勤務票システムのアーキテクチャ

Architecture of the Work Record Management System

佐々木 勝信

要 約 日本ユニシス株式会社は，1998 年に Java, CORBA を用いて全社員の勤務票を管理するシステムの開発を行った．当システムの開発には，レスポンスの確保と社内イントラネット基盤としての普遍性の確立という二つの課題が課せられた．

業務システムの開発には，アーキテクチャが必要になる．アーキテクチャを構築する事によって，さまざまな課題・要件の解決を図る．

本稿は，Java, CORBA を要素技術に，勤務票システムのアーキテクチャをどう構築したかについて報告する．

Abstract In 1998, Nihon Unisys, Ltd. developed a system for managing attendance records for the entire employees in the Java and CORBA environment. Two issues imposed upon development of this system were the achievement of the adequate response time and designing for general use as the intranet infrastructure.

Development of an application system requires for the architecture. By building a new architecture, we have attempted to resolve the various issues and requirements.

This paper describes how we built the architecture for the attendance records system with Java and CORBA as component technologies.

1. はじめに

日本ユニシス株式会社（以下，当社）は 1998 年度に，全社員の勤務報告を Web アプリケーションで行う勤務票システム（以下，当システム）を開発した．当システムは，当社の社内システムでは初の Java, CORBA[*1] を用いた大規模トランザクションシステムである．

1998 年度に本格開発を開始する前に，Java, CORBA を使った業務システムの実証としてプロトタイピングを行った．プロトタイプで得られたシステムは，諸々の事情により実運用には不十分な代物であった．その理由は，Java, CORBA に対して十分なノウハウをもっていなかったためである．プロトタイプの経験を元に，当システムの構築を着手することになった．

本稿では，プロトタイプにより見つけた課題，また社内イントラネット・システムとして課せられた技術的要件を紹介し，この要件を解決すべく構築したアーキテクチャについて解説する．

2. システム構築にあたっての課題

2.1 プロトタイプにより得られた課題

2.1.1 プロトタイプの経緯

プロトタイピングは，Java, CORBA を使った業務システムの実証を目的に行われ

た．プロトタイプで使用したプロダクトセットは諸般の事情により，以下に示すセットで構築することとなった．

- JDK 1.2 Beta 4
- JavaIDL Compiler EA
- SYSTEM ν［nju：］Ver 3.1.1

当時，Java は CORBA-IDL[*2] をサポートしたばかりで，まだ IDL コンパイラも JDK も β 版しかリリースされていなかった．SYSTEM ν［nju：］は CORBA-ORB[*3] として使用した．

プロトタイプは Java，CORBA という組み合わせを効果的に利用しようと考え，すべての工程をオブジェクト指向方法論に則って推進した．オブジェクト指向は，意味的に独立したオブジェクト同士が情報をやり取りして処理を行う，メッセージ・パッシングと呼ばれる駆動方式をとっている．C＋＋，Smalltalk などのオブジェクト指向言語だけでは，メッセージはメモリのみを伝達媒体としていたが，CORBA に代表される分散オブジェクト技術によりネットワークもその伝達媒体となった．従来，物理的制約により論理設計から物理設計において設計の乖離が指摘されるオブジェクト指向であるが，分散オブジェクト技術により物理的制約からの解放を得たとも言える．

プロトタイプが行う実証は，分散オブジェクト技術によるオブジェクト指向方法論の可能性も含まれている．

2.1.2 プロトタイプの結果

CORBA の採用により，オブジェクト間を交信するメッセージはネットワークを越えて伝達することが可能になった．また，オブジェクト指向方法論と Java，CORBA との親和性の高さも相まって，論理設計からの乖離を見ることなくプロトタイプは実装を行うことができた．

当初の目的を果たしたと思われるプロトタイプだが，落とし穴も見つかった．それは，通信コスト増加によるレスポンスの低下であった．

先にも述べたようにオブジェクト指向はメッセージ・パッシングを主体とした駆動方式をとる．そのため，オブジェクト指向に準じるほどメッセージ・パッシングの度合いは増加する．論理設計では，メッセージ・パッシングにかかるコストはゼロとして考えられるが，実装を行った時点で一定のコストが発生する．

メッセージ・パッシングにかかるコストは，その伝達媒体と転送量によって決定される．分散オブジェクト技術により伝達媒体となったネットワークは，伝達媒体としては遅い部類に属する．たとえば，CORBA のリモートメソッド呼び出しは，1 回あたり約 20〜30［msec］の通信コストを消費する．

プロトタイプでは，勤務票オブジェクトをリモートオブジェクトとして実装していたため，勤務票に対する操作は，属性の参照であっても，すべてリモートメソッド呼び出しとして扱われた．

