基幹系向けミドルウェアのオブジェクトモデル

Object Model for Mission-Critical Middleware

石 田 政 海

要 約 オブジェクト指向技術を基幹系ビジネスアプリケーション領域に適用する気運が高まっている.

アプリケーション構造は大きく AP 本体，AP 制御構造，システム制御構造に分割することができる.

基幹系ビジネスアプリケーションを支えるミドルウェアは，メインフレームシステムで提供している厳しい機能，性能，品質要求を満たすと同時に AP との境界面を明確にする必要がある.

AP 本体部分のモデリングに関してはよく議論されているが，同様に基幹系ミドルウェアの領域へのオブジェクトモデリングは有益である.

本稿では，実績のある基幹系ミドルウェアから機能要件を抽出し，システム制御と AP 制御構造を AP から分離する基幹系ミドルウェアのオブジェクトモデルを提示する.

Abstract The trend has emerged to apply the object-oriented technology to mission-critical business application domain.

Application structure can organized with three major parts, that is, the main body of application, application control structure, and system control structure.

Middleware supporting mission-critical business application must satisfy severe factionality, performance, and quality requirements provided with a mainframe system, as well as the well-defined boundary to business applications.

It is frequently discussed on modeling of main body of application, howecer, applying the object modeling to area of mission-critical middleware is useful as well.

This paper introduces an object model of mission-critical middleware where the system control and application control structures are separated from a complete application system, while extracting functional requirements from the proven mission-critical middleware.

1. はじめに

現在では適用段階に突入しているオブジェクト指向技術であるが，日本ユニシスでは 1995 年頃に金融機関の肥大化する勘定系システムをリエンジニアリングするプロジェクトを発足させた．IT 技術要素としてのオブジェクト指向技術に注目して生産性・保守性・品質を向上させるため次の観点から開発可能性の検証を行った.

- 業務ロジックの部品化・再利用
- 基幹系ミドルウェア
- 統合開発環境

これまでの基幹系ミドルウェア機能は OS に依存した API 形式によって提供してきたがマルチプラットホーム対応を考慮して，基幹系ミドルウェアを含んだアプリケ

ーションアーキテクチャにオブジェクト指向モデルを適用することとした．

本稿ではその成果をふまえて，2 章で基幹系ミドルウェア要件定義を概観し，3 章でアプリケーション制御とシステム制御を業務アプリケーションから分離させるオブジェクトモデルを紹介する．Java はオブジェクト指向言語にとどまらず，インターネットとの高い親和性やマルチプラットホーム対応，コンポーネントウェア指向による生産性保守性の向上といった特徴からも注目されている．さらに新たな戦略として EJB（Enterprise Java Beans）や J 2 EE（The Java 2 Platform, Enterprise Edition）が発表されサーバーサイドにおけるビジネスアプリケーション開発環境及び実行環境へと拡大するに至っており，ミッション・クリティカル・アプリケーションへの期待も高い．このため最後の 4 章では，基幹系ミドルウェアのオブジェクトモデルと対比することで Java/EJB/J 2 EE の位置づけを考察する．

なお本稿での基幹系システムは，メインフレーム上で稼働する日本の金融機関向け勘定系システムを題材としているが，他業種における基幹系システムにも応用することが可能である．

2. 基幹系ミドルウェアの要件定義

金融機関向けの基幹系システムはミッション・クリティカル・アプリケーションの代表例であり外部要件としては大量高速処理，耐障害性など厳しい性能要求を満たさなければならない．

内部要件としては業務 AP（Application　Program）を基幹系ミドルウェアが提供するシステム制御から分離，多様なチャネルや金融商品の開発などに対応できる柔軟な保守構造を提供しなければならない．要件定義をまとめると次の 4 点となる．

・基幹系ミドルウェアとしての基本機能および処理性能
・アプリケーションとシステム制御構造の分離
・柔軟な保守構造
・業務 AP へのサービス機能

2.1 オンラインシステム概説

オンラインシステムの一般的なシステム構成の鳥瞰図を図 1 に示す．

営業店端末からの入力はプラットホーム固有の OS 機能や通信制御サブシステムを経由して Receiver が受け取る．Receiver は OS 層とアプリケーション層との接合点となる．通常 Receiver では通信制御サブシステムなどから入力元情報やプロトコル情報を得ることができるので，入力情報を付加するヘダー加工処理や入力デバイスから受け取った生の電文をシステム内部で定義されている標準電文書式に変換する編集作業にも責任を持つ．

入力電文処理には電文の追い越しを禁止するなどフロー制御が必要となる．このため加工された電文は受信用 Queue に書き込まれる．

ディスパッチは受信用 Queue から電文を一つずつ取り出し種別を判断してアプリケーション制御 TP（Transaction　Processing）に渡す．アプリケーション制御 TP はアプリケーション制御構造の実装であり業務処理 TP の呼出制御やトランザクション成立可否判断を行い，処理結果から応答電文を組立て要求元の営業店端末に返す責

任を負う．

業務処理 TP が電文を送出する場合，出力用のリダイレクタに依頼する．出力宛先は物理識別子ではなくニモニック（論理宛先）で指定する．リダイレクタというのは通常の Inquiry/Answer 型や交換型などの経路変更，電文を仮想端末からの入力電文として再入力する機能やプログラム間通信を実現するための機能を備えている．入力元や出力先は物理端末や回線だけではなくコンソール，プログラム，ファイルなども仮想端末として指定できる．端末や回線の特性差を吸収するためにデバイスという概念も導入している．例えば，プロトコルの異なる営業店端末には各デバイスが定義されていて，入出力編集や固有の制御処理はデバイス単位で局所的に処理する機構となっており入出力デバイスに依存しない透過的な機能サービスを実現している．

Sender は業務アプリケーションから出力要求のあった無編集電文（標準電文書式）を取り出して，論理宛先を物理宛先に変換し出力デバイスに応じた出力編集と固有な制御を行い，通信制御サブシステムに対して編集済み出力電文の送出依頼を行う．

