課
システム開発業務担当

手　塚　俊　雄

1981 年　(株)ニチリン入社
商品開発業務担当
1985年　システム部システム課
システム開発業務担当
1999年　～現在
情報管理部　部長

■ 論文要旨 ■

ここ１０年で広範囲となってきた社内のシステム開発技術の整理とランニングコスト低減を目的に，汎用機で稼働している基幹システムのダウンサイジングを Windows サーバ，ＣＯＢＯＬ，バッチファイル，ＩＩＳ（ＡＳＰ），ＳＱＬサーバを採用したシンプルな方法で自社で実施した.

実際のシステムではインターネットを使ったＥＤＩなど，追加した機能や改善点も多いが，新しい機能，仕組みを構築することよりも，現在の汎用機のシステムと同等以上の事を別のハードウェアで実現すること自体が主目的であるので，本論文では，どの様な機能のシステムを構築したかではなく，手法，選択した技術要素とその問題点などについて述べている.

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

当社は，液圧ブレーキホース・カーエアコンディショニングホースをはじめとする自動車用ホースの製造を国内で最初に手掛け，これらを主軸にした高機能ホースを主に製造する企業である．多くの企業と同様，数十年前から汎用機を使った自社開発の基幹システムを導入している．

使用している汎用機（FUJITSU-GS8400）を更新するにあたり，ここ１０年で社内の技術が汎用機とＰＣ系に分かれ，様々な非効率を抱えている点や，情報通信技術の進歩に効率的に追随する必要性などから，基幹システムのライトサイジング（ダウンサイジング）を行う方向で検討することとした．汎用機の更新に要する費用，工数をサーバで稼働するシステムの開発に振り向け，新システムとして再開発する事で上記の問題点の改善に加えて，ランニングコストを低減することが目的である．基幹システムはここ数年大きな拡張はなく，小中規模システムは極力サーバで構築してきた経緯もある．開発を外部ベンダーに任せたのでは，後々，頻繁にあるシステム変更が難しくなるため，開発は極力，社内で行うこととした．

上記の様に，この取り組みは新しい機能，仕組みを構築することよりも，現在の汎用機のシステムと同等以上のことを，別のハードウェアで実現すること自体が目標となる．よって，本論文では，どの様な機能のシステムを構築したかではなく，手法，選択した技術要素，問題点などに重点を置いて述べることとする．実際のシステムでは，インターネットを使った取引先とのＥＤＩなど，追加した機能や改善点も多い．

２．システム構成の検討

２．１　ＯＳの決定

まず，開発のベースとなる，ハード，OS，開発言語を決めなければならないが，一般的には安定性や実績から基幹システムをダウンサイジングする場合のＯＳは，UNIX が選択されることが多い．しかし，我々は次の理由から Windows を選択した．

(1)　生産現場のシステムで多くの Windows でのクライアントサーバシステムを開発している実績があり，安定性に大きな問題はない．
(2)　UNIX を選択した場合，社内技術が Windows と UNIX の２本立てとなってしまう．
(3)　我々の UNIX の技術知識は浅く，その習得の時間は残されていない．

２．２　ＤＢＭＳの選択

データベースサーバについては，本計画に先立って，汎用機の部品表を Microsoft SQL Server に連携して参照させるシステムを構築しており，一般的な部品展開では，ネットワーク型のＤＢを使った汎用機の数倍の処理速度（展開モジュールは VB で開発）となることが確認できていた．最新の高速なハードウェアを使えばレスポンスに問題が出ることはないはずである．

また，他のシステムで何種類かの古いバージョンの Microsoft SQL Server を使った稼働実績から，安定性についても問題はなく，Microsoft SQL Server を選択した．

２．３　ハードウェア構成

汎用機に比べると低い信頼性を，サーバでは何らかの方法でカバーする必要がある．

そのためにクラスタリングを検討したが，データベースサーバには使えないクラスタシステムがあったり，最近でこそ富士通のＦＴシリーズなど十分に練られた専用システムもあるが，２年前は十分な実績があるとは思えないものが多かった．無理をすると逆に安定性が損なわれる結果になりかねないため，サーバはRAIDと冗長電源のシンプルな構成とした．ただ，開発の手順やレスポンス，保守性を考慮し，サーバはある程度，機能別，サブシステム別に分割した．