1 ヵ月の勤務票は約 500 の属性から構成される．これを先の通信コストを元に試算を行うと，1 ヵ月の勤務票を表示するのに要する通信コストは，

（属性数）×（属性 1 つあたりの通信コスト）＝（消費する総通信コスト）

（500）×（20〜30）　12500［msec］

となる．この勤務票オブジェクトは，使用頻度も高いため，多くの通信コストを消費した．

プロトタイプで得られた課題は，通信コストの軽減．この課題の解決がシステム構築の命題となった．

この命題の背景には，分散オブジェクト技術の適用に関するノウハウの確立がある．分散オブジェクト技術は，オブジェクト指向方法論にとって有効な実装技術ではあるが，ある方向性をもって実装に望まなければプロトタイプのような結果を招く．そのため，技術利用の方向付け――ノウハウまたはコツ的な考え方が必要となる．通信コストの軽減は，その構成要素の一つになる．

2.2 イントラネット基盤としての要件

川口[1]で述べられたように，当システムは単なる業務システムとしてではなく，社内イントラネットの基盤としての役割も担っていた．

これは，個人認証などのイントラネット上で共通に扱われる部分（イントラネット・エンジンと呼ぶ）と，業務機能を実現する部分（アプリケーション・サービスと呼ぶ）とに分かれている．

2.2.1 イントラネット・エンジンとして求められた要件

イントラネット・エンジン部分は，イントラネット上の業務アプリケーションを統括するのに必要な共通機能を提供する必要がある．主な機能要件には，

- 業務アプリケーション管理
  登録，解除，業務サービス提供
- クライアント管理
  ユーザ認証，端末監視

がある．

また，クライアントは，イントラネット用 Web サーバのトップページのみを知っていれば，イントラネット・エンジンが提供する業務アプリケーション・サービスが受けられるようにする必要もあった．

2.2.2 アプリケーション・サービスとして求められた要件

アプリケーション・サービスは，イントラネット・エンジンに登録可能で，かつ一般的な業務アプリケーションに適用できるアプリケーション構造を提供する必要があった．

当システムの構築は，勤務票管理というアプリケーション・サービスの一事例とみなすことができる．

また，アプリケーション・サービスの要件には，リレーショナル・データベースの利用を包含している．一般的な業務アプリケーションと呼ばれるものにはデータベース，それもリレーショナル・データベースの利用を抜きにして考えることができない．

業務データは，なんらかの方法で永続化する必要がある．永続先の対象として挙げられるのがデータベースであり，とりわけリレーショナル・データベースが最有力候補となるのが常である．リレーショナル・データベースは，そのアプローチの違いからオブジェクト指向方法論にとってアプリケーション・モデルの乖離を引き起こす難

問として立ちふさがる（この問題については川口[1]で深く論じられている）．データベースには，オブジェクト指向データベースという選択肢も存在するが，技術的な熟練度からリレーショナル・データベースのサポートは，業務アプリケーション構築における必須項目と捉えるべきであろう．

3. システム・アーキテクチャ概説

3.1 アーキテクチャ

前述した要件，課題を受けて構築したアーキテクチャを図1に示す．

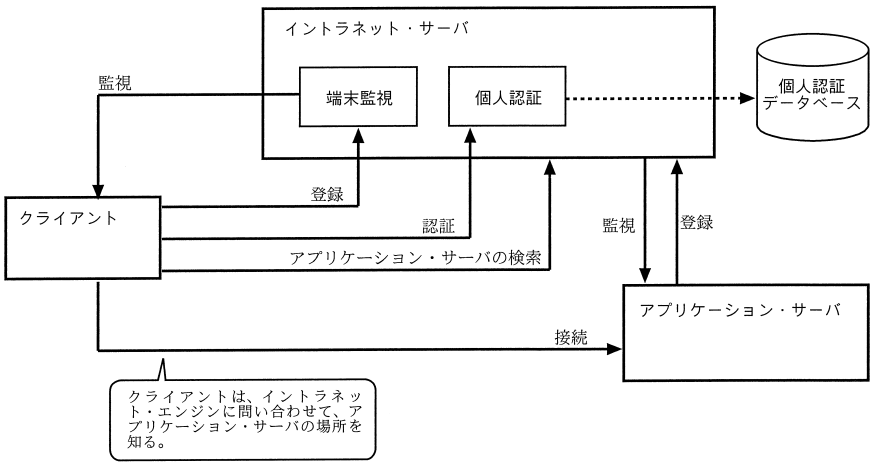

図 1　アーキテクチャ概念図

イントラネット・エンジン・アーキテクチャは，次のモジュールで構成している．

- イントラネット・サーバ
- 管理モジュール
- アプリケーション・サーバ

イントラネット・サーバは，アプリケーション・サーバ，端末監視などの管理モジュールを集中管理しシステム全体の中枢を担う．主なサービスは，アプリケーション・サーバの管理で，ネーム・サーバとして振舞う．イントラネット・サーバの位置さえ固定しておけば，その他のモジュールはネットワーク上のどこで稼働してもよい．