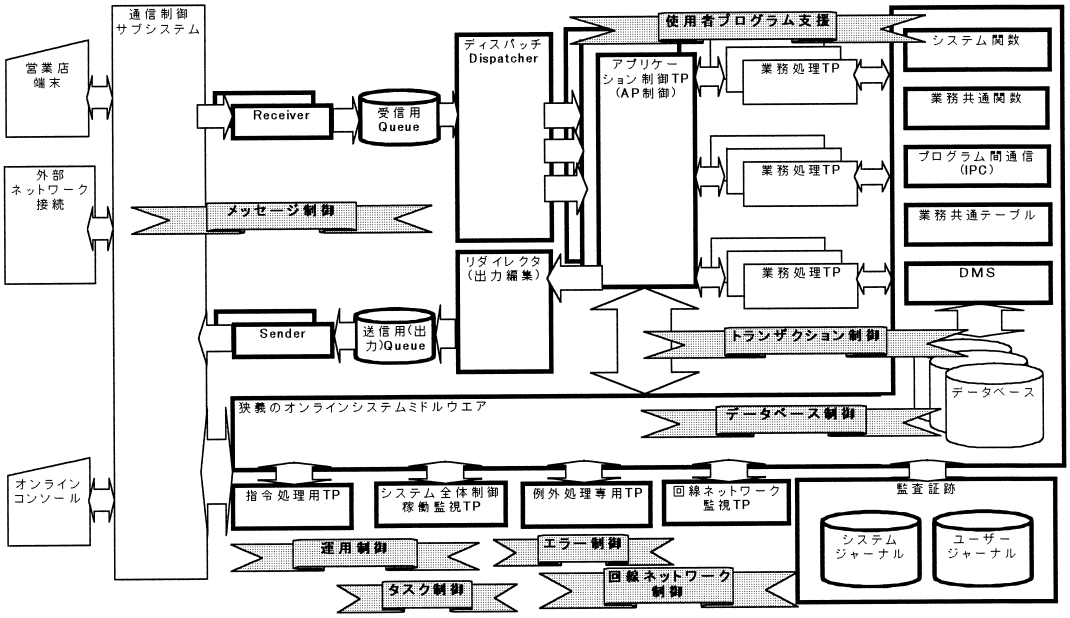

図 1　オンラインシステム構成図

2.2　基幹系ミドルウェアの機能分解

基幹系システムを機能別に見ると以下の制御機能やサービス機能に分けられる．重要な機能要件を以下に整理する．

1)　回線ネットワーク制御

入出力対象となる端末，回線，ネットワーク状態を管理する．システムの起動制御とオンラインサービスの開始は区別されている．また端末のデバイス種別や

回線の種別によってセキュリティに関わり業務 AP のサービス機能が差別化される．

2） メッセージ制御

端末からの要求メッセージを受け付け，内容を判断した上で適切な業務処理を行い，必要に応じて応答メッセージを指定された端末や回線に返す機能を実現する．運用上の要件から監査証跡や障害回復機能が必要となる．

3） タスク制御

端末からの処理要求を実行する業務処理プログラムの事をトランザクションプロセス（TP）と呼ぶ．TP の稼働状態を制御することで処理要求が円滑に実行されることを保証する．オンラインサービスと同様にシステム起動とサービスの開始は厳密に区別されている．

4） データベース制御

データベースに対する操作，排他制御，障害回復処理機能を備える．またトランザクション制御と協調しながら永続データに対する ACID 特性（Atomicity：原始性，Consistency：一貫性，Isolation：独立性，Durability：持続性）を保証する．

5） トランザクション制御

ACID 特性を保証する．データベースの更新処理とメッセージ制御のうち応答電文送出を同期させる．アプリケーション固有の処理から制御構造を分離させるためアプリケーション構造を，アプリケーション層，アプリケーション制御層，システム制御層の 3 層に分割する（図 2）．

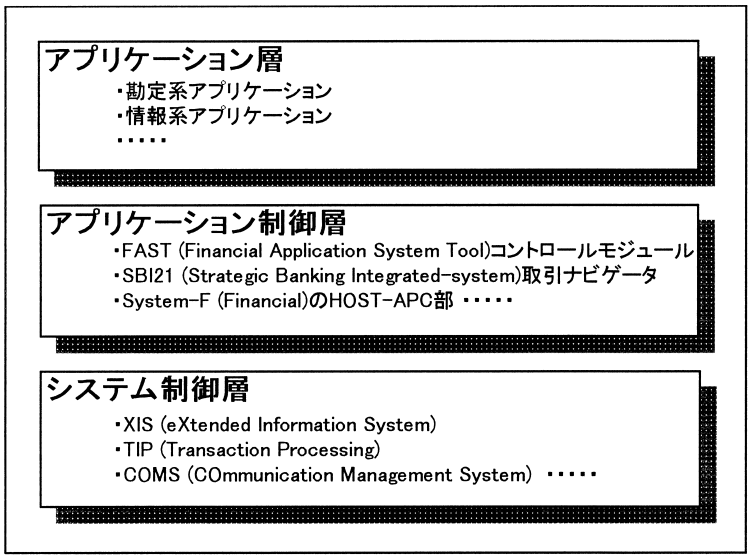

図 2　アプリケーションの 3 層構造

明示的なトランザクション制御（commit/abort 指示）はアプリケーション制御層の役割であり，端末は取引電文を基幹系システムに送信することで暗黙の

commit を指示している．トランザクションの実現はシステム制御層が提供する．

トランザクション制御に関しては業務要件がある．

・エラー処理時に明示的なトランザクション制御ができること

・上記状態で業務エラー処理に対応したトランザクション制御が可能なこと

6) エラー制御

業務アプリケーションからエラー処理構造を分離する．

・システムエラーと業務エラーの区別および管理方法

・エラー情報の採取，伝搬方法

・最終的なトランザクション制御（成否）との関わり

業務要件としては次のものがある．

・いかなるエラー状態も検出できる（原則）

・検出できないエラーは最終的に運用レベルで対策が講じられる

・発生したエラー内容を調査できる

・エラー状態を回復できる

7) 運用制御

基幹系システム稼働時に運用上必要となる機能を基幹系ミドルウェアのサービス機能として提供する．

8) 使用者プログラム支援

共通性が高いサービス関数や情報領域などの機構を提供する．システム関数は OS が提供する裸の API から分離させる目的を持つ．

上記のように，基幹系ミドルウェアの機能要件を各モジュールが独立または相互補完しながら満たしている．金融固有の機能要件にはバッチ処理（一括処理，大量帳票処理）があるが，データベースおよびビジネスロジックをリアル処理（オンライン処理）と共有する傾向が強いため基幹系ミドルウェアの機能拡張として実現されている[1][2]．

