ユーザ主導型のＣ／Ｓシステム運用促進とそのアプローチついて

（株）テイコクインフォメーション・システム

■ 執筆者Ｐｒｏｆｉｌｅ ■

守　屋　　真

1999 年　(株) ﾃｲｺｸｲﾝﾌｫﾒｰｼｮﾝ･ｼｽﾃﾑ入社
2002 年　現在 本社開発グループ
　　グループリーダー

■ 論文要旨 ■

クライアント／サーバ型のＰＣアプリケーション(以下Ｃ／Ｓシステムという)は，汎用機やオフコンのシステムに比べるとユーザ寄りであるといえるが，そのイメージとは裏腹に，クライアント管理やプログラムの配布及びバージョンアップなど，システム管理者の負担は少なくない上に高いスキルと幅広い知識が必要であるといえる．これらを解消するためには，オペレーションの簡素化や自動化が重要であると考え，その仕組みとユーザへのアプローチについて論ずる．

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

１．１　当社概要

当社は，汎用機やオフコンを利用したアパレル産業向け販売・生産管理システムの開発や運用保守を主たる業務として行っている会社である．

１．２　当社におけるＣ／Ｓシステムの現状

近年，Ｃ／Ｓシステムはごくあたりまえのように導入され続けているが，２０年以上も汎用機やオフコンのシステム開発や受託業務を中心に行ってきた当社ＳＥは，オペレーションの違いや管理・運用面での考慮など，様々な問題に直面していた．

当然のことながら，これらの問題は社内からだけではなく，ユーザからも数多く指摘され，早期改善を求める声も少なくはなかった．

また，開発したアプリケーションの性質上，度重なるバージョンアップにも対応する仕組みが必要となっていた．

本稿では，汎用機やオフコンのユーザの観点に基づいた，Ｃ／Ｓシステム向けのインターフェースの仕組みとその開発における考慮点について論ずる．

また，ここでのＣ／Ｓシステムとは，**図１**に示す様な構成であるとする．

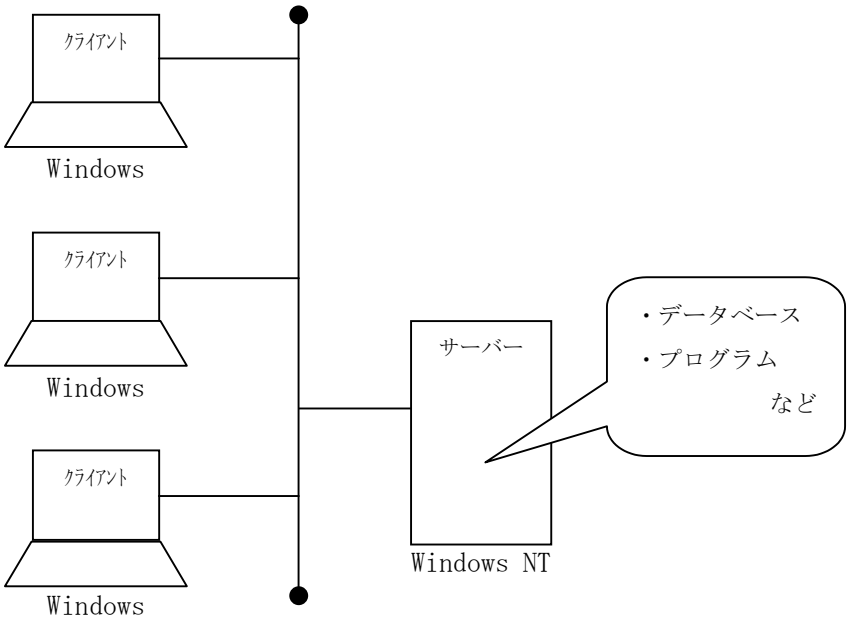

図１　Ｃ／Ｓシステムの構成図

２．汎用機やオフコンとの違いによる問題点

汎用機やオフコンのシステムとＣ／Ｓシステムでは，全くと言って良いほど違う仕組みであるにも関わらず，当社ＳＥのように最近になってＣ／Ｓシステムの開発に乗り出した者をはじめ，多くのユーザからは汎用機やオフコンと同様の仕組みが要求されていた．それらの問題点のうち４つを以下に示す.