管理モジュールは，ユーザ認証，端末監視，管理コンソールから構成される．

アプリケーション・サーバは，イントラネットに登録可能なアプリケーションの仕様を規定しており，この部分をアプリケーション・サービス・アーキテクチャと呼ぶ．

アプリケーション・サービス・アーキテクチャは，以下の三つの要素からなる．

- アプリケーション・サーバ
- コントロール
- ホーム

アプリケーション・サーバは，業務アプリケーションの外皮となる要素で，コントロール，ホームはアプリケーション・サーバの構成要素となる．コントロール，ホー

ムは，それぞれに固有の名前を持ち，図 2 のようにアプリケーション・サーバの管理下におかれる．

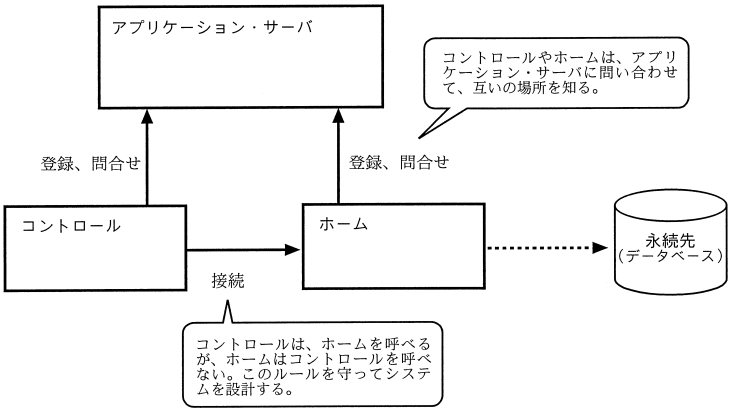

図 2　アプリケーション・サーバとコントロール，ホームの関係

アプリケーション・サービス・アーキテクチャでは，アプリケーションを業務ロジック部分と，データ・アクセス部分とに分類し，前者をコントロール，後者をホームが受け持つ．

コントロールは，実際の業務手続きをオブジェクト化したものである．ホームは，業務データの管理を行うもので，永続先（リレーショナル・データベースなど）への登録・更新などのやり取りを一括して行う．ホームは，オブジェクト・モデルとリレーショナル・データベースとのゲートウェイとしての役目を持つ．

このアーキテクチャ・モデルは, コントロールが主体となって動き, データの参照・更新などの必要に応じて適時ホームを呼び出す．コントロールはホームを呼ぶがその逆はない．

業務データは，ホーム，コントロール間を行き来するメッセージとして扱う．メッセージにすることで，業務データは値として各モジュール間に行き渡る．通信コストをコントロール，ホームのサービス呼び出しだけに限定させ，課題だった通信コストの解消を図った．

この方式にすると，オブジェクトの属性数に関わり無く，リモートメソッド呼び出しは 1 回で済むので，理屈上ではプロトタイプの方式に比べ，圧倒的に通信コストの消費を押さえる事ができる（およそ数百分の 1）．ただし，値渡しによりオブジェクトのコピーが頻発するため，オブジェクトの一意性を保証する機構が必要となる．

3.2　アーキテクチャの当システムへの適用例

当システムの上流工程は，プロトタイプと同様にオブジェクト指向開発技法に則って作業を進めた．結果として得られたユースケース図とクラス図のサンプルを図 3, 図 4 に示す．この図を元に，コントロールやホームを具体的にしていく．