3. 基幹系ミドルウェアのオブジェクトモデル

3.1 概　　要

与えられたテーマは基幹系ミドルウェアの機能要件から同等機能を実現するオブジェクトモデル構造を提供することである．基幹系ミドルウェアを含む全体アーキテクチャ参照構造として次の四つのユニットを提示する．

・BAU（Banking Application Units）

・BCU（Banking Control Units）

・BDU（Banking Development & Deployment Units）

・BMU（Banking Meta-data Management Units）

BCU は，銀行業態におけるミッション・クリティカル・アプリケーションとビジネス・サポート・アプリケーションである BAU に対する総合システムイネーブラである．システム制御・メッセージ制御・運用制御・ネットワーク制御など高度に抽象化したサービス機能を実行時に BAU に提供することを目的とする．BCU，BAU に加えてメタデータ管理を行う BMU や開発環境となる BDU とも密接な機能連携を実

現する．

本稿では主に BCU 部分，すなわち基幹系ミドルウェアのオブジェクトモデルを対象とする．

3.2 BCU オブジェクトモデル

BCU オブジェクトモデルでは，システム制御とアプリケーション制御との分離を Control クラスと Message クラスで実現する．Message クラスは端末からの取引要求（取引電文）に対応し処理を実現するために複数の BusinessObject の呼出順序を制御して最終的なトランザクション成否判断を行い取引結果を端末に応答指示するまでの責任を負う．BusinessObject は Message から呼び出されアプリケーション固有の機能を実現するだけなのでシステム制御やアプリケーション制御から分離され透過状態となる．アプリケーション固有の処理においてシステム制御に関する機能が必要となる場合は Control クラスのサービスを利用する．Control クラスは業務処理実現に必要となるトランザクション制御, メッセージ制御, エラー制御や各種コンテキスト情報などをカプセル化して Message クラスや BusinessObject に提供する（図 3）[3]．

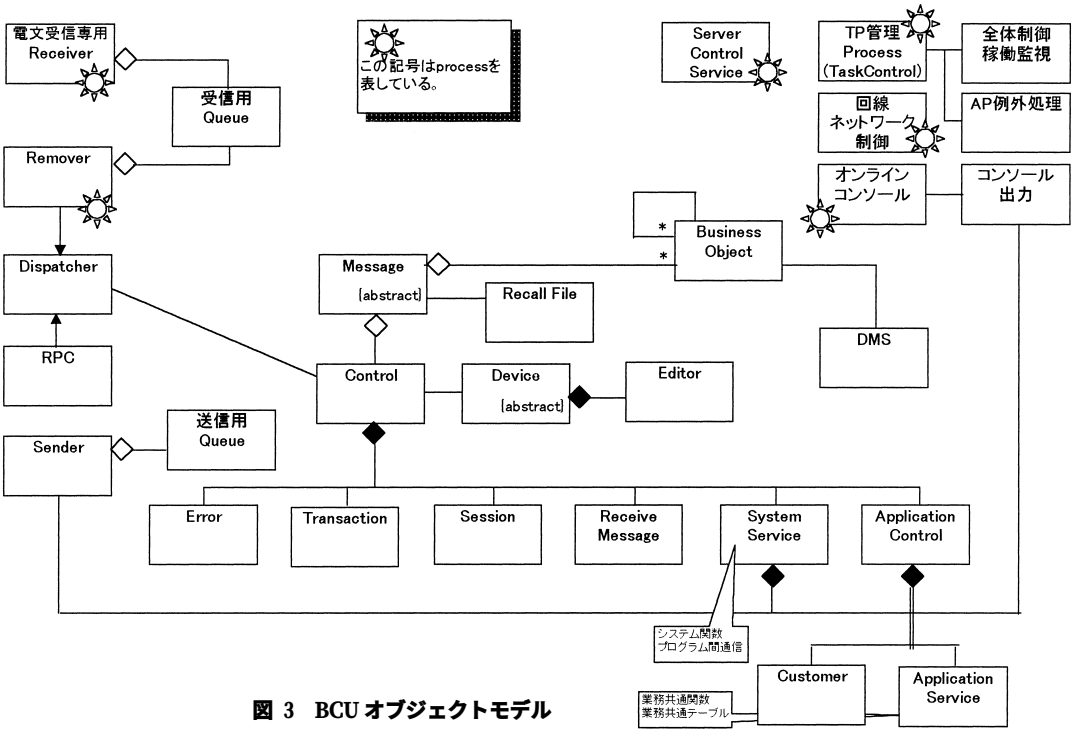

図 3　BCU オブジェクトモデル

基幹系ミドルウェアはコンカレントな処理要求に対して共有資源の排他制御を行いながら個別の処理要求に関しては異なるアドレス環境を用意して処理しなければならない. 個別の処理要求に対応する異なるアドレス環境のことをスタックと定義すると, BCU でのスタックは Control インスタンスをベースとして Device，Message，複数の BusinessObject のインスタンス対から成り立っている．

スタック中の ApplicationControl，特に Customer が Message インスタンスよりも

生成順序として上位に位置づけられているのは，ユースケースモデルにおけるアクタに関する情報たとえば個人情報，認証情報などによって処理内容を差別化する必要があるためである（図 4）．

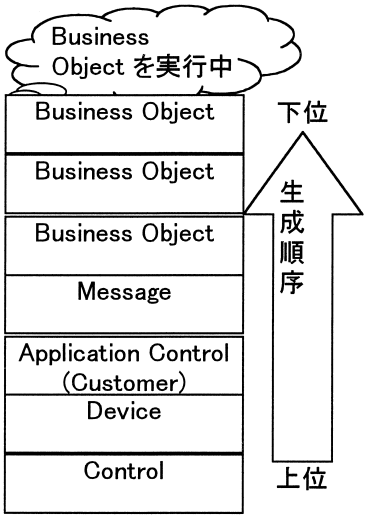

図 4　BCU スタック

1）　Control クラス

メッセージ制御やエラー制御，トランザクション制御などシステム制御の分離を実現するクラス．Client（端末）からリクエストされた一つの業務処理要求を実現するために作用する複数の業務処理オブジェクトは一つの Control インスタンスを共有することになる．