また，万が一，ハードの復旧に時間のかかる大きな問題が発生したり，再現性が少なく原因究明が困難なトラブルがあった際等に備えるため，それぞれのハードの予備を確保することとした．

ハード構成概要

(1)　データベースサーバ（MicroSoft SQL Server）
(2)　電子帳票サーバ（ListWorks）
(3)　ＪＣＡ手順通信サーバ（CORDEX Server）
(4)　アプリケーションサーバ（３台）

※アプリケーションサーバではCOBOLやVBのバッチ処理，ASP(IIS)が動作する．

３．開発方法の検討

３．１　バッチ処理の開発言語の選択

開発言語については最も検討が難しく，将来性から選択するとJAVAということになるが，我々の知識の浅さやJAVAを選択した場合に遭遇するであろう未知の障害とその対応を考えると選択はできなかった．かといって，VisualBasic では将来のバージョンアップでソースの大幅な変更を余儀なくされる心配もある．

最大の懸念は，現在の汎用機の技術者が COBOL から他の言語に移行することであり，これに手間取れば計画自体が困難なものになってくる．COBOL はソースが冗長であるが，事務計算のバッチ処理の開発では，自由度の少なさ，シンプルさが逆に先々のメンテナンス性にメリットであるとも考えられる．ここは無理をせずに，COBOL を採用することとした．

COBOL コンパイラは８ビット時代から存在する海外製も調査したが，サポート面，日本での実績の不安から日本製を選択．その中でも「.NET」をサポートするなど，COBOL にも精力を注いでいる富士通の COBOL97 を第一候補として検討し，SQL のアクセス，レスポンス等の基本機能，性能に大きな問題もなく，我々の汎用機の COBOL とも互換性が高いなどのメリットもある，COBOL97（PowerCOBOL）に決定した．

３．２　ＪＣＬ（ジョブ制御言語）の代替

サーバでの開発においても，バッチ処理では複数の COBOL 実行プログラムやファイルのコピー，削除，比較，変換などのユーティリティを順に起動し，実行状況によって中断するなど，これまでの方法に準じたプログラムの制御方法が効率的である．このためには，

通常，汎用機 OS では JCL で行っている様な制御を Windows でも行う必要がある．

JCL の代わりになるものとして，UNIX や Linux で一般的な bash，perl の Windows 版や，WSH（Windows Shell Script）の利用が考えられたが，それ自体にバグがあった場合のこと，処理的にはあまり複雑な制御の必要性がないことなどから，MS-DOS の時代から慣れ親しんでいるバッチファイルを使うことにした．

バッチファイルは決して使いやすいとは言えず，言語としての様を呈していないが，シンプルな使い方をする上では十分である．後になってわかったことであるが，Windows2000 以降，コマンドインタープリタが拡張されており，変数の文字列の切り取りや，簡単な演算，日時の取得，サブルーチンに類似した使用法など，以前はフリーソフトを併用しないとできなかった多くの機能が追加され，思ったより高度な制御も可能である．

３．３　オンライン画面をＷＥＢシステムに移行する際の工夫

オンライン画面は，VB または PowerCOBOL 等を使用してクライアントサーバシステムとして開発する方法もあるが，保守性や，将来のモバイル対応への準備のため，多少の困難があっても WEB ベースのシステムとしたかった．

幸い，情報系システムで WEB ベースのワークフローシステムや，出荷実績や出庫実績を WEB で参照するシステムを開発していたため，Windows IIS 上の ASP での開発技術の蓄積があり，開発担当者は利点も欠点も十分把握している．しかし，今回は基幹システムであるので，正確性を期すために，クライアント側スクリプトの使用を極力避けたり，画面で参照した結果を元にデータベースを更新する場合でも受け渡されたデータでそのまま更新せず，再参照するなど，ブラウザ側の問題やクライアント側ソースの改変があった場合にも問題が起こりにくいような工夫を盛り込む必要があった．

なお，利用者のスムーズな移行への配慮と操作説明の手間を省く意味もあり，画面のレイアウトは現在のレイアウトに似せ，操作もマウスを使わずにファンクションキーを含め現在の操作性を基本とした．（**図１**　WEB 画面例参照）