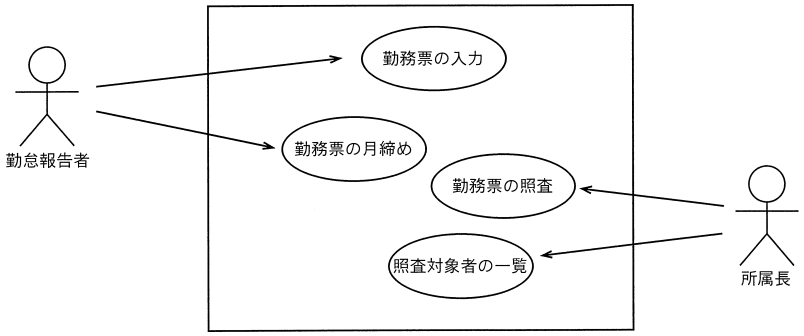

図 3　ユースケース図のサンプル

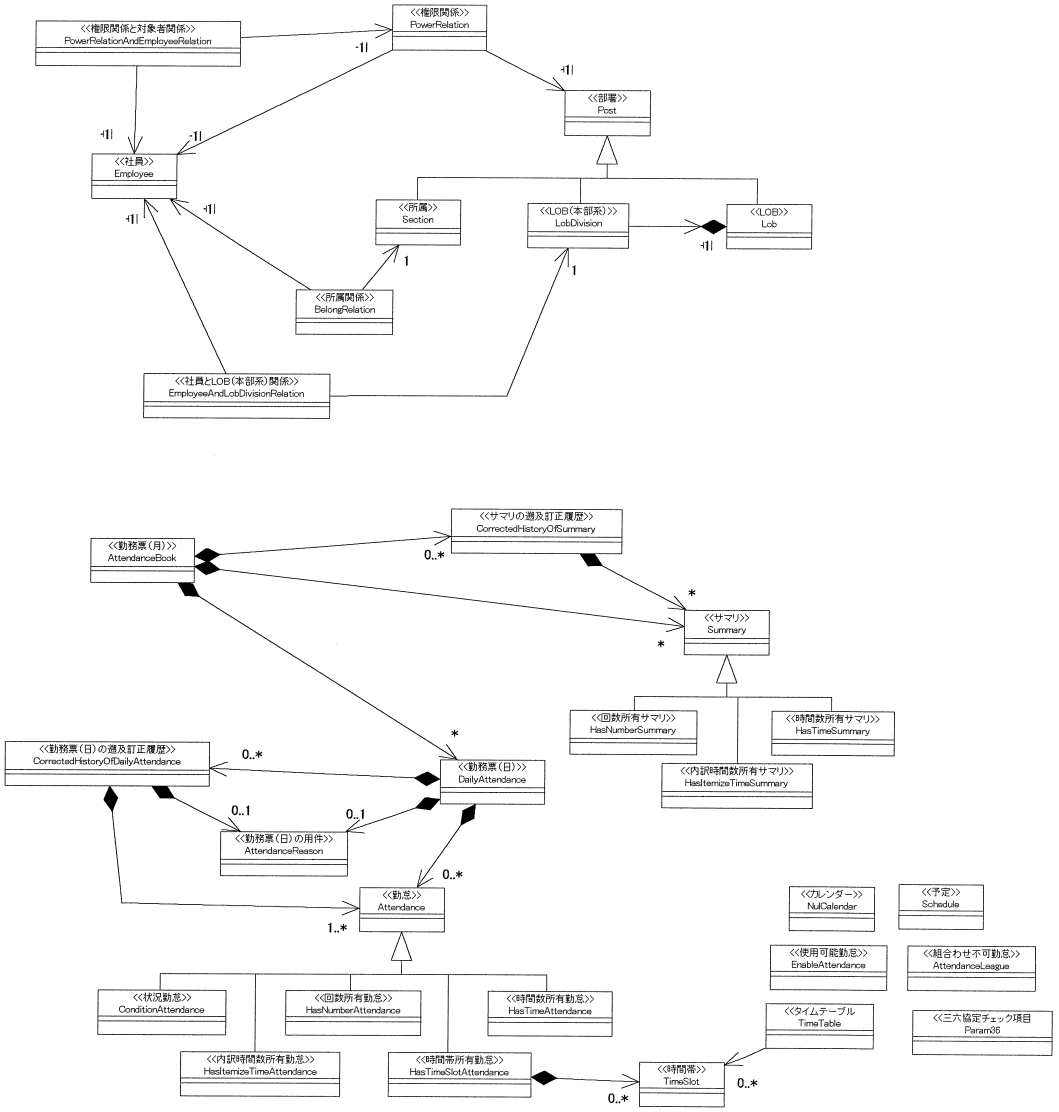

図 4　クラス図のサンプル

コントロールは，ユースケース図から抽出を行った．トップレベルのユースケース図にある各ユースケースごとにコントロールを設けた（表 1）．

表 1　ユースケースとコントロールの対応付け（例）

ユースケース	コントロール
勤務票の入力	勤務入力コントロール
勤務票の照査	照査コントロール

ユースケース図からコントロールを抽出したのに対し，ホームはクラス図から抽出した．基本的にクラスごとに対応するホームを設けるが，サブクラスに対してホームは設けず，スーパクラスに対してホームを設けた（表 2）．

表 2　クラスとホームの対応付け（例）

クラス	ホーム
勤務票(月)クラス	勤務票(月)ホーム
勤怠クラス	勤怠ホーム

コントロールは対応するユースケースごとに提供するサービスが異なるが，ホームは必ず，挿入，更新，削除の基本サービスを持つ．必要に応じて検索用サービスを一つないし複数提供する．

4.　アーキテクチャ実装で行った工夫

前述したアーキテクチャを実現するにあたって，選定したプロダクトによって実現できたもの，プログラムの工夫によって実現できたものとに分け，その内容を説明する．

4.1　プロダクト選定による工夫

本開発にて採用したプロダクトを以下に示す．

- Java 2 SDK Standard Edition v 1.2
- JavaIDL Compiler EA
- SYSTEM ν［nju :］Ver 3.2
- Oracle 7.3.4

各プロダクトの選定理由については，川口[1]を参照されたい．

4.1.1　CORBA の採用により実現できたこと

アーキテクチャの各モジュールが提供するサービスを全て CORBA インタフェースとして公開. イントラネット・サーバ, アプリケーション・サーバなどネーミング・サービスを提供する義務のあるモジュールは，IOR[*4] を利用して，管理対象のモジュールの位置を管理する．これによりネットワークを越えたモジュールの分散を実現することができた．

