勤務票システムの業務用件とシステム要件

Business Requirements and System Requirements of the Work Record Management System

川 口　真 一

要　約　日本ユニシス株式会社では Java と CORBA をベースとして社員の勤怠管理を Web 上でおこなうシステムを開発した．

このシステム開発経験をもとに，Java や CORBA を基幹業務システムに適用することのメリット，適用する際の課題と対応策について紹介する．また，基幹業務システムへの適用を前提とした場合に，現時点で Java を適用すべき場面/適用すべきでない場面について言及する．

Abstract　Nihon Unisys, Ltd. developed a transaction processing system based on Java and CORBA whereby employee attendance records can be managed over the Web in 1998.

This paper introduces the merits of applying Java and CORBA to a mission critical business system, and problems encountered and dealings in system development process. It also discusses cases where Java is suited or not suited for application to mission critical systems at the present time.

1. はじめに

日本ユニシス株式会社（以下，「当社」）では，全社員の勤怠報告を Web アプリケーションでおこなう「勤務状況報告・管理システム（以下，「当システム」）」を 1998 年度に開発した．このシステムは

1） Java＋CORBA[*1]（SYSTEM ν［nju:］[*2]）をベースとしたトランザクションシステムである．
2） Java 2（JDK 1.2[*3]）を採用した業務システムとしては，国内最初のシステムの一つである．
3） オンライン処理としてはサーバ側/クライアント側とも全て Java を採用している．
4） Java Applet により，高度な GUI（Graphical User Interface）を Web アプリケーションで実現している．