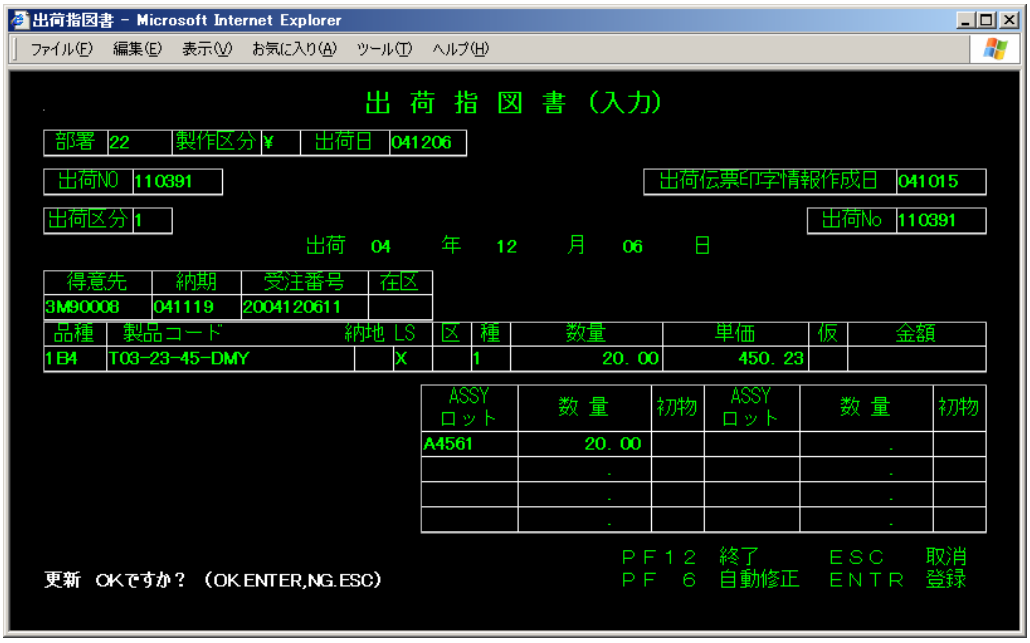

図１　WEB 画面例

４．ＪＯＢ実行状況の把握について

４．１　バッチ処理の実行と稼働状況の把握

汎用機で行っている時間指定の JOB 起動や，他の不定期処理の実行を待って起動する等の運用をスマートにサーバで行うために，ミドルウェア（富士通 SystemWalker Operation Manager）を導入している．汎用機の OS では JCL の稼働状況や端末通信のエラーの状況がコンソールで集中把握できるのは当然のことであるが，Windows ではこの様な機能はない．Windows サーバで集中把握するには，Operation Manager のような，何らかの専用ミドルウェアを導入する必要があるが，そのためにはバッチ処理の全てをミドルウェアの配下で実行する必要がある等，自由度が制限される．

オペレータが手動で実行する処理は，処理途中での件数確認やオペレータの判断待ちを行う必要があるものもあり，全てをミドルウェアに任せるとかえって繁雑となることも多い．また，WEB から起動されるバッチ処理は，Operation Manager で管理できない．これらの処理は，Operation Manager と別管理とし，ログファイルとコマンドプロンプト両方に実行状況を出力する等の工夫をした．Operation Manager 配下の処理もログを残しているので，処理に異常があった際にメールを送信するなどの工夫もあり，ログの記録状況で一応の処理状況の把握ができる．（ログは排他の関係でバッチ毎に別々のファイルである）

４．２　JOB 稼働状況の集中把握ユーティリティの開発

上記の工夫だけでは全体の稼働状況を一覧できないため，稼働するシステムが多くなるに従ってオペレータの負担が大きくなり，実行状況を一覧で把握する必要性が出てきた．

そこで，汎用機のコンソールの様なイメージで JOB の実行状況を一覧で把握できるユーティリティを開発することにした．バッチファイル中で実行してメッセージ情報をサーバに送信するプログラム，UNIX の SYSLOG を参考にしたメッセージを受信し記録するサーバプログラム，その最新状況を表示するモニタープログラムの３種であるが，単純な仕様の割には役立った．バッチファイル中のこのユーティリティ自体の記述にミスがあると記録されないし，全てのバッチファイルに記述しなければならない手間があるが，正確に記述すれば，実行状況が一覧でき，個々のログファイルへのリンクも表示しているので使いやすく，コンソールに慣れているオペレータにも抵抗がない．（**図２**　実行状況の監視画面参照）　テストやトライアルの効率化にもこのシステムは大いに寄与した．