業務データは，CORBA の構造体（struct 型）として定義．それをメッセージとしてネットワークを流通させることでオブジェクトの値渡しを実現している．また構造

体はクラスと異なり継承の概念がないため，抽象クラスはすべて Any 型として扱うことにした．Any 型は文字通り Any（なんでもよい）型であるため，コンパイルレベルでは型チェックを行えない欠点も併せ持つ．

また，例外（Exception）も CORBA オブジェクトとして定義できるので，論理モデルで設計したサービスの内容を，ほぼそのままの形で実装モデルにマップすることができた．

4.1.2 SYSTEM v［nju:］の利用

プロトタイプと同様に CORBA-ORB として使用．また，Ver 3.2 の新機能である ERI（Embedded Resource Interface）を用いホームとリレーショナル・データベースとの橋渡しを可能にした．

ERI は，ユーザが作成したデータベース・アクセスルーチン（ストアドプロシージャ）を CORBA アプリケーションから呼び出す機能で，リレーショナル・データベースが持つ強力なデータ操作機能を活用する事ができる．

ERI により，単なるメッセージとして扱われる業務データを永続化することが可能になった．業務データをリレーショナル・データベースに格納するため，構造体に表 3 に示す属性を追加している．これらの追加属性を用いて，オブジェクト・モデルとリレーショナル・モデルの差異の吸収や，値渡しによる業務データの一意性の保証を行った．key，fkey は前者にあたり，updateCount は後者にあたる．

表 3　構造体に追加する属性

属性名	型(IDL)	役割
key	String	オブジェクトの ID．データベースの主キーに相当． データベースに格納するとき，主キー項目を作らず，既存の項目を組み合わせて複合キーとしている場合は，それらを組み合わせた値を格納させる．
fkey	String	外部キー．関係を表す必要がある構造体だけがもつ． オブジェクトモデルでは，親が子を知っているが，リレーショナル・データベースでは，子が親を知らなくてはいけない．この違いを埋めるために設定する．
updateCount	Long	データの更新回数をもつ． データの一意性を保証するのに用いる．

4.1.3 Java の採用により実現できた事

Java の実行形態の一つであるアプレット方式をとる事で，クライアントへのプログラム配信の必要を無くし，かつ HTML では表現しきれない高度な GUI の提供を可能にした．

ただし，クライアントの配布不要のメリットを享受する代わりに，アプレット固有とも言える多大なデメリット（実行環境の調整，セキュリティの設定など）をかぶる事になった．この件の詳細に関しては，川口[1]，山本[2]を参照されたい．

4.2 プログラム構造による工夫

4.2.1 構造体のラップ

Any 型を利用して抽象クラスの代替を行い構造体をうまく活用していたが，ある

程度の制約があった．構造体はそれ自体がメソッドを持つことができない．また Any 型自体のハンドリングが煩雑であるため，プログラミングは容易ではなかった．そのため，構造体にオブジェクトの皮を被せ，ユーザプログラムからはあたかも通常のオブジェクトを利用しているかのような状況を提供した．ちなみに，CORBA の構造体は Java では final クラスにマップされるため，マップクラスそのものを拡張する事ができない．そのため，ラップする以外の対応策はなかった．

このオブジェクトの皮を構造体に対してモデルと呼ぶ．

モデルは構造体と対に作成し，対応する構造体を一つだけ内包することができるようにした．モデルが提供するサービスを以下に示す．

- 更新チェック
- 値の比較
- クローンの取得
- 構造体のメンバへのアクセス

また，モデルでラップすることで構造体では表現できなかったオブジェクトの継承関係を再生することができた（図 5）．

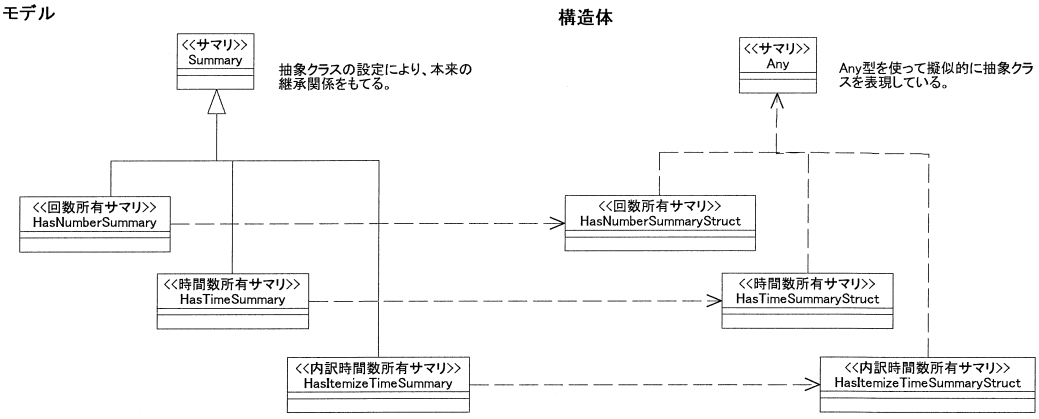

図 5　モデルによる継承関係の再生

4.2.2　代理オブジェクトの利用

CORBA 公開インタフェースであるコントロール，ホームを利用するとき直接公開インタフェースを呼び出さず，代理オブジェクト（以下，プロキシ）を置き，すべてプロキシ経由で各公開インタフェースへアクセスさせる．