2）　Message クラス

Dispatcher クラスによってインスタンス生成されるアプリケーション制御を実現するクラスである．BAU の中で最初にインスタンス生成され，いわゆるメインプログラムに相当する．BCU では抽象クラスの Message を提供するが，業務アプリケーション担当はサービス要求, すなわち電文種別ごとに対応する Message サブクラスを作成する必要がある．

3）　Transaction クラス

Control クラスにカプセル化されたサービスの一つ．トランザクションの原子性を保証する機能を提供する．commit や cancel の機能を提供する．

4）　Error クラス

Control クラスにカプセル化されたサービスの一つ．エラー処理とエラー情報の蓄積機能を提供する．業務処理オブジェクトは Error インスタンスを共有することで処理結果を蓄積する．最終的には Message インスタンスに返されトランザクション成否の最終判断材料となる．

5）　Device クラス

デバイス固有のプロトコルやメッセージ書式変換，コード変換，フロー制御な

どの制御処理をカプセル化する．営業店端末や出力用のプリンタ，対外接続などはそれぞれ別のデバイスとなる．BCU では抽象クラスの Device を提供する．新規にデバイスを追加する場合は該当する Device サブクラスを作成する．

6） Business Object

業務モデルから OOA/OOD によって導出された業務処理オブジェクト群．Client からの要求によって最初に生成される必要のある業務処理オブジェクトはメインプログラムに相当する Message サブクラスに定義する．

3.3 システム制御とアプリケーション制御

BCU において中核となるオブジェクトは Control クラスと Message クラスの導出にある．

・Control クラス　システム制御を実現

・Message クラス　アプリケーション制御を実現

システム制御に関する諸機能を API 関数による実装から状態遷移をもったオブジェクトのサービスとして実装する．一つの処理要求を実現するための Business Object はシステム的な大域領域や共通領域ではなくて，専用のコンテキストとシステムサービスへの Entry Point の集合として一つの Control インスタンスを共有する．これによりアプリケーションは Control クラスを通して提供されるシステム制御に関するサービスを利用することができる．

Message クラスはアプリケーション制御を実現してアプリケーションから最終的なトランザクション成否判断や監査証跡など運用上必要となる諸機能の実現などを透過にする．これは Business Object が永続化までを含めたアプリケーション部品として構築されることを前提にしており，Message クラスが端末からの処理要求をアプリケーション部品の組み合わせによって実現する．Message クラスは Control クラスと異なりアプリケーションの制御ロジックを含んでいるため開発時にはアプリケーション制御担当が作成・保守する．

さらにインターネットや衛星などの新規チャネルへの対応は Device クラスでのプロトコル変換だけではなく，顧客の認証やサービスの差別化と密接な関わりがある．アプリケーション用の Control クラスの位置づけとして Application Control クラスを導出する．

1） Control クラスの役割と Business Object との境界

Control クラスの主要なインスタンス変数とメソッドイメージを表 1 に示す．BCU のサービスを享受するため Business Object を作成する場合に守らなければならない構造規約がある．

・一つの処理要求には一つの Control インスタンスを共有する

・業務処理オブジェクトはインスタンス生成時に Control インスタンスをアドレス参照可能とするように引数で受け取り保存しておく

・システム制御に関するサービスはすべて Control クラスが提供するサービスを利用する

Control インスタンスの共有方法を図 5 に示す（C++風による記述）．

① Control クラスのインスタンス変数を MyControl として宣言する．

表 1　Control クラス定義

Control クラスの主要なインスタンス変数とメソッド イメージサンプル	
■インスタンス変数	
station	入力元端末情報(or 回線)
message	受信電文
errorStatus	エラーステータス
transactionStatus	トランザクション状態
SessionStatus	セッション状態
■メソッド	
getStationInfo()	端末情報を返す
:	
setErrorStatus(errorCode)	システムエラーコードを設定する
checkErrorStatus()	システムエラー状態を調べる
:	
beginTransaction()	トランザクションを開始する
endTransaction()	トランザクションをコミットする
cancelTransaction()	トランザクションを破棄，回復させる
:	
sendMessage(destination,message)	論理宛先(エンティティ)に対して電文を出力する

② コンストラクタでは第1引数として Control インスタンスを受け取り①の MyControl に代入しておく．

③ すべてのメソッド頭部において，MyControl の状態を検査する．エラー状態であれば return 文等によって呼出元へ制御を戻す．

④ Business　Object のメソッド処理中，アプリケーション・エラーが発生した場合は，必ず MyControl インスタンスのサービスを利用して，エラー情報の採取処理を行う．

3.4　BCU メッセージ制御

銀行業務の普通預金入金を事例に BCU におけるメッセージおよびオブジェクト呼出の連鎖を紹介する（図 6）．

営業店端末から送信された普通預金入金電文を通信制御サブシステム経由で BCU 内部で実装されている Receiver が受け取り，受信用 Queue に insert して次の電文受信待ち状態に戻る．Remover は受信用 Queue から先頭にリンクされているメッセージを取り出す．制御は Dispatcher に渡され，ベースとなるスタックとして Control インスタンスを一つ生成して各種コンテキスト状態などを設定する．Control のサービスを呼び出すことで，入力元となる営業店端末に対応する Device を特定し Editor を使用して BAU 標準のメッセージ形式へメッセージ変換する．

Dispatcher はメッセージの内容から普通預金入金の Message サブクラスをインスタンス生成する．生成後，インスタンスを活性化させるメソッド（例えば run（））を呼び出し，制御を Message インスタンスへ移す．

普通預金入金 Message インスタンスには普通預金インスタンスが宣言されておりインスタンス生成する．その際 BCU 制御構造を実現するために Control インスタンスを引数として渡す．その後入金メソッドを呼び出す．