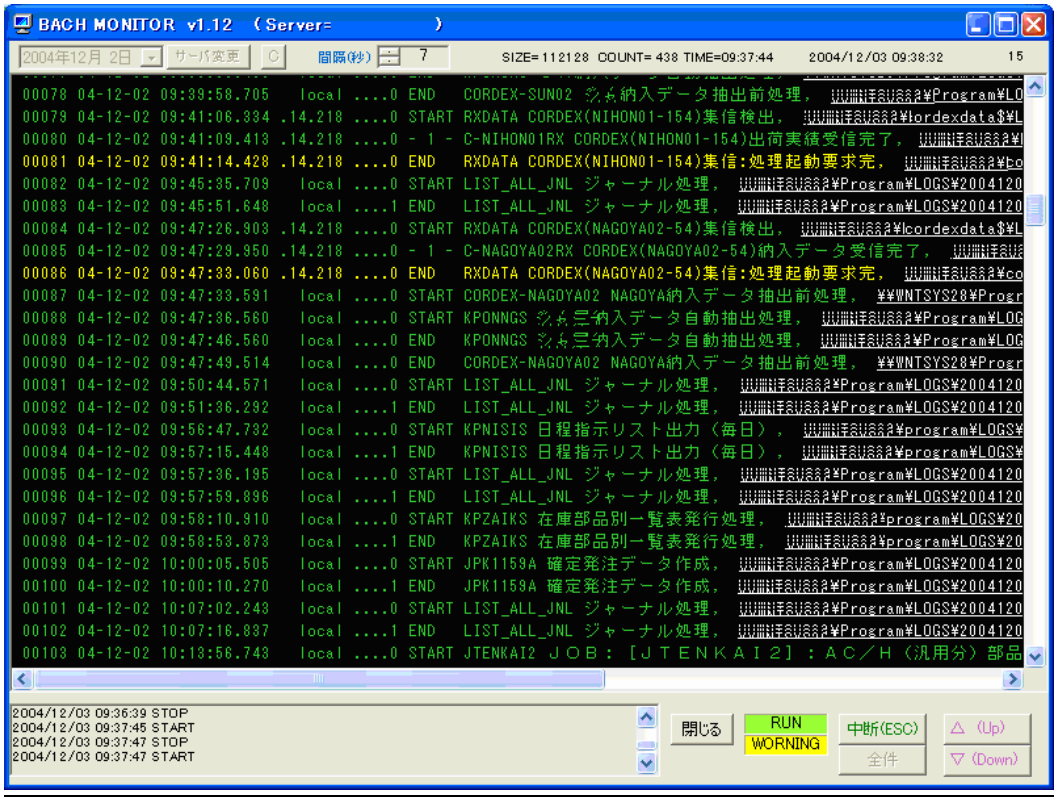

図２　実行状況の監視画面

５．稼働のために役だったシステム

今回の開発に，次の２つの既存システムが大きな役割を果たしたことも，重要なポイントである．

５．１　電子帳票システム

汎用機システムの電子帳票として ListWorks を導入していたため，出力帳票の切り替えは，利用者に新しいサーバにアクセスしてもらうだけでよく，もし，電子帳票システムを導入していなければ，利用部門のプリンターの切り替えやサーバからの印刷設定，レイアウトの調整などに多くの工数がかかっていたことと思われる．テスト，トライアルにおいても効率化に大変役立った．

５．２　汎用機ーサーバ間のファイル転送システム

新しいシステムの立ち上げは，安全を期すため，開発に合わせてサブシステム毎に実施していったが，システムによっては，仕様書のメンテナンスが十分で無いものも多く，退職，異動などにより，以前の担当者が不在であるものも多い．このため，切り替えにあたっては，汎用機の旧システムとの並行稼働期間を長く取り，イレギュラーな運用やデータが発生した際も双方の結果に想定外の差異が無いことを確認していった．

並行稼働を続けるには，双方で同じマスターやインプットデータが必要であるが，利用者に長期間，重複入力させることはできないため，新旧システム間でタイムリーなデータ

の連携（アップロード，ダウンロード）が必要となる．このために汎用機とサーバ間のファイル転送システムを新たに導入するわけにはいかない．

汎用機のデータを利用部門に提供するために１０年前に開発した「自動ファイル転送システム」があったので，これを改良して双方向のファイル転送を行うことにした．処理能力の関係から，最終的には，計５台のこのシステムで連携を行うことになった．