プロキシの設置により，コントロール，ホームの実装を隠蔽し，各プロセスを繋ぐ実装方法を変更させることを容易にした．たとえば，コントロール，ホームの実装がネットワークやメモリのどこにいようとも，クライアントはそのことを意識する必要が無く，プロキシを相手にするだけでよい．

また，プロキシで公開インタフェース間を渡ってくるメッセージ（構造体）をモデルでラップして要求元に返す構造にしている．そのため，クライアント側のプログラマは，分散オブジェクトやアーキテクチャを意識することなくコーディングを行うこ

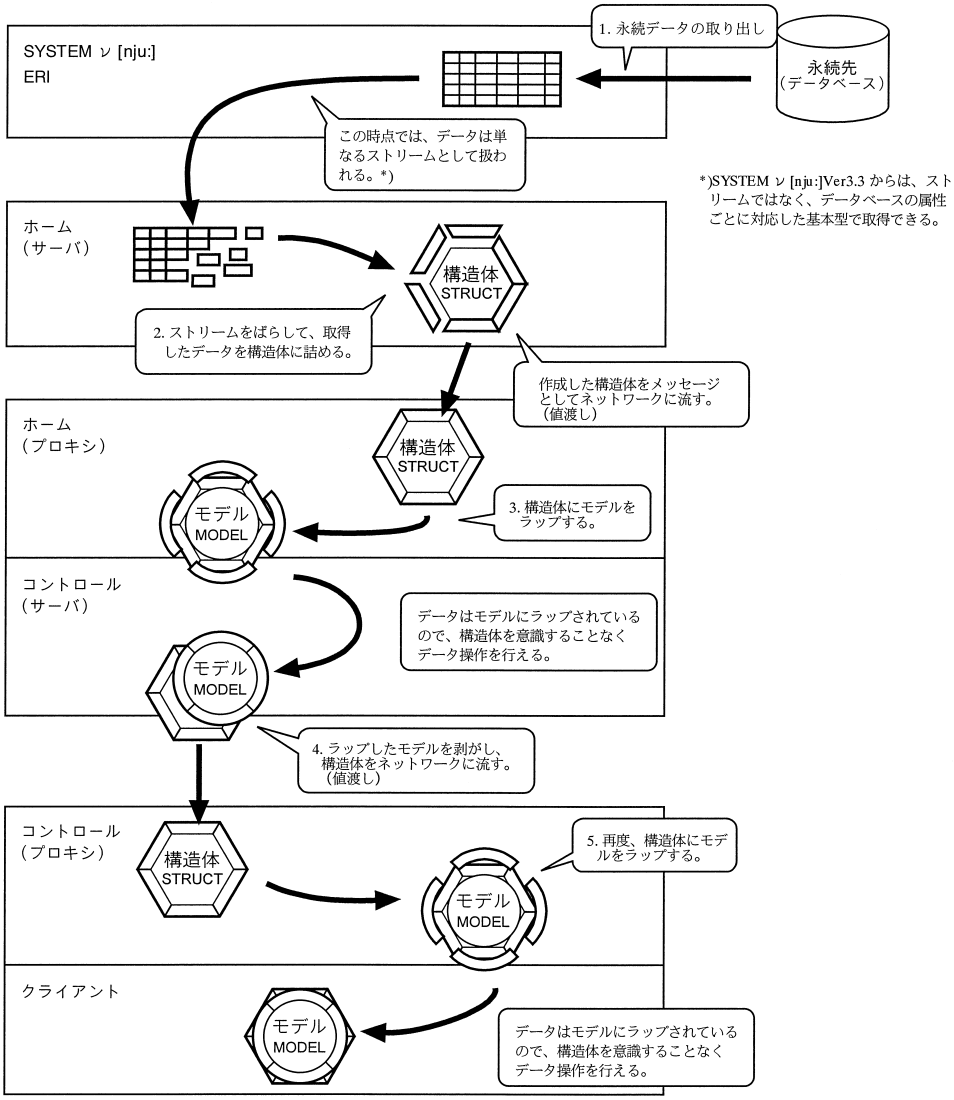

図 6　プロキシとモデルの関係

とができる（図 6）.

なお，プロキシによるモデル―構造体間の可逆変換を実現するため，ユーティリティクラスにコンバータを用意した.

4.2.3　オブジェクト生成の一元化

コントロール，ホーム，構造体，モデル，プロキシ，など当システムで使用するオブジェクトの生成はすべて，ユーティリティクラスに任せた．このユーティリティクラスをファクトリクラスと呼ぶ.