などの特徴を持っている．

本稿では，Java と CORBA をベースとした業務システム構築事例として当システムを紹介する．

2. システムの概要

2.1 開発の背景

従来，当社では社員の勤怠報告・管理に紙の「勤務状況報告票（以下，「勤務票」）」を用いて運用してきた．勤務票の運用の概略は次の通りである（図 1）．

1） 毎月月初に，全社員に勤務票を配布する．

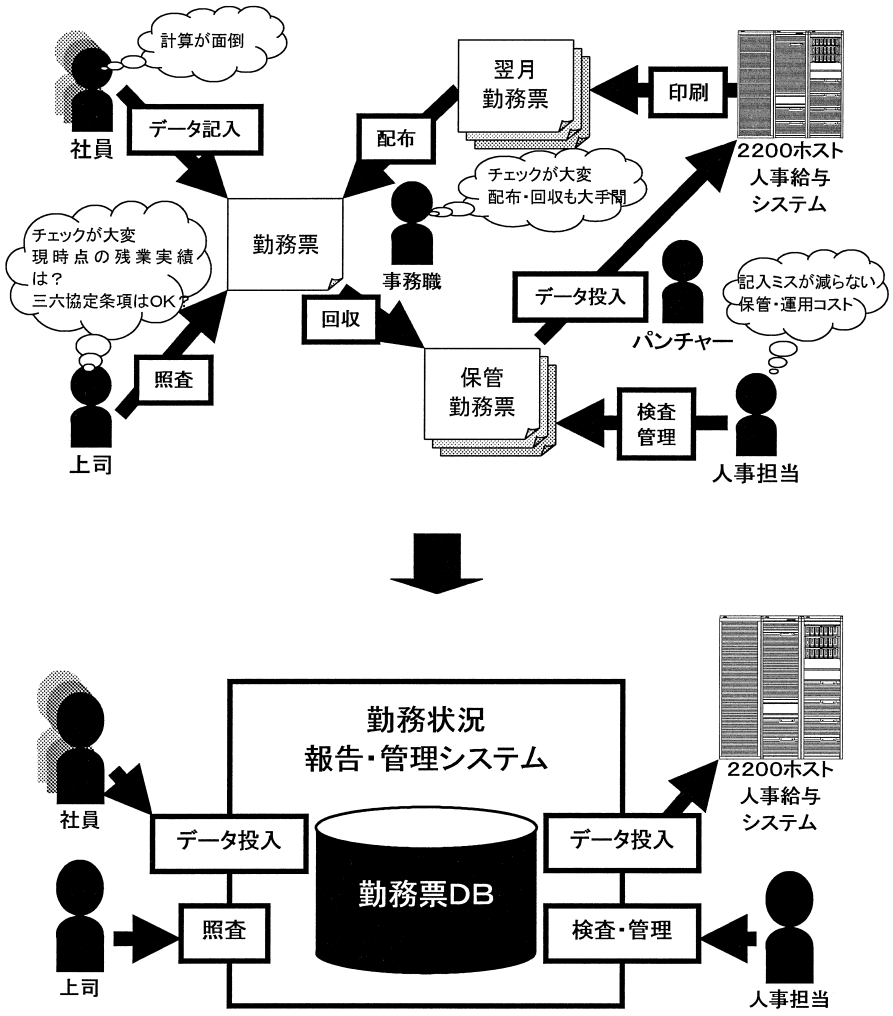


図 1　システム化前後の勤怠管理運用

2)　本人が日々の勤怠状況を勤務票に記入する.
3)　上司が，部下の日々の勤怠状況を照査する.
4)　月末に本人が各項目の月間集計をおこない，勤務票にサマリデータとして記入する.
5)　上司が，部下のサマリデータが記入された一ヶ月分の勤務票を承認する.
6)　承認された勤務票を回収し，サマリデータ部分をホストの人事システムへデータ一括入力する.
7)　回収した勤務票を 4 年間（法定保管期間は 3 年間）保管する.

紙の勤務票による運用では本人が勤務時間数などを計算するため勤務票記入に手間がかかる上，計算ミスや就業規則の誤解，さらにはデータ入力ミスによるエラーデータが毎月約 300 件ほど発生していた．人事部のチェックで記入ミスが見つかった勤務票は現場へ差し戻されて再提出となるので，給与計算などの後続処理への影響も問題である.

また，勤務票は毎日記入して上司の照査を受けるというのが本来の運用である．しかし，本人が毎日提出しないケースの他，本人や上司が外出していて物理的に勤務票の受け渡しが不可能なケースなどがあり，タイムリな勤怠管理がおこなわれにくいというのが実態である．

紙という媒体にまつわる問題もある．毎月一回，約 8000 枚の勤務票を約 400 ヶ所へ配布して回収するという手間がかかる上，回収した勤務票を 4 年分（約 40 万枚）保管しなければならない．保管スペースも必要であるが，毎月約 20 件程度，過去分の勤務票に対して参照や修正の要求があり，その都度人事担当者が 40 万枚の中から目的の 1 枚を探し出さなければならない．データ一括入力や専用用紙代などのコストも見逃せない．

このように，紙の勤務票による勤怠管理運用ではさまざまな問題があり，その解決は人事部にとって長年の懸念事項であった．一方，当社々内ではイントラネット環境整備が進み一人一台の PC（Personal　Computer）環境が整っていた．そこで，各社員の勤怠報告のツールとしてイントラネットを使うこととし，勤怠報告・管理をシステム化するに至った．

2.2 開 発 の 目 的

当システム開発本来の目的は，付帯的なものも加え，次の 3 点とした．

1） 従来の勤務票運用にまつわる諸問題を解消する．
2） 全社員が利用する初のイントラネット上の業務アプリケーションなので，今後追加が予想される同種アプリケーションの基盤を整備する．
3） 最終的に Java と CORBA を採用することになったので，Java＋CORBA による業務システム構築を実証する．

2.3 機能要件とその実現方法

上記の背景・目的から当システムに求められた主な機能要件は次の通りである．

1） クリーンな勤怠データ収集
2） 人事異動対応
3） 勤怠データの保管
4） 運用の簡素化
5） イントラネット上での他業務システムへの横展開

これらの要件を満たすため，次のような方針で当システムの開発にあたった．

1） クリーンな勤怠データ収集

- 高度な GUI の採用
  勤務形態に応じた勤怠メニュー表示，時間帯入力要否や要件入力要否の自動判別など，必要十分なデータ投入をナビゲートすることで，データの誤投入をなくす．
- データ投入時点でのエラーチェック
  就業規則や三六協定などの人事ルールに照らして認められないデータをデータ投入時点でチェックすることにより，クリーンデータのみが DB（Database）に登録されるようにする．
- 自動計算

勤務時間数やサマリデータの集計などをシステムが自動計算することで，計算ミスをなくす．

2） 人事異動対応

- バッチ処理による人事異動対応

  転勤に伴う照査権限関係の変更，昇格に伴う勤務形態の変更などの人事異動対応は，ホストの人事システムと連動して夜間バッチ処理でおこなう．

3） 勤怠データの保管

- 電子媒体による保管

  投入された勤怠データは保管期限満了まで DB 上で保管することにより，過去分勤怠データ検索効率向上，勤務票保管スペース削減，などを図る．

4） 運用の簡素化

- Web アプリケーション化

  WWW ブラウザさえあれば稼働する Web アプリケーションとして構築することで，PC へのプログラム配布を不要とする．

- 運用管理パッケージの導入

  運用管理パッケージ（DSAdmin[*4]）を導入することで，稼働状況監視，障害監視，バッチ処理の自動スケジューリング，などを実現し，運用管理者の負荷軽減を図る．