６．開発時の問題点と対応

一般的な開発と同様に実際の開発現場では色々な問題があるものであるが，今回のシステムに特徴的なものも多かった．

汎用機の COBOL で開発してきた担当にとっては，MicroSoft SQL Server の ＲＤＢにアクセスする開発は初めての経験であり，これまで蓄積してきた技術との違いを解消するのに時間を要した．また，システム設計においてはマスタの統合やシステム改善により仕様変更を行ったことでテスト検証に予想以上の時間を要した．

６．１　デッドロックによる処理の異常停止

稼動後 1 ヶ月が過ぎ安定したかに見えたころ，バッチ処理がデッドロックにより異常停止した．至急，状況を確認したが互いの処理はファイル（MicroSoft SQL Server テーブル）の異なるレコードを更新対象とした処理であり，これまでの汎用機のＶＳＡＭファイルではそのような状況でデッドロックが発生することは考えられず，原因究明に時間を要し再発を繰り返した．

ＳＱＬのマニュアルや資料の再調査や色々な試行錯誤の結果，ようやく，原因はインデックスと SQL 文の WHERE 句の不整合からロックを増大させているためであることが分かった．（**図３**　デッドロック発生の一因　参照）

インデックスを付けているからと言って，そのプログラムにとって最適な項目と並びで安易に WHERE 句を指定して良いものではない．インデックスと SQL 文を見直し，ロックのかかるレコード数と時間を最小限に縮めるようチューニングを繰り返すことで，半年以上再発なく安定して稼動している．