生成ロジックを一元化することで，実装方法の変更や，生成方法の制約（インスタンスの個数など）を管理することができる．ファクトリクラスの構造を図 7 に示す.

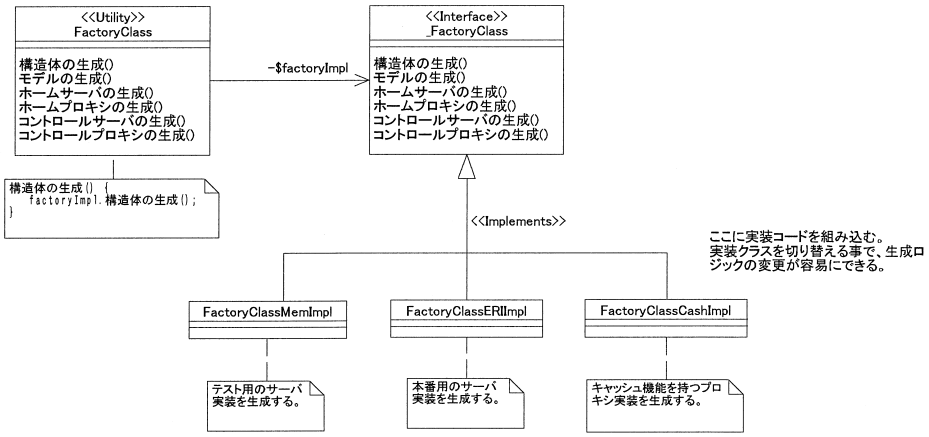

図 7　ファクトリクラスの構造

ファクトリクラスを使用してオブジェクトの生成を集中管理できた背景に，インタフェースと実装の分離を徹底的に行った事が挙げられる．構造体を除く全てのオブジェクト（コントロール，ホームとそれらのプロキシ，および構造体をラップするモデル）は，必ずインタフェースを定義し，別途実装クラスを用意してコードの実装を施した．

5. 考　　察

実行効率の確保，業務システムのスタンダードとして使用に耐えるアプリケーション構造の確保という目的レベルの異なる二つの課題に対して，今回提示したアーキテクチャを回答に出した．

効率面では，オブジェクトを参照でもつプロトタイプの方式に比べ，構造体を用いた一括転送の形式のほうが効果を挙げる事ができた．プロトタイプと当システムとはデータ構造や，実行環境が異なるため，実測値の比較が困難であるが，参考までに表4にプロトタイプでの参照方式，一括転送方式の比較結果を示す．

表 4　参照方式と一括転送方式のレスポンスタイムの比較

CPU	メモリ	参照方式[sec]	一括転送方式[sec]
486DX2/66MHz	32	25	5.1
Pentium/200MHz	96	10	2.7

1 ヵ月分の勤務データを画面に表示するのに要する時間．勤務データは約 500 の属性を持っている．

サーバ	US120U　256MB Memory Solaris2.5.1
クライアント	OS Windows95
ブラウザ	Netscape 4.05+JDK.1patch

通信回数では，500：1の差があるが，各方式ごとに内部処理が異なるため，［通信コストの節約＝レスポンスの向上］の等式は成り立たない．しかし，それを差し引いても一括転送方式が効果的だということは，測定結果からも明らかであろう．参考ま

でに，当システムでの測定結果を表 5 に示す．実行環境など条件がプロトタイプとは異なるが，これからもレスポンスが向上していることがわかる．

表 5　当システムで 1 ヵ月の勤務票を参照するのにかかったレスポンス・タイム

CPU	メモリ[MB]	所要時間[sec]
Pentium/166MHz	32	17.5
Pentium/166MHz	64	14.5
サーバ	US450　512MB Memory Solaris2.6	
クライアント	OS Windows95	
ブラウザ	Netscape 4.5+Java Plug-in 1.2	

もう一つの課題である社内イントラネット基盤インフラとしての要件に関しても，今回提示したアーキテクチャで満足のいく結果を出せたと思う．

まず，実行環境面について言えば，Java，CORBA の採用により，ネットワーク透過でプラットフォーム非依存の自由度の高いシステム環境を構築する事ができた．実際の開発においても，以下に示すような構成変更がたやすく実現できている．

1）開発機（Windows マシン）上でモジュールをコンパイル
2）そのまま同機上で実行し，テストなどを行う
3）モジュールをサーバに転送．サーバで実行し，テストを行う

一度だけコンパイルすれば，1)〜3）の作業は構成ファイルを数行書き換えるだけで実現できた．川口[1]や山本[2]で Java の「Write Once Run Anywhere」に対して警鐘を鳴らしているが，あるレベルの環境さえそろえば，この標語は言葉通りの意味を持つ事ができる．

アーキテクチャ面においては，現時点では明確な評価を挙げきれない．このアーキテクチャモデルを他の業務システムに当てはめて，初めて評価できる．ただ，当システムを実装することができたことや，（筆者が）別業務で類似的なアーキテクチャを提供して成功した経験などをから，見とおしは明るいと思う．