5） イントラネット上での他業務システムへの横展開

- イントラネット共通機能の独立化

  個人認証など，イントラネット上のアプリケーションで必要な機能を実現する共通部分（この部分を NICE[*5] システムと呼ぶ）と，業務機能を実現するアプリケーション部分（この部分を WISE[*6] システムと呼ぶ）の二段構成とし，今後の業務システム追加は WISE サブシステムの追加のみで対応できるようにする．

2.4 Java の採用

Web ブラウザ上で高度な GUI を実現するために，当システムではクライアント側のプログラムに Java Applet を採用した（図 2）．Java Applet を採用することで，次のような効果が期待できる．

・入力ナビゲート機能や現行勤務票と同様の画面表示など，凝った GUI を実現できる．

・クライアント PC へのプログラム配布の手間を省くことができる．

・クライアント PC の機種が多種多様であることに対応できる．

一方，サーバ側のプログラムは，オンライン処理系は Java Application，バッチ処理系は PL/SQL，Pro*C に分けている．

オンライン処理系に Java Application を採用した理由は次の通りである．

・C 言語に比べ開発生産性向上が期待できる．

・オブジェクト指向プログラミングによる保守性向上が期待できる．

・プラットフォームに依存しないので，将来のサーバ環境変更に柔軟に対応できる．

バッチ処理系にはJavaを採用しなかった．その理由は，SQLによるDB操作だけで実現できる処理ではそのためにわざわざプログラムを作る必要はないことと，「4.3 バッチ処理」で述べるように，大量更新処理が主になるバッチ処理ではJavaのオブジェクト指向プログラミングのメリットが少ないためである．

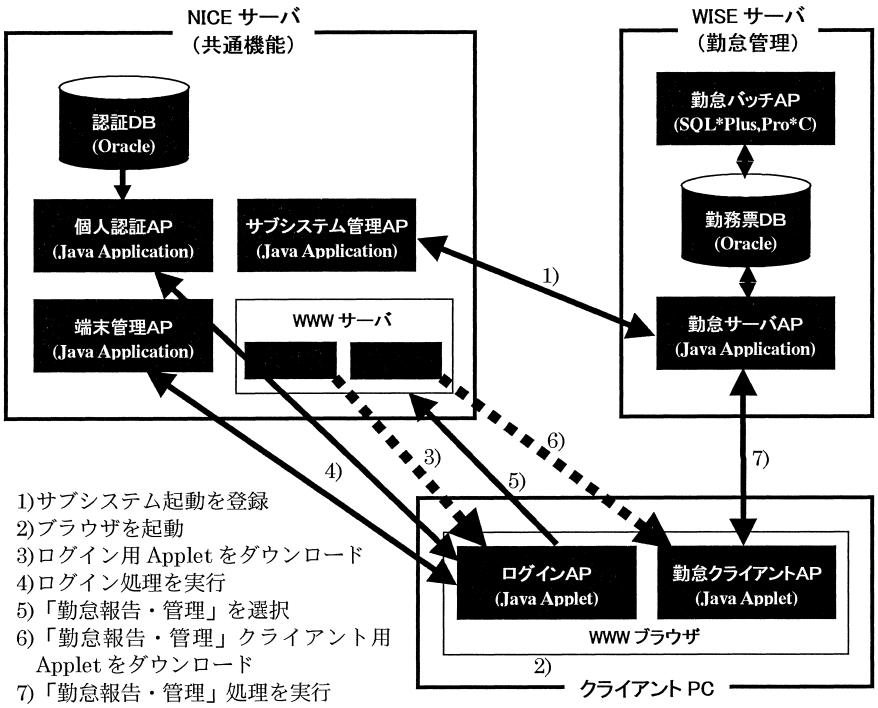


図 2　当システムの構成

2.5　CORBA の採用

当システムでは通信インフラとしてCORBAを採用した．CORBA採用の理由は次の通りである．

・業界標準技術である．
・位置透過性により，将来のネットワーク構成変更に柔軟に対応できる．
・プログラミング言語を問わないので，既存システムとの連携を図りやすい．
・プログラムモジュールの部品化，再利用化が期待でき，将来の機能拡張に有効である．

また，CORBA準拠のORB製品としてSYSTEM　ν［nju:］を採用した（Javaプログラムとの接続部分はJava　IDL）．SYSTEM　ν［nju:］採用の理由は次の通りである．

・自社製品である．
・トランザクション制御，流量制御などの機能が用意されていて，Java　IDLに不足している機能を補完できる．

2.6 Java＋CORBA 採用のメリット

Java が持つプラットフォーム独立性と，CORBA が持つネットワーク位置透過性を組み合わせることにより，当システムでは次のようなメリットを得ることができた．

2.6.1 開発フェーズ

複数のサブシステムの開発・テストを同時に進める場合，従来であれば開発・テスト環境の確保のため開発用ハードウェア（特にサーバ機）を開発者同士で取り合う場面がみられた．