普通預金インスタンスは設計通りの業務処理を行う．ここでは CIF（顧客情報）を

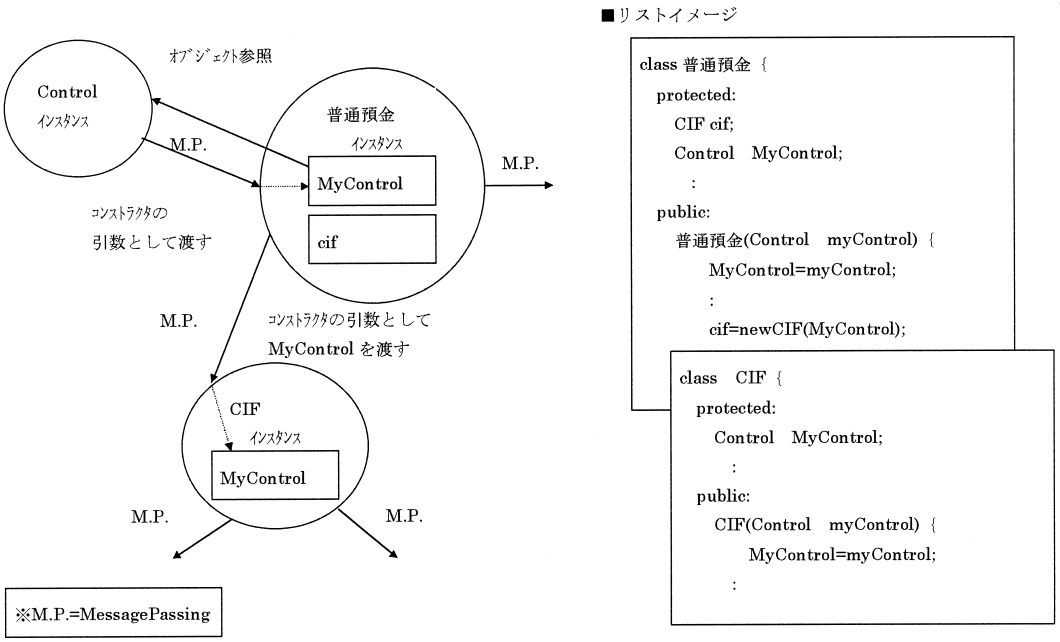

図 5 Control インスタンスの共有方法

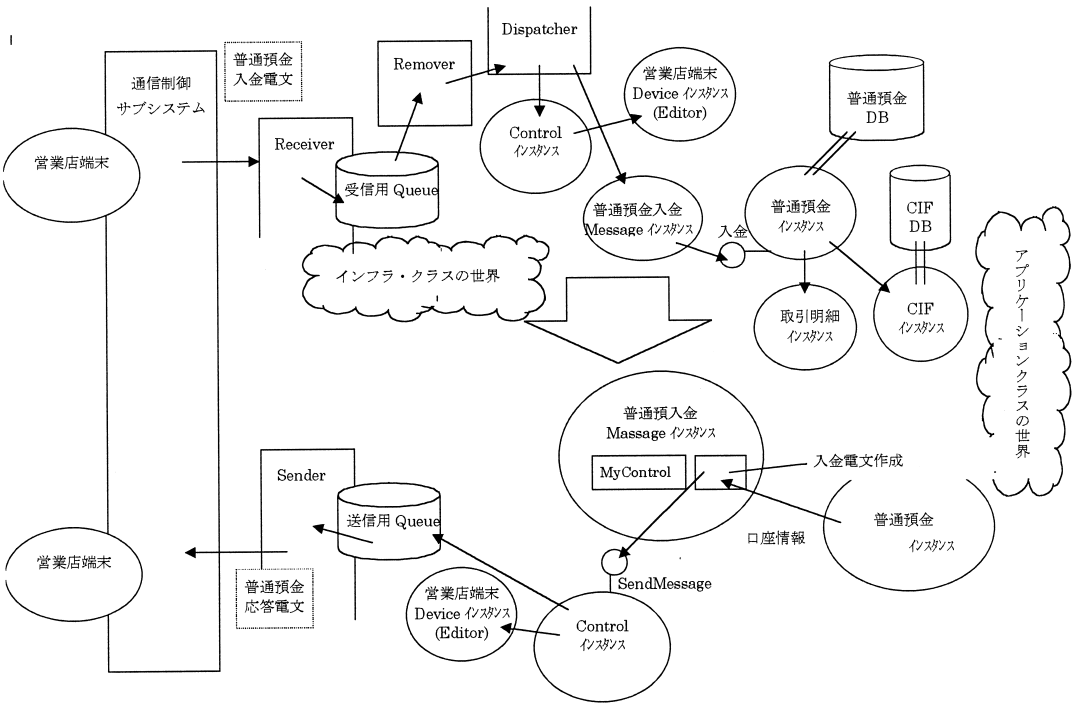

図 6 普通預金入金の事例

検査するため CIF インスタンスを生成，取引明細インスタンスを生成，普通預金 DB（データベース）を更新する．

応答電文を作成するために普通預金入金 Message インスタンスはメソッド呼出による引数の戻り情報や普通預金インスタンスから入金後の口座詳細情報を得て，応答電文を作成する．

Message インスタンスも Control インスタンスへの参照を持っている．Control クラスには電文を送信するための sendMessage メソッドが公開されており，作成した応答電文を引数にして sendMessage メソッドを呼び出す．

control クラスの sendMessage メソッドでは，論理宛先を物理宛先に変換して，出力先となる Device の Editor によってメッセージ変換を行い，送信用 Queue に insert する．

Sender は常時送信用 Queue を監視しており，メッセージを通信制御サブシステムに渡すことによって電文出力を実現する．

3.5 BCU トランザクション制御とエラー制御

3.5.1 トランザクション制御

トランザクションを制御する中核機能は Control クラスのトランザクション制御メソッドによって提供する．Control クラスはシステム制御に関わる多数の機能をサービスするためにそれぞれの機能を備えたクラスに対して実装の委譲を行っている．トランザクション制御に関する機能は Transaction クラスで実装している．

実際にトランザクションの開始とコミットやキャンセルを判断するのは Message クラスが業務処理オブジェクトのメソッド呼び出しからの戻り値や Control クラスに採取されたエラー情報を勘案して行う（図 7）．