・操作性と汎用性

サーバのデータベースは高い能力を持っているが，活用するためにはある程度スキル習得が必要となる．また，管理者はＰＣアプリケーション特有ともいえる度重なるバージョンアップなどに対応する必要があった.

・アプリケーション管理

データベースレベルでの管理や制御はサーバのデータベースが持つ能力で十分行えるが，アプリケーションレベルでのそれは不十分である．このため，汎用機やオフコンのシステムに比べて誰が何をしているかの判断や，プログラムの世代管理が容易ではなくなっている.

・各種モニタの必要性

バッチ処理(ストアドプロシージャ)の実行中の情報が非常に少ない上に経験者でなければ判断が容易ではないといえる．また，その処理結果の確認を行う際も，データを表示するためにはＳＱＬコマンドを習得するか，別途プログラムを作成しておくなどの考慮が必要となる.

・世代管理

汎用機やオフコンではプログラムなどの資産を一元管理しているので，さほど気にはならないが，Ｃ／Ｓシステムでは，クライアントにセットアップしているケースが大半を占めるため，バージョンアップの度にシステム管理者が手動ですべてのＰＣに対して更新作業を施している.

また，その管理方法には標準的なものがあるわけではないので，ドキュメントなどで補うほかないのが現状である.

３．運用や管理における課題と改善策の概要

これらの問題点を解消するために我々が出した結論は，アプリケーションレベルでの標準的な管理ツールを開発することであった．

まず，このツールに求められたのは，いかなるＰＣアプリケーションにも対応ができるという汎用性の高さである．

当社ではすでに多くのＣ／Ｓシステムを開発した実績があるが，作り方や仕様が年々変化しているのですべて手直しすることが容易ではない．そのため，これらの仕組みを既存のプログラムに組み込むという手段は選択できない．

そこで目を付けたのは「メニュー」である．すべてのアプリケーションの玄関ともいえるメニューに，いわゆるコンソール的な能力を持たせることで問題を解決させるというものである．当然，様々なＣ／Ｓシステムに対応させるために，起動するアプリケーションを簡単にカストマイズできるようにするなどの考慮が必要となる．

また，このメニューはより汎用機やオフコンの運用形態に近づけるため，大きく分けて２種類を用意する必要があった．以下にその概要を示す．

・管理者（サーバ）用メニュー

クライアント，データベース，テーブル，プログラムなどの管理やメンテナンスが可能であること．また，各種情報のモニタ機能やデータベースの保守機能などを有する必要がある．

・クライアント用メニュー

起動するアプリケーションのバージョンチェック及び自動更新が可能であること．

また，クライアントから起動されたバッチ処理のモニタ機能を有する必要がある．

４．問題点を解決するための考慮点とその目的

４．１　操作性と標準化

操作性はユーザから最も要求される考慮点の１つとして挙げられるが，当社のユーザの場合は，汎用機やオフコンでのキーボードオペレーションに慣れているため，その点も考慮しつつ，ＰＣアプリケーションの要ともいえるコンボボックスやボタンなどのグラフィカルユーザインターフェースを活用して操作性の向上を図る必要があった．

また，標準的に組み込む機能に関しては，データベースの退避・復元など運用する上で不可欠なものを中心に選別し，よりシンプルでスマートな運用が可能なことを強調しなくてはならない．

また，あらゆる機能は対話式にし，操作に必要なパラメータなどの入力も可能な限り省略させることで，運用の省力化を図るなどの考慮が必要である．

更に，専門用語や略語はユーザに敬遠されがちなので，その点も考慮する必要がある．

４．２　アプリケーション管理

通常のＣ／Ｓシステムでは，設計段階から考慮していなければ，誰がどの様なプログラムを起動しているかを判断することは困難である．これらの情報を取得するには，個々のプログラムの起動時に一元管理されている管理テーブルを更新するような仕組みが考えられる．しかし，数多くある既存のプログラムにそれらの機能を組み込むことは容易ではないので，メニューの中だけで制御する必要があった．