Java＋CORBA であれば，例えばクライアント AP（Application Program），サーバ AP とも Windows 95 の PC 上で開発・テストした後，サーバ AP を UNIX サーバへ移してそのまま本番稼働させる，ということが可能となる．サーバ AP の開発・テストの大部分を PC 上でおこなうことができるので，限られたハードウェア資源の中で効率的な開発作業を進めることができる．

2.6.2 運用フェーズ

共通サーバとしての NICE サーバにネーミングサービス的な機能を持たせ，NICE サーバに問い合わせれば，その時点でサービスを提供中のサーバの位置情報を入手できるようにした．サーバの位置情報だけ渡せばあとは CORBA がセッションをつないでくれるので，クライアント側ではサーバのネットワーク上の位置を意識しなくて済む．NICE サーバのネットワーク上の位置さえ固定しておけば，他のサーバの位置は自由に変更できることから，

・障害時に代替サーバへの切り替えが容易．

・将来サーバ構成を変更する場合，プログラム変更の必要がない．

などのメリットがある．事実，当システムでも試行本番から全社展開に向け AP・DB サーバ×1→AP サーバ×2＋DB サーバ×1 へサーバ構成を変更したが，プログラムの変更なく対応できている．

また，Java プログラムは稼働プラットフォームを問わないので，例えば UNIX サーバで稼働していた AP を NT サーバでバックアップする，といったことも可能であり，ハードウェア調達やシステム運用面での融通がきくようになる．

3. 課 題 と 対 応

当システム開発で直面した課題として処理効率とオブジェクトの永続化の問題を取り上げ，当システムでとった対応について記す．

3.1 処 理 効 率

3.1.1 Applet のダウンロード時間

当システムで使用する Java Applet は NICE クライアント用が約 200 KB，WISE クライアント用が約 1.5 MB（JAR ファイル圧縮時の容量）である．当システムを使用する都度，各社員がこの Java Applet をダウンロードするということは，

1） Applet のダウンロードに時間がかかる（特に回線速度が低い場合は顕著）．

2） ネットワーク全体のトラフィックが増加する．

という問題を引き起こすことになる．

そこで，プログラム事前配布が要らないという Java Applet の利点は失われること

になるが，Java Applet（JAR ファイル）を事前にクライアント PC へダウンロードしておくことで，実行時のダウンロード時間短縮とネットワーク・トラフィック量増加の抑制を図ることとした（詳しくは本誌山本史朗の別稿を参照）．

3.1.2 クライアント PC のスペック

当システムを使用する場合，クライアント PC 側でブラウザの起動，JavaVM（Virtual Machine）の起動，Java Applet のバイトコードから実行コードへの翻訳処理が必要となる．これらの処理はいずれも CPU 負荷が高い．また，当システム起動後，初めて開く画面はクラスロードと，実行コードへの翻訳処理が入る．したがって，ある程度の CPU スペックを持つ PC でないと，処理が遅くなる．

また，ダウンロードしてきた Applet や翻訳後の実行コード，サーバとの間でやりとりしたオブジェクトデータなど，すべてメモリ上で保持される．ブラウザや JavaVM そのものもメモリをかなり消費するので，実装メモリが少ないとスワップ処理が入り，その分処理が遅くなる（図 3）．