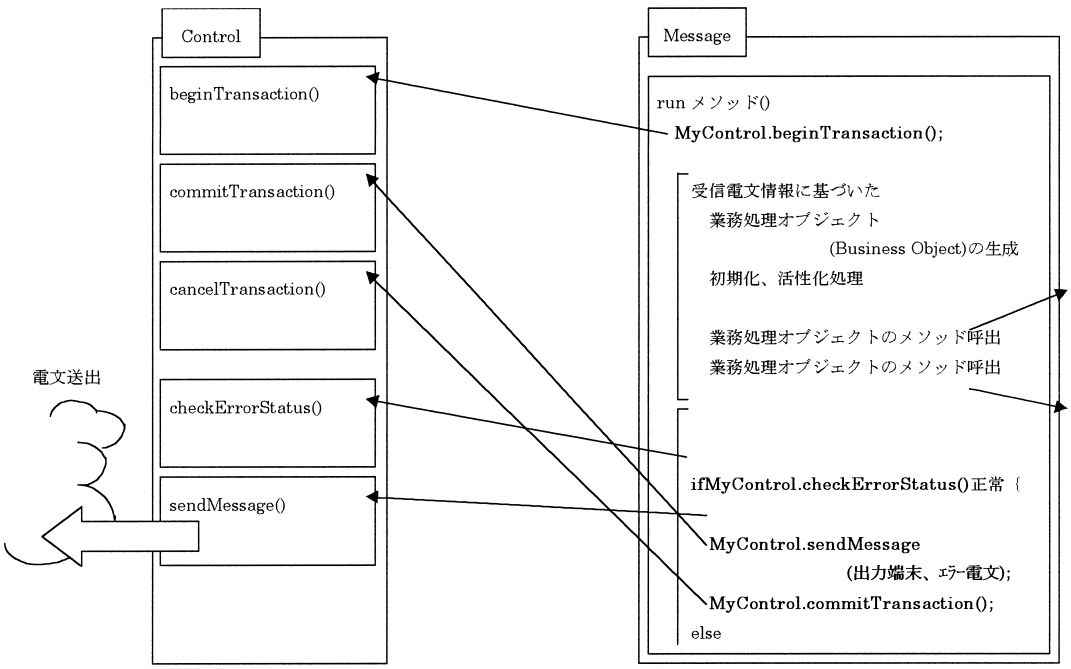

図 7　Control クラスと Message クラスによるトランザクション制御

Message サブクラスの主な制御処理の流れは以下の通りである．

① トランザクション処理とは独立した前処理
② トランザクション状態を開始するため Control クラスの beginTransaction メソッドを呼び出す
③ トランザクション内での前処理
④ 業務処理を実現する Business Object の生成及びメソッドを呼び出す
⑤ トランザクション内での後処理
⑥ ④の処理中にエラーが発生していないかどうかを Control クラスの checkErrorStatus メソッドを呼び出して調べる
⑦ トランザクション成否判断を行い，正常ケースの場合は応答電文を組み立てて Control クラスの sendMessage メソッドを呼び出して送信待ち状態とする
⑧ Control クラスの endTransaction メソッドを呼び出しトランザクションを成立させ（データベース更新と応答電文の送信）トランザクション状態を終了する

エラーが発生してトランザクション状態を不成立にした場合，データベースの状態は更新前の状態に戻される．オンラインシステムでは，これらの回復処理が完了するとエラーが発生した事実を情報としてデータベースに保存して，エラー発生を通知する応答電文を受信待ち状態の端末に送信する必要がある．

3.5.2 エラー制御

オブジェクトに対するメソッド呼び出しが階層的に行われているためエラーが発生した箇所におけるエラーの意味と呼び出し元に復帰した箇所でのエラーの意味が異なっている可能性がある．一つの事象に起因するエラー情報は階層に応じて複数の意味を持つことになる．また深い階層で発生したシステムエラーも業務エラーとしての意味を付加することで正常に処理完了させることができなかった旨を応答として通知する必要がある．エラー制御としては以下の機能の実現が重要である．

・エラー情報採取
・エラー電文送信（端末からの処理要求に対するエラー応答）
・トランザクションキャンセル
・後続処理の中断

Control クラスの委譲として Error クラスを実装する．Error クラスでは階層ごとにエラー情報を蓄積，状態検査するためのサービスを提供する．
階層的な呼出によって入れ子状態にあるオブジェクトは，エラー情報を Control インスタンス内部に待避させておき，呼出元へ順次復帰することによって Message インスタンスに制御を戻す．復帰したメソッドの最後では，必ず Control インスタンスに待避されているエラー情報を調べてトランザクションの確定処理またはキャンセル処理を実現する．

図 8 の例（C++風）では，CIF クラスの存在 check メソッド内部で CIF ファイルが存在しないという致命的エラーが発生した．このため Control クラス（myControl インスタンス）の setErrorStatus メソッドを呼び出して“ CIF 不存在 ”というエラ

ー情報をスタッキングする．

次に存在 check を呼び出した普通預金クラスの入金メソッドに復帰したあと“CIF 未登録”（厳密には未登録ではなくシステム障害）というエラー情報を追加してスタッキング指示する．

最終的なエラー判断は Message クラスで行う．エラー情報や致命度合いを判断する情報は Control クラス（myControl インスタンス）の errorStack にスタッキングされており checkErrorStatus メソッドで調べることによってトランザクションを不成立にしたりエラー電文を送出することができる．