具体的には，メニューから個々のプログラムを呼び出す際に，管理テーブルを更新．その後は呼び出されたプログラムを監視し続け，終了時に管理テーブルを復元すれば常に最新の情報を得られる．この制御をメニューで行えば，既存のプログラムに手を加えることなく監視することが可能となる．

しかし，管理ということとなれば起動状況を判断できるだけでは意味がない．

複数のアプリケーションが混在していると，個々のプログラムが何処に存在し，いつ更新されたものなのか，また用途や種別などの付加情報が必要である．

４．３　各種モニタの必要性

システムを運用・管理していく上では，前述したアプリケーションを始めとし，様々な情報を監視する必要がある．

ところが，多くのサーバにはデータベースレベルのモニタしか存在していないので，情報内容はデータベースに関するものしか取得できない上に，特殊な操作やＳＱＬコマンドなどを習得する必要があった．

このため，操作性や標準化を考慮したモニタが必要になると考えた．モニタの種類は，汎用機やオフコンの運用・管理において利用頻度が高かったものを選別する必要がある．以下にその一部を示す．

・バッチ処理の実行ログ及びメッセージモニタ

用途は自由．処理件数やエラーの追跡に役立てることを目的としたモニタ．

処理日時，起動ユーザ，プロシージャ名及び任意のメッセージを表示する．

・ユーザモニタ

クライアントの稼動状況などを監視するためのモニタ．

ユーザＩＤ，ユーザ名，起動日時，実行中のプログラムなどの情報を表示する．

汎用機やオフコンと同様に利用制限や強制終了などの機能が必要である．

・テーブルモニタ

データベーステーブルの情報や内容を表示するためのモニタ．

データの確認や修正を目的としたものである．

４．４　世代管理

ここでの世代管理とは，クライアントにセットアップされているプログラムの自動更新であるが，この手の世代管理は様々な形態で既に開発・導入されている．

我々が選択した方法は，すべてのアプリケーションの玄関であるメニューからプログラムの起動時に世代情報(更新日時など)をクライアント側とサーバ側とで比較し，異なっていれば自動的にクライアントの指定したフォルダへ，サーバに保管されている最新のプログラムをコピーするというものである．**図２**