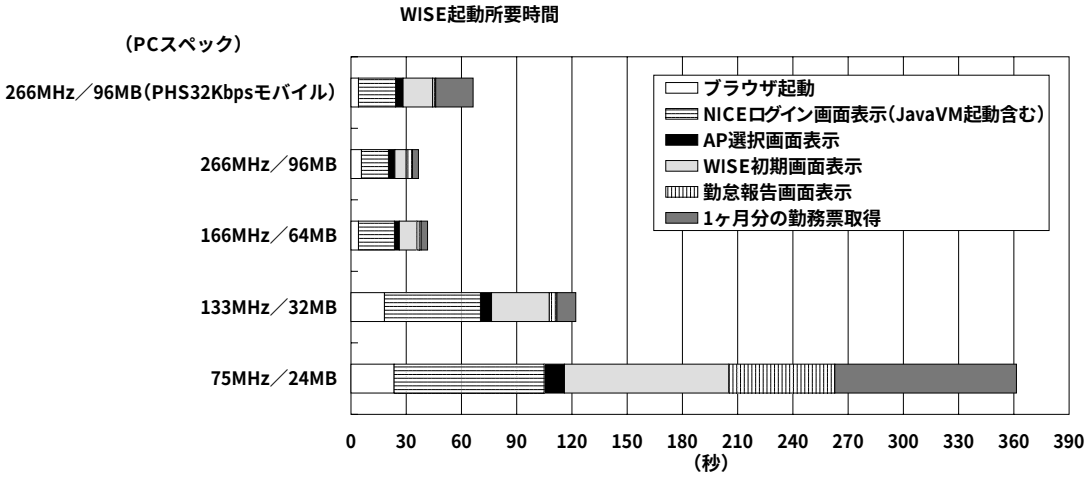


図 3　PC スペック別の当システム起動所要時間

当システムの場合，クライアント PC の最低スペックを CPU : Pentium 166 MHz，実装メモリ：64 MB とし，この最低スペックに満たない PC は高スペック PC への置き換えを図ることとなった．

現在市販されている PC のスペックは上記スペックに比べてはるかに高いので，これから新たに PC を購入する場合は問題とならないが，既存の PC 上で Java Applet を用いたシステムを稼働させる場合は PC のスペックに注意が必要である．PC の置き換えが必要な場合，台数が多ければ多大な投資が必要となる．PC 置き換え費用の額によっては Java Applet の採用を見送る判断も必要となるだろう．

3.1.3 Java VM の処理負荷

JavaVM 上で稼働する Java プログラムは，Java VM プロセスのスレッドとして処理される．そのため，Java プログラムが消費した CPU 時間は Java VM プロセスの

消費 CPU 時間として積算される．

1 ヶ月分の勤務票取得処理における Java VM プロセスの CPU 時間を測定したところ，1 トランザクションあたり平均で 379 m 秒（A）であった（US 450：UltraSPARC II 300 MHz×2）．この時，Java Application 内部で開始・終了時点のタイムスタンプを採取して処理時間を測ったところ，約 210〜270 m 秒(B)であった．この値は ELAPS 値なので，Application 処理に要した CPU 時間はこれより少ないはずである．

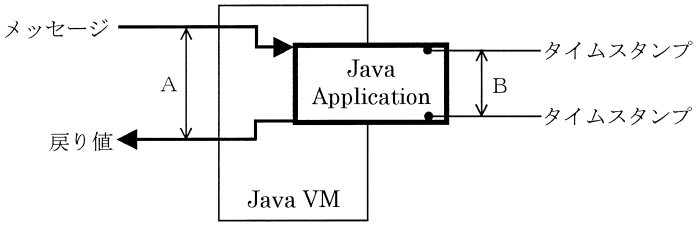
A と B の時間差（約 110〜170 m 秒）は Java VM 自身がバックグラウンドで消費している CPU 時間ということになる（図 4）．Java VM がバックグラウンドでおこなう処理は，

1） ガーベージコレクション処理

2） IIOP 通信制御処理

などであると推定される．

処理内容によって左右されるが，Java プログラムそのものの処理に比べて割と大きな比重で Java VM 自身が CPU 時間を消費するようである．サーバの機種選定や効率改善などの際には，Java VM 自身の CPU 消費を念頭におく必要がある．



	測定時間の内容	当システムでの測定結果
A	Java VM プロセスが 1 トランザクション処理で消費した平均 CPU 時間	379m秒
B	Java Application 処理時間（ELAPS）	約 210〜270m秒
A－B	Java VM がバックグラウンドで消費した CPU 時間（推定）	約 110〜170m秒

図 4　Java VM のバックグラウンド負荷

3.2　オブジェクトの永続化

3.2.1　データストア

業務システムでは，永続性を必要とするデータが必ず存在する．オブジェクトのインスタンスはメモリ上にしか存在しないので，オブジェクトを永続化するためには HDD（Hard Disc Drive）などの外部記憶にデータを保存する必要がある．

オブジェクト（インスタンス）を外部記憶に保存する手段としては

1） オブジェクトをシリアライズしてファイルに書き出す．

2） RDB（Relational DataBase）へ格納する．

3） OODB（Object Oriented DataBase）へ格納する．

などの方法がある．

シリアライズしてファイルに書き出したデータは再利用性に欠け，業務システムの

データストアとしては不適切である．OODB は新しい技術であり，使用実績，大量データ（当システムでは 30 日×48 ヶ月×8000 人＝約 1150 万件程度のテーブルが存在する）を登録した際のパフォーマンス，バッチ大量更新のパフォーマンス，等に疑念があり採用には十分な検証が必要である．従って，当システムではデータストアに RDB（Oracle 7.3.4）を採用した．RDB を採用することの利点としては，

1） 使用実績や安定性の面で信頼できる技術である．
2） SQL を使ったバッチ大量更新が可能である．
3） 障害対応などの際，一部データを標準ユーティリティで強制修正することができる．