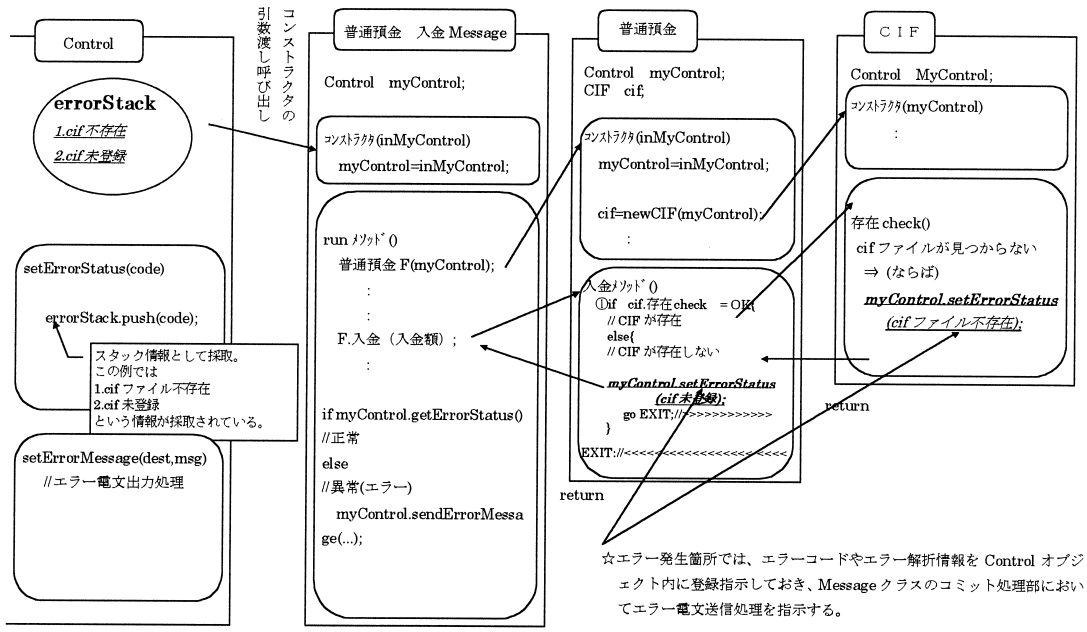

図 8　エラー情報の採取

3.6 CORBA/HTTP

BCU はメインフレームによる Queue を介在させた通信機構をベースにオブジェクトモデルへ展開している．そのため MQ などのメッセージ交換方式とは親和性がよい．一方 EJB では RMI（Remote Method Invocation）を Client との通信手段としている．CORBA（Common Object Request Broker）の IIOP（Internet Inter-ORB Protocol）や GUI 部分に HTML（Hyper Text Markup Language）ブラウザを利用するため HTTP（Hyper Text Transfer Protocol）を選択する場合もあり，これらは RPC（Remote Procedure Call）方式となる．

通信制御装置や OS 機能の対応によっては RPC 方式も Queue を介在させた機構へと内部変換することで Device の一種にしてしまうアーキテクチャもある．

BCU では，トランザクション制御やメッセージ制御の実現として Control クラスと Message クラスが連携したフレームワークを与えているため RPC 方式に対しては Remover に相当する RPC クラスを用意する．RPC クラスは RPC の種類単位にサブ

クラスを用意して外部から呼出可能なインタフェースを実装する（図9）．メッセージ交換方式とRPCでは実現構造が異なるため次の前提や制約を留意する必要がある．

・関数呼出形態であるため応答電文はreturnの引数で返される

・Inquiry/Answerが基本であり交換型やブロードキャストは実現できない

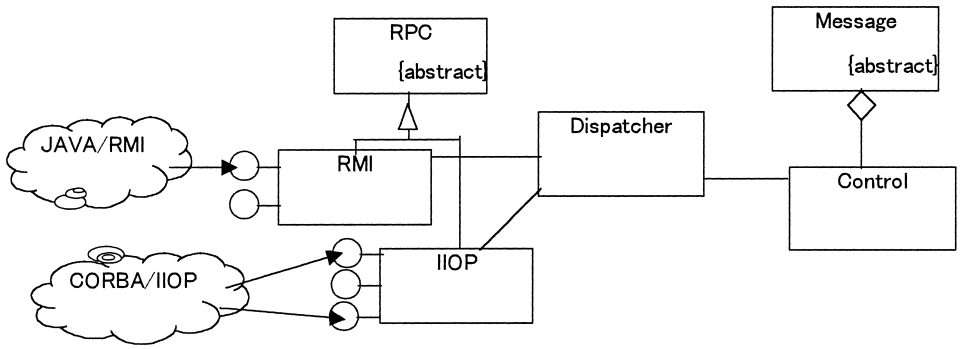

図 9　BCUのRPC対応

4.　Java/EJB/J 2 EEの位置づけ

EJBを包含するJ 2 EEは，開発環境を含む基幹系システムモデルと対応している（図10）．Java/EJB/J 2 EEを基幹系システムに適用する場合に両者を比較することによってEJBの位置づけやJ 2 EEへの機能拡張内容が明らかとなる[4][5]．