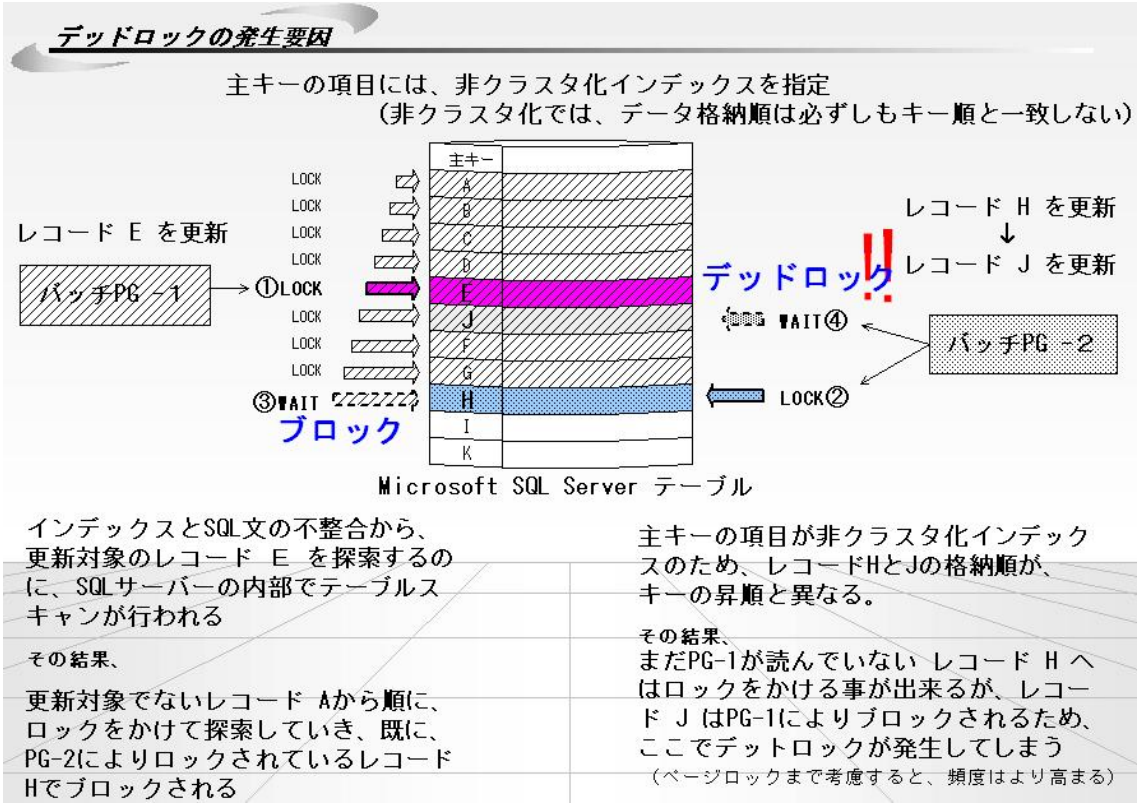

図３　デッドロック発生の一因

６．２　データ移行処理でのレスポンス低下

汎用機の既存データをサーバーへ移行する処理で２０万件中の１万件を超えたところから急に極端にレスポンスが低下し，予測時間を大幅に超えても完了しないため途中でプログラムを停止した．やみくもに，試行錯誤の修正と再処理を繰り返したが，深夜になっても解決できず，このときの移行予定を一週間変更せざるをえなかった．

原因は，数日前にレスポンスの向上とロック軽減を目的に採用したインデックスのクラスタ化にあった．クラスタ化インデックスでは，インデックス領域のページ分割が発生し始めるとデータ領域についても再構築するため，データ量や並びによっては，大きく負荷がかかりレスポンスが低下する．理屈では認識していたが，これほどレスポンスが低下するとは考えていなかった．テーブル設計を見直すことでこの問題は解決したが，今回の問題と，前述のデッドロックをきっかけとしたインデックスの問題は，技術知識を高める上では良い経験となった．

６．３　テスト検証の工数増

旧システムはシステムの追加，変更を繰り返してきた結果，類似したマスタが複数存在し，なかでも技術部品表や単価情報にあたる基盤情報とも言えるマスタまでも 2 重管理されていた．今回の取組みでは，これまでの反省から，どうしてもその統合と正規化を行う必要があった．

マスタの統合は多少の困難は予測していたものの，相矛盾する内容が登録されている項目や単純に併合できない項目などもあり，利用部門に確認しなければ分からないものが多

く，それを調査しながら統合していかなければならない．新システムと旧システムの処理結果の比較テストにおいては，その違いがマスタ統合による仕様通りの結果なのか，バグなのかの判断にも大変に時間がかかった．

７．実績

開発はモデルケースとして一つのサブシステムを開発し，それを元に２年の開発スケジュールとしていたが，半年の遅れとなった．その原因は，主に，今回の取り組みに合わせてシステムの仕様変更も行ったことと，前述の開発時の問題であったと考えている．苦労はあったが，現在は，ハード，ソフトのレスポンス，安定性等，問題なく稼働している．

実際のシステムでは追加システムによる利用部門の改善も織り込みながら，当初目的の社内の開発技術の整理と，ランニングコストの低減（１／３）を達成するメドが立った．

８．今後の課題

最初に開発したサブシステムは汎用機と並行稼働期間を含め，既に一年以上の安定稼働の実績があるといえるが，運用上，改善すべき点も沢山残っている．最も特徴的なものの一つは，実行中にバッチファイルを更新すると，実行途中で中途半端に新しいバッチファイルが採用され，思いがけない動きをすることである．他のシェルを選択していたなら，この様な問題はなかったのかも知れない．この他，汎用機ＯＳでは，当然の機能であった資産管理機能についても，ユーティリティを組み合わせるなど，今後，自分たちで，運用，管理手順を再構築していかなければならない．

ハード面では，前述の様にハードディスクや電源などの一般的な冗長化をしているが，我々の技術力不足により，クラスタ化や，ネットワーク，ＨＵＢの冗長化構成がとれていない点などが反省点である．当初に比べ，これらに対応したハード，ソフトも増え，採用実績も増えていることと思われる．仮想サーバの技術も，サーバシステムの安定運用の重要な要素となってくる可能性がある．今後は，これらの新しい技術を採用した製品を取り入れ，より安定したシステムにステップアップさせたい．

９．おわりに

今回の取り組みにあたり，富士通の営業，SE の皆様には，多くのアドバイスや，貴重な情報を頂きました．特に，技術的ネックの一つでもあった，CORDEX（JCA 手順）等のホスト間の通信システムについては，サーバでの可否や実装方法等について，技術的な問い合わせへの対応や，色々なシステムの紹介を頂いたことにより，ようやく形にすることができました．皆様に深く感謝いたします．