などがあげられる．

採用した Oracle 7.3.4 の場合，継承や構造体型などの概念がないため，オブジェクトモデルのクラス構造をそのまま DB スキーマ構造に反映させる訳にはいかない．いわゆるインピーダンス・ミスマッチに対する考慮が必要となる．SQL 3 に対応した Oracle 8 であれば，このあたりの問題は多少改善されるだろう．

3.2.2 オブジェクトモデルと RDB スキーマのマッピング

オブジェクトモデルを RDB スキーマにマッピングする場合，オブジェクトモデルのクラスを RDB のテーブルに，オブジェクト（インスタンス）を RDB のレコードに対応させるのが通例である．

クラスをテーブルに 1 対 1 でマッピングすると，集合オブジェクト（複数のオブジェクトで構成されるオブジェクト）のクラスでは，正規化されたテーブル構造になる．データをオブジェクトとして扱うオンライン系処理（Java　Application）では問題ないが，データを RDB として扱うバッチ系処理では処理効率が悪いケースがある（図 5）．

例えば勤務票の場合，1 ヶ月分の日々の勤怠データを集計したサマリデータというものがある．サマリデータそのもの（出張回数，代休時間数，等のデータの集合体）も一つのオブジェクトだが，出張回数や代休時間数など個々のサマリデータ項目もそれぞれ一つのオブジェクトとみることもできる．しかもそれらは，出張回数などのように「回数」を属性に持つもの，代休時間数などのように「時間数」を持つもの，等のクラスに分けることができる（実際オブジェクトモデルではそのようなサブクラスに分割した）．これらのクラスをそれぞれ別々のテーブルとした場合，1 勤務票のサマリデータは複数テーブルの複数レコードで構成されることになる．ホスト上の人事システムへサマリデータを転送するバッチ処理の場合，複数テーブルの複数レコードを寄せ集めて 1 レコードを構成するという RDB 的には非効率な処理をおこなわなければならない．

バッチ処理効率を考えるのであれば，RDB 設計の定石通り「正規化くずし」をおこなうべきである．オブジェクトモデルと RDB スキーマとの差異は，RDB からデータを読込んでオブジェクト（インスタンス）の属性値にセットする，またはその逆をおこなうデータアクセスモジュール（オブジェクト）を用意して，ここで両モデルの差異を吸収させればよい．例えばサマリデータの場合，RDB 上は 1 勤務票のサマリデータを 1 レコードとして格納しておき，データアクセスモジュール（オブジェクト）