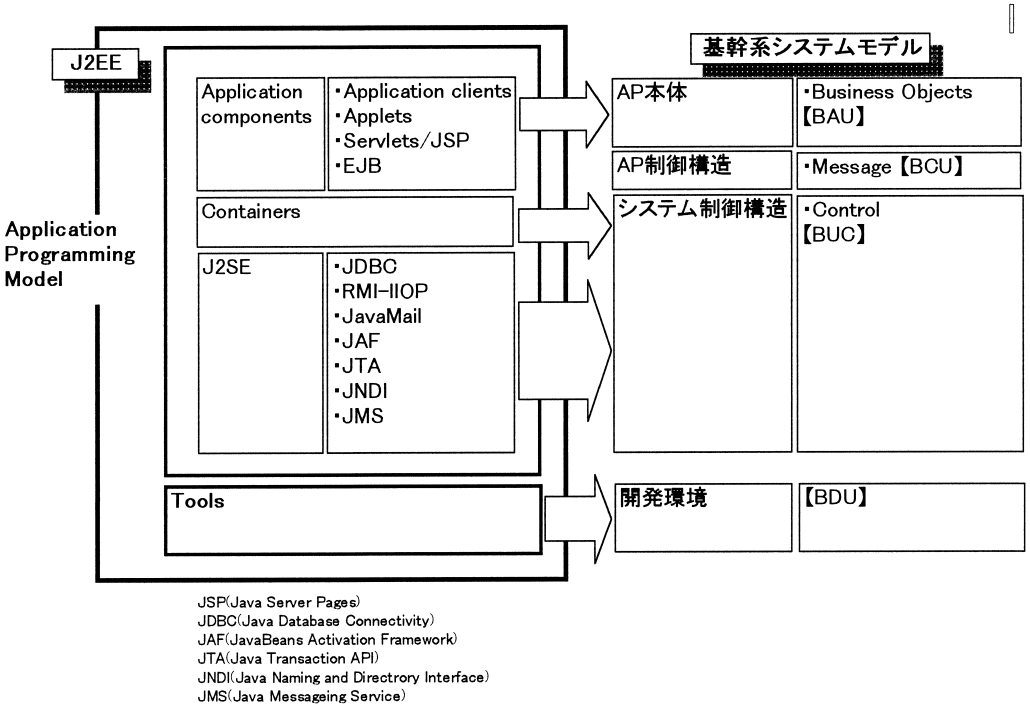

図 10　J 2 EEと基幹系システムモデルとの対応関係

1)　EJB のモデル構造

アプリケーション本体のモデル構造については言及されていない．モデル構造は保守生産性に影響する．また AP 制御構造やシステム制御構造との境界面を明らかにする必要がある．

2)　AP 制御構造に該当する機構

勘定系システムのように複雑な取引を実現する場合に AP 制御構造が重要となる．AP 固有の制御処理を含むことからコンテナでの実装ではなく EJB の一部として実装する．EJB として実装することで J 2 EE の目的である異なるアプリケーションサーバーへの移植性も実現できる．

3)　システム制御構造からみた不足機能

システム制御機能は基幹系ミドルウェア機能の視点から整理することができる（表 2）．主な不足機能としては以下のものがある．

・複数クライアントの制御や交換型によるメッセージ処理（回線ネットワーク制御）
・スケジュール管理（タスク制御）
・稼働監視やコンソールからの運用指示（運用制御）

表 2　基幹系ミドルウェア機能による対応表

基幹系ミドルウエア機能	BCUオブジェクトモデル	Java/EJB/J2EE
回線ネットワーク制御	回線ネットワーク制御	ー(複数端末制御,交換型など)
メッセージ制御	Dispatcher,Device,Editor,RecallFile, ReceiveMessage,Session,Control	SessionBean(セッション管理)
タスク制御	TP管理process	ー
データベース制御	DMS	EntityBean,JDBC
トランザクション制御	Transaction,Message,Control	EJBコンテナ,JTS/JTA
エラー制御	Error,Message,Control	Java Class Libraries(構造化例外のクラス)
運用制御	オンラインコンソール	ー
使用者プログラム支援	SystemService,ApplicationControl	Java Class Libraries
(ネーミングおよびディレクトリサービス)		JNDI
(クライアント間プロトコル)	Device(任意のプロトコル)	RMI/CORBA

4.1　EJB/J 2 EE が持つ特性

EJB や J 2 EE の特性を二つにまとめることができる．

・システム制御構造からの AP の分離（コンテナ）
・コンポーネント

システム制御構造をコンポーネント化したとしても実行時の安定性には寄与するが，アプリケーションそのものの生産性にはあまり寄与しない．コンテナの導入によってシステム制御構造をアプリケーションから透過にすることで，システム制御構造との関わりにおいて生産性を向上させることをねらっている．EJB/J 2 EE が想定しているアプリケーション形態は，thin-client から inquiry-answer 形式によるデータベース更新処理に重点を置く thin-application タイプにフォーカスしたシステム，とな

っている．

4.2 EJB/J 2 EE への取り組み

EJB/J 2 EE は今後も機能拡張のためバージョンがあがっていくことが予想される．基幹系システムに適用する場合には技術論だけではなくソフトウェアのライフサイクルやサポート体制など総合的な観点に立って情報技術の選択を決定しなければならない．そのうえで取り組み姿勢を使い分ける必要がある．

1） EJB への拡張

アプリケーション制御構造は EJB に対する機能拡張として実現可能である．EJB への機能拡張は移植性が保証されるように拡張するべきである．

2） コンテナへの拡張

システム制御構造のうち不足機能はコンテナで実現することが可能であるが，コンテナへの機能拡張は EJB の移植性を阻害する可能性がある．コンテナに追加される機能は EJB からは透過な機能として設計されなければならない．

5. お わ り に

基幹系システムを構築するためのシステム基盤として基幹系ミドルウェアのオブジェクトモデルを提示してみた．その実現コンセプトは EJB/J 2 EE と同じく，究極には業務アプリケーションの保守性・生産性の向上と稼働時における安定性を目指すものである．

オブジェクト指向技術は機能およびデータ構造を抽象化・局所化できる設計・実装技術としては秀逸であり，基幹系ミドルウェアの実装技術としては最適であると言える．本稿では基幹系システムの構造が三層に分割されることを主張した．アプリケーション構造とアプリケーション制御構造とシステム制御構造である．BCU を基幹系ミドルウェアの参照モデルとして利用すれば EJB/J 2 EE や他の基幹系ミドルウェアとを機能比較検証することが可能になるだろう．しかし現時点での技術は BCU も EJB/J 2 EE も同様にシステム制御構造にフォーカスされたものとなっている．本来の目的を考えれば，アプリケーション制御構造やアプリケーション構造に着眼したオブジェクトモデルやフレームワークが議論されなければならない．

参考文献 ［1］ UNISYS A シリーズ金融機関向け総合オンライン・システム SYSTEM-F FCM 解説書 MCS 編, 1992 年 7 月.
［2］ UNISYS シリーズ 2200 統合オンライン・システム XIS システム解説書レベル 3 R, 1995 年 3 月.
［3］ Martin Fowler with Kendall Scott, 羽生田栄一訳, UML モデリングのエッセンス, アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1998 年 10 月.
［4］ Java 2 Platform Enterprise Edition specification, v 1.2, http : //java.sun.com/products/ejb/, 1999.
［5］ Enterprise JavaBeans specification, v 1.1, http : //java.sun.com/products/ejb/, 1999.

執筆者紹介　石　田　政　海（Ishida Masami）

1985年立命館大学法学部卒業，同年日本ユニシス(株)入社．統合OAシステム開発，SWIFT　CBT，SYSTEM-F制御系開発，オブジェクト指向技術適用の企画推進，SBI 21 PC開発環境の開発に従事．現在，生産技術部情報技術室所属．