コーディングレベルの話では 4.2 節で述べたような工夫により，プログラマ全体の生産性向上を試みた．この試みは半分成功で，半分失敗と言える．

成功面は，プロキシ，モデルなどのラッパ層の提供により，利用者は実装を意識することなく，自分の担当個所のコーディングに集中することができた事であろう．
また，ファクトリクラスやインタフェースと実装の分離により，開発の状況に応じて，適時実装を切り替える工夫はかなり効果があった．当システムでは最終的に，表 6 に示すような実装を行ったが，ファクトリクラスのおかげで利用者は実装が変わったことすら意識することなくコーディングを進める事ができた．

失敗面は，アーキテクチャをフレームワーク的なところまで落とし込む事で生じる裏方的作業の発生である．プロキシ，モデルなどのラッパ，ファクトリクラス，コンバータクラスなどのユーティリティクラスを利用するのは，確かに生産性向上に繋がったが，そのラッパ，ユーティリティクラスそのものの作成は非生産的な作業が多かった．

まずなによりもコード量が多い．構造体―モデルの変換を行うコンバータクラスは，

表 6　当システムで作成した実装のバリエーション

実装名	内容
メモリ版サーバ	ホームのサーバ実装にSYSTEM ν [nju:]を用いず，メモリ上にデータを格納するテスト用の実装
ERI 版サーバ	SYSTEM ν [nju;]の ERI を使用したホームのサーバ実装．最終的には，このホームを用いる．
ノーマル版プロキシ	通常のプロキシ実装．
キャッシュ版プロキシ	マスターデータを扱うホームのプロキシにキャッシュ機能を実装したプロキシ実装．最終的には，このプロキシを用いる．

構造体とモデルの数だけ変換メソッドが必要になる．さらにファクトリクラスになると，システムが扱うオブジェクトの数だけ生成メソッドが必要になった．コード量もさる事ながら，処理自体は単純な内容が多いためコーディングは単調で苦痛を伴う．プロキシ，モデルのラッパは，利用者に実装を意識させないことが目的なのだが，当然，ラッパをコーディングする場合は実装を意識することになる．結果，ラッパの利用者，提供者に技術的な格差が生じる．生産性の向上を目指して難解な実装部を隠蔽させたが，それによりプログラマの大半はアーキテクチャの概要を知るだけで，詳細は裏方プログラマだけが知る事となった．生産性とスキルの難度は相反する要素だと思うが，この二極化はプログラム保守時に問題になるではと懸念する．

これらの懸念事項に対する見解としては，ラッパの自動生成などの自動化によるフレームワークの確立が挙げられる．ラッパのコードは，ある一定のパターンの繰り返しで組まれている．そのため，十分な時間と労力をかければすべてのラッパを自動生成することはできる．事実，当システムの開発でも Java や Perl を用いて簡単なテンプレート・コードの生成を行った．このテンプレート・コードが，IDL コンパイラが生成するスケルトンやスタブほどの完成度で提供できれば，先のような懸念は杞憂に終わるだろう．

6. おわりに

当システムが Java，CORBA を使って，どのようにシステムを構築したのか一通り報告した．川口[1]や山本[2]と合わせて読んでいただければ，幸いである．

また，当システムで行った工夫の大半は GoF[*5] に代表されるデザインパターンの適用である．パターンと Java の相性はよく，その効果も期待以上のものがあった．Java 開発を行う際にパターンの適用を念頭に置くことをお勧めする．

最後に，当システムのアーキテクチャを考案した，生産技術部山本史朗氏に深く感謝を示し，本稿の結びとする．

＊1　CORBA：Common Object Request Broker Architecture. 分散環境におけるオブジェクト間のやりとりを行うための仕様．オブジェクト指向技術の標準化団体 OMG（Object Management Group）が策定．

＊2　IDL：Interface Definition Language．プログラミング言語に中立なインタフェース定義言語．

＊3　ORB：Object Request Broker．オブジェクト間のすべての通信を処理し，分散オブジェク

トシステムのコアをなす．

＊ 4　IOR：Interoperable Object Reference. CORBA 規格で定められた分散オブジェクトへのアクセス方法を規定したフォーマット．

＊ 5　GoF：ソフトバンク刊「デザインパターン」のこと．4 人の作者が執筆した事から，Gang of Four（通称 GoF）と呼ばれる．

参考文献　［1］　川口真一,“ 勤務票システムの業務要件とシステム要件 ”, 技報, 63 巻
［2］　山本史朗,“ 勤務票システムにおける技術課題 ”, 技報, 63 巻

執筆者紹介　佐々木　勝信（Masanobu Sasaki）
1972 年生．1993 年国立一関工業高等専門学校化学工学科卒業．同年日本ユニシス(株)入社．電力関連の客先サービスに従事し，現在，生産技術部技術一室に所属．