前述したアプリケーション管理と同じタイミングで行うことにより，効率良く処理されることが期待できる．

具体的には，メニュー内にアプリケーション管理と世代管理が可能な，個々のプログラムを呼び出すための共通関数を組み込むことで対応する．

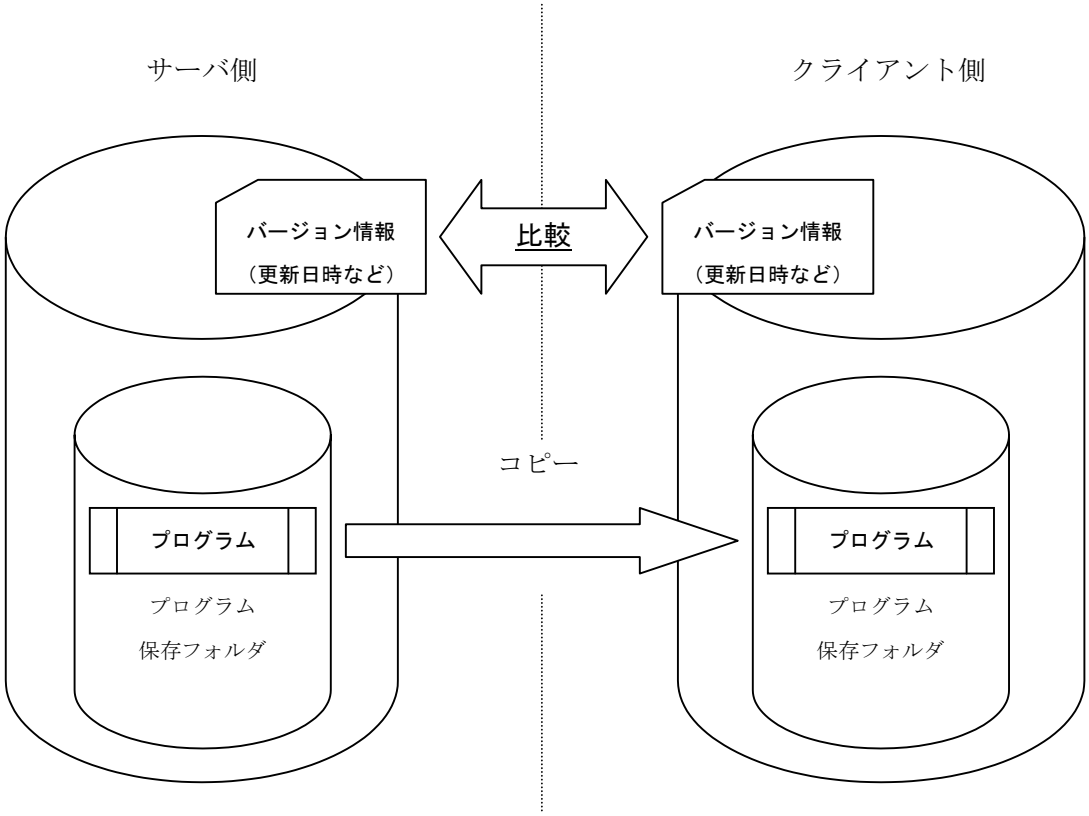

図２　世代管理の概念図

５．管理ツールの開発とその評価

５．１　開発にあたって

開発において特に注意した点は仕様の統一であった．何よりもユーザ主体の運用を目的としていたので，システム管理者だけが理解できれば良いという考えは完全に捨てる必要があったため，画面レイアウトやファンクションキーの設定，配色に至るまで，解りやすく操作性の高いものを目指した．

また，実際に開発に入ると，当初予定していたより多くの機能を盛り込むことになった．以下に追加になった機能を示す．

１．更新頻度の比較的少ないマスタ(区分ファイルなど)を，プログラムの世代管理と同様の手段でクライアントにコピーする機能．
この機能を追加したことにより，クライアントでのレスポンスが向上した．

２．プログラムの世代管理で，サーバ側にあるマスタ(コピー元)プログラムを最新のプログラムに対話式で差し替え，自動的にバージョン情報を更新する機能を追加した．これにより人為的なミスを軽減できた．

５．２　評価

概ね予定通りのツールの開発に成功はしたが，問題はユーザの評価であった．

運用面や管理面でのシステム管理者の負荷は大幅に軽減できるようにはなったのだが，アプリケーションそのものに変化を加えたわけではないので目新しさに欠け，クライアントからは大した評価は得られなかった．

また，確かに既存のアプリケーションに手を加えなくても利用が可能ではあるが，バッチ処理のモニタの様に，全く無修正では利用価値の無い機能があった．

結果的に，アプリケーションそのものの操作性や標準化を改めて見直す必要性を感じることにはなったが，ユーザ主導型のＣ／Ｓシステム運用促進に，ある程度の成果を収めることができた．

更に，この管理ツールでは，過去の資産を継承することも開発する上での重要なポイントになっていたが，新規システムを開発する際にも活用できるものとなった．

しかし，開発には多くの時間をかけられなかったため，現在のものは Micrisoft Access 2000 で作成されている．今後は，Visual Basic 化や Web アプリケーションでの対応を考えていく必要がある．

６．おわりに

システム管理者からすれば，このような省力化を図るツールは非常にありがたいものかもしれないが，ユーザの育成という観点からすると，果たして正しい選択であったであろうか．システムのトラブルに対する迅速な対応や，それを未然に防ぐ方法などを，ユーザといっしょになって考えることが本来の姿ではないだろうか．

これらのことを怠ってしまっては，どれだけ良い仕組みを作っても無意味なものとなるであろう．