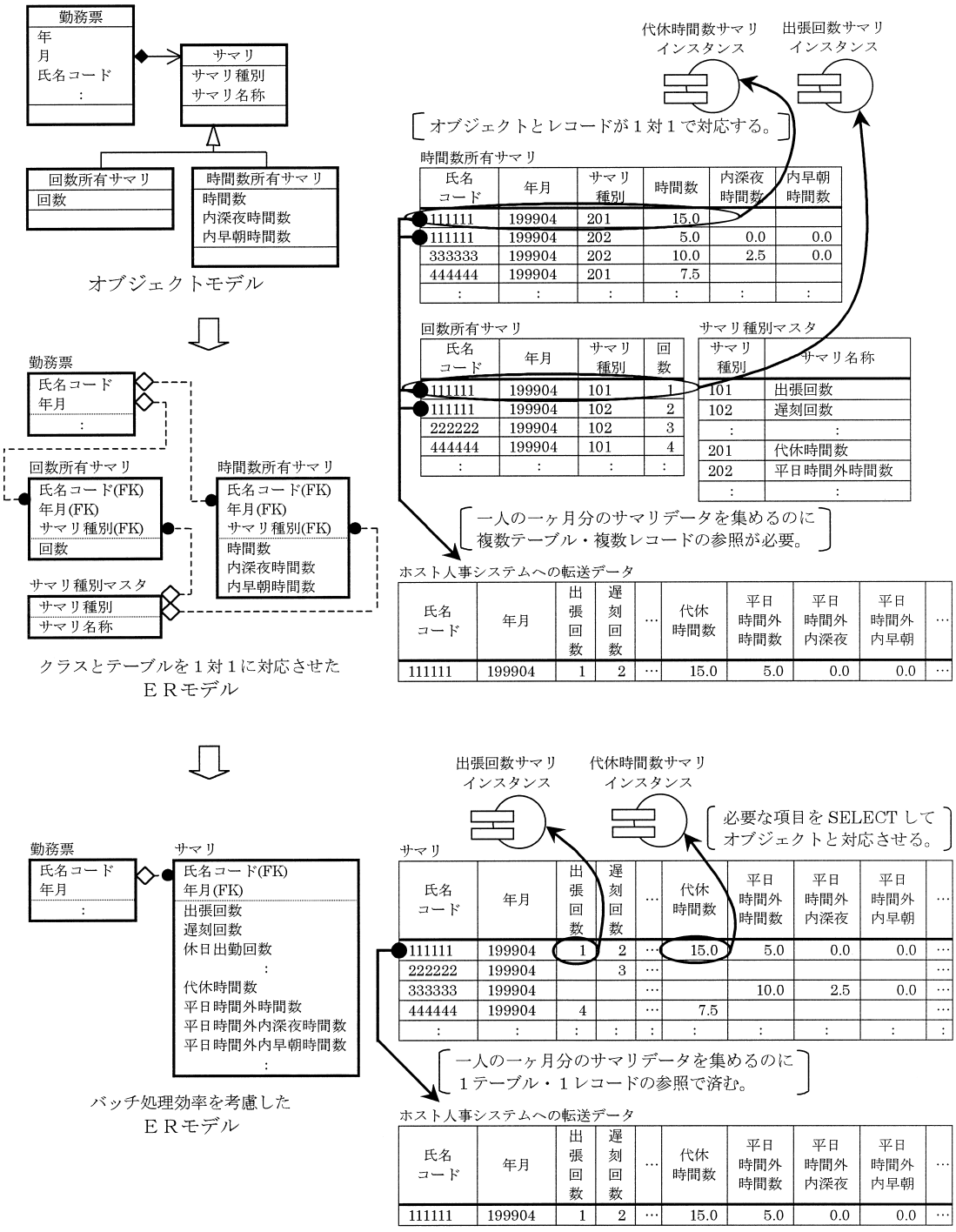


図 5　オブジェクトモデルと RDB スキーマのマッピング

で該当するオブジェクトのデータ項目（例えば「出張回数」）を SELECT して，これをオブジェクトの形に置き換えてクライアントへ返せばよいのである．そうすれば，RDB スキーマはバッチ処理やバックアップなどの都合に合わせた処理効率のよい設

計にすることができる．

4. 基幹業務システムへの Java の適用

Java を基幹業務システムへ適用できるか，という点については，適用できると言ってよいだろう．ただし，適用すべきではない場面がいくつかあるので，システム全体を全て Java で構築するということは，現時点では無理な場合があると考える．

当システム開発の経験から，Java を適用できる場面/適用すべきではない場面，または適用する際に注意すべき点について，以下にまとめてみた．

4.1 Write Once, Run Anywhere

Java 実行環境（JRE：Java Runtime Environment）のバージョン（1.0/1.1/1.2）が混在し，かつバージョン間で一部非互換がある現状では，Java がどのプラットフォーム上でも稼働するとは言い切れない．Sun Microsystems 社が提唱する「Write Once, Run Anywhere」は JRE のバージョンが統一されていること，という条件付きでの話である．当システムでは，ブラウザ（Netscape Communicator）に Java Plug-in（JRE 1.2.1）を別途導入して実行環境の統一を図っている．

また，当システムでは，「Applet はダウンロード元のサーバとしか通信できない」というセキュリティを「当社内のサーバであれば通信できる」というレベルに緩和するため，各クライアント PC の policy ファイル[*7] の設定を変更している．Java のセキュリティレベルを変更する必要があるシステムの場合は，JRE のバージョンに加えて，各クライアント PC の policy ファイルの内容統一も必要となる．

従って，比較的実行環境を統一しやすいサーバ側の処理プログラムには Java を適用しても問題ない．イントラネット環境下のクライアント PC 側の処理プログラムについても適用の余地はある．

逆に不特定多数のクライアント PC を相手にするインターネット環境下のクライアント PC 側の処理プログラムへ Java（Java Applet）を適用することは，現時点では見送るべきである．

4.2 印 書 機 能

Java の印書機能は低レベルな API（Application Program Interface）しか用意されておらず，業務帳表を印書するには不十分である．

Java 対応の印書パッケージもいくつか出回るようになってきているが，業務ニーズに応じて選択できるほど出揃ってはいない．満足できる印書パッケージが見つからない場合は，印書機能部分だけを Java 以外で実装することも視野に入れるべきである．

4.3 バ ッ チ 処 理

基幹業務システムにはデータの大量更新などのバッチ処理が付き物である．大量のデータをひとまとめにして扱うバッチ処理をオブジェクト指向で設計することは困難である．永続化するオブジェクトであれば，データストアに RDB を採用し，SQL によって検索・大量更新をおこなわせるのが現実的である．

データの大量更新を伴うようなバッチ処理プログラムを Java で書くことも可能であるが，その場合は JDBC 経由で（従来の Pro*C プログラムと同様に）SQL を投げ

るだけのプログラムと割り切り，オブジェクト指向の世界と切り離して考える必要がある．Java はオブジェクト指向と親和性が高いからといって，Java を使う場合は必ずオブジェクト指向設計を適用しなければならない，ということはない．

4.4 Java Applet

Java Applet の魅力は高度な GUI プログラムを，クライアントに配布することなしに使える点にある．

反面，Java Applet が動くまでには，ブラウザの起動，JavaVM の起動，Applet のダウンロード，バイトコードの翻訳等の処理が必要であり，起動してからアプリケーションが使えるようになるまでに時間がかかるという問題がある．この問題は，PC やネットワークが高速化することによって将来的には問題にならなくなるだろうが，現時点では無視できない．また，Java Applet（と，その実行環境）はメモリ消費量が多いという問題もある．従って，Java Applet の採否については，アプリケーションの特徴とクライアント PC 環境に応じて判断すべきである．

一度起動すれば 1 日中稼働し続けるようなアプリケーション（例えば，営業店窓口システム）では，朝一回の起動処理が多少（1〜2 分）長くなる程度なので，問題ない．PC の実装メモリ量を十分確保することに注意すれば，Java Applet の長所を生かしたシステムを構築できるだろう．

逆に，使用する都度起動するようなアプリケーション（例えば，当システム）では注意が必要だ．起動時間がネックとなってユーザに受け入れられないシステムになる可能性がある．当システムでは，「3.1.1 Applet のダウンロード時間」で述べたように，JAR ファイルを事前にクライアント PC に配布しておくという方法で起動時間短縮を図っている．

なお，Java Applet による高度な GUI 機能を必ずしも必要としないアプリケーションの場合は，クライアント側の GUI に HTML を採用するという選択肢もある．HTML を採用した場合，起動時間はブラウザの起動＋HTML のダウンロードだけで済むことになる．

5. おわりに

今回，Java と CORBA をベースにした業務システムを初めて構築した．Java＋CORBA をベースにすれば稼働プラットフォームを問わない，拡張性・柔軟性があるシステムを構築することができると言われているが，実際その通りのシステムを構築できたと考える．

ただし，Java，CORBA とも新しい技術であることから，基幹システムとしてまともに動く環境を構築するのに苦労したのも事実である．発展途上のソフトウェアに付き物の初期不良とも言うべき Bug にも何度か遭遇し，その都度回避策を講じなければならなかった．

だが，今後 Java や CORBA が成熟するに従って，今回我々が経験したような諸問題もいずれ解決されるであろう．この先 Java や CORBA が情報処理技術のスタンダードとして大きな位置を占めることは間違いない．

今後増えるであろう Java や CORBA を採用したシステム開発に，本稿が参考とな

れば幸いである．

＊1　CORBA：Common Object Request Broker Architecture. OMG（Object Management Group）が策定するORB（Object Request Broker）の標準仕様.
＊2　SYSTEM ν［nju：］：日本ユニシス(株)が提供する，CORBA準拠のORB製品.
＊3　JDK：Java Developers Kit. Sun Microsystems社が提供している Java 開発環境. 通常, JDKのバージョンがJavaのバージョンとして扱われる．JDK 1.2は「Java 2」の通称で1998年12月に正式リリースされた.
＊4　DSAdmin：日本ユニシス(株)が提供する運用管理パッケージ.
＊5　NICE：Nihon unisys Intranet Common Engineの略．当社社内イントラネット上で稼働するアプリケーションに対して，個人認証，端末監視，サブシステム（アプリケーション）稼働状況監視など，全アプリケーション共通で使用する機能を提供する.
＊6　WISE：Web Intranet Service & control for Employeeの略．NICE基盤上で稼働する一アプリケーション．特定の業務担当社員に限らず，全社員が共通で使用する業務を想定しており，当システムが第一弾となる.
＊7　policyファイル：Javaのセキュリティ方針を設定するファイル．JREのlib/security/ディレクトリ配下にあるjava.policy.

参考文献　［1］　中山陽太郎,“ SQL・IDLマッピング ”, 技報, Vol. 17, No. 3, 1997.
［2］　岩田裕道, 他,“ メール討論―オブジェクト・モデリングと実装 ”, 技報, Vol. 18, No. 4, 1999.

執筆者紹介　川　口　真　一（Shinichi Kawaguchi）
1964年生．1986年中央大学法学部法律学科卒業．同年日本ユニバック(株)(現日本ユニシス(株)）入社．汎用機のシステム移行，汎用機ユーザのSEサービス等を経て1998年よりJavaによるシステム開発を担当．現在，生産技術部技術一室所属.