大規模ミッションクリティカルシステムを支えるアーキテクチャとフレームワーク

Architecture and Framework to Support Large Scale Mission Critical System

鮫島荘介

要　約　大規模ミッションクリティカルシステムであるANACoreの開発では，AirCore（Airline Core Systems Solutions）をベースに，ソフトウェアアーキテクチャを策定し，またそれを実現するフレームワークを開発した．策定・開発したアーキテクチャとフレームワークは，SOA（Service-Oriented Architecture）を考慮し，メインフレームの性能と運用性を実現するべく様々な工夫を凝らしている．

本稿では，ANACoreに適用したソフトウェアアーキテクチャとフレームワークを紹介し，ノウハウの一端を解説する．

Abstract　ANACore development was a development of very large-scale mission critical system, and the system was developed based on AirCore (Airline Core Systems Solutions). Software architecture to support such system was designed and a framework to realize the design was developed. This architecture and framework was based on SOA (Service-Oriented Architecture) and to inherit the performance and operability of host mainframe computers.

This report covers the software architecture and framework of ANACore, its features as well as the knowledge behind it.

1. はじめに

近年のオープン系システム開発において，フレームワークと呼ばれるソフトウェア部品を用いることは一般的である．フレームワークは，広義にはソフトウェアの共通部品全てを指して利用されることもあるが，言葉の通り“骨組み”という意味を踏まえると，アプリケーション開発に“プログラム言語やミドルウェアの提供機能を拡張し，生産性あるいは保守性の高い実装方法を提供するソフトウェア部品”であると考える．

ここで言うところの“実装方法”を提供する点が重要である．実装方法が提供されるということは，フレームワークを開発/選択することが，アプリケーション開発者がどのようなソースコードを作成するか，つまりソフトウェアアーキテクチャをどのようにしていくのかを定義することも意味している．

ANACore[*1]は，ミッションクリティカルシステムであり，稼働までの生産性はもちろん，稼働後の保守フェーズにおいても，高い保守性と，品質の良い改修を継続できる必要がある．またその開発プロジェクトであるProjectAI[*2]は，非常に大規模なプロジェクトであり，多くの協力会社が同時に，時に分散しての開発が実施できる実装方式を提供する必要があった．

本稿では，ANACoreのアプリケーションを下支えするフレームワークとソフトウェアアーキテクチャについて，2章で概要を述べ，3章で詳細を解説する．4章は評価と考察である．

2. ANACore フレームワークの特徴

2. 1 大規模ミッションクリティカルシステム用フレームワークの考慮点

大規模ミッションクリティカルシステムに用いるフレームワークで考慮すべき特性にはどのようなものがあるかは，対象とする業務や非機能要件等により異なるが，ANACore フレームワークに施した実装項目を整理すると，以下に分類できる．

1） 柔軟な運用を実現する機能の提供

リアルタイムでのシステムの監視，稼働状態が把握できる機能と，動的な閉塞など，システムの振る舞いをコントロールできる機能

2） 性能向上のための機能の提供

データベースの検索結果のキャッシュ，業務処理結果のキャッシュ，データベースアクセスの効率改善，並行処理など性能向上のための機能

3） 生産性・品質向上と開発者の実装負担軽減を目的とする機能

データベースアクセス部品など設計成果物から様々なソースコード・設定ファイルを自動生成する機能，エラー処理など標準化された振る舞いの自動処理の機能

4） 業務実現の共通部品

文字列処理，入力チェック処理，業務的な共通処理などの各種ユーティリティ，ログイン/ログアウト処理や認証情報の管理機能

ANACore ではメインフレームで動作していた前身となる able-D[*3] と同レベルの運用性，性能を目指したため，1），2)に特徴的な機能を持つことになった．これは以下の問題の対応としてフレームワークに機能を拡張して実現したためである．

- 運用の問題

  メインフレームでは，OS 機能レベルで提供される柔軟な運用を可能とする仕組みがある．ダイナミックに特定機能を処理するプログラムのみのリリースや，処理スレッド数の制御，プログラムの停止が可能であった．これらは一例であるが，どれも JavaEE ミドルウェアの標準機能では提供できない機能である．

- 性能の問題

  高速な CPU と高速なバスでつながるメモリを持つメインフレームと同等性能を出す必要がある．ANACore では，Linux/UNIX のオープンサーバを数多くそろえ，台数を増やすことで処理量の性能を増やすというハードウェア構成である．この場合システム全体の処理性能（スループット）では問題ないが，1 機能あたりの応答性の性能では，単純に CPU 性能を比べても同等性能が出せない．

2. 2 ANACore フレームワークの概要

Java 開発では，有償や OSS（オープンソースソフトウェア）のフレームワーク製品が豊富であり，既存製品を利用したうえで，さらなる生産性向上等を目的にプロジェクト特性を考慮した拡張や開発を行うことが多い．ANACore フレームワークは，ANACore 開発のために開発したフレームワークであり，AirCore[*4] のフレームワークである SHARED（シェアード）モジュールをベースに，複数の既存フレームワーク製品も組み合わせて開発したものである．このフレームワークはメインフレームで動作していた able-D で実現していた性能・運用性を

実現するよう工夫しており，以下の特徴がある．

- AOP（Aspect Oriented Programming）を採用し，業務プログラムの処理の呼び出し前，呼び出し後に，フレームワーク処理を実行することができる．この仕組みのもと，メインフレーム OS およびメインフレーム上で動作するソフトウェアが持つ基盤機能と同等の機能を実現した．リアルタイムな状態監視やスレッド制御機能を含め，充実した運用機能を持つ．
- メインフレームと同等の性能を，オープンサーバ構成で達成できるよう，複数の性能向上施策およびアプリケーション開発者が利用できる性能向上機能を持つ．
- ANACore 開発プロセスのツールと連動させ，設計書やデータベース構造からの自動生成を活用し，生産性向上だけでなく品質も高い開発が実現できる．
- 画面などのインタフェース処理部分と，業務処理部分を分割したアーキテクチャを採用しており，SOA を前提とするサービス志向の開発が可能である．
- モジュールというサブシステム単位に開発，ビルド，テストが可能で，オフショア開発や複数のソフトウェア開発会社への分割発注が容易となる．

3.　ANACore アーキテクチャとフレームワーク詳細

本章では，ANACore のソフトウェアアーキテクチャとフレームワークの詳細を説明する．

3. 1　ソフトウェアアーキテクチャ全体像

ANACore アプリケーションは，層と呼ぶ異なるアプリケーション構造が複数組み合わせられたソフトウェアアーキテクチャとなっている．層は，サービス層と，サービスクライアントと称するサービス層を利用する複数の層で構成される（図 1）．

- サービス層
  画面等のインタフェースに依存しない，純粋な業務処理機能と業務データの総称である．サービス層の内部は，業務領域別のサブシステムに分割されており，これをモジュールと呼ぶ．モジュールは専用のデータベーススキーマを持ち，業務処理で利用する．また各モジュールは，業務処理を提供するインタフェースを複数持ち，このインタフェースで提供される機能をサービスと呼ぶ．
- サービスクライアント
  サービス層を利用する層の総称であり，CUI 端末や GUI のブラウザ，外部システム接続のインタフェース等，接続形態毎に外部とサービス層を仲介する役割を担う．サービス層を業務処理に注力させることを目的とし，画面表示処理など外部とのインタフェース固有処理をこの層で実装する．

純粋な業務処理に特化しているサービス層と，サービス層から接続要件毎に切り離すサービスクライアントという階層を持つことにより，SOA を実現する基盤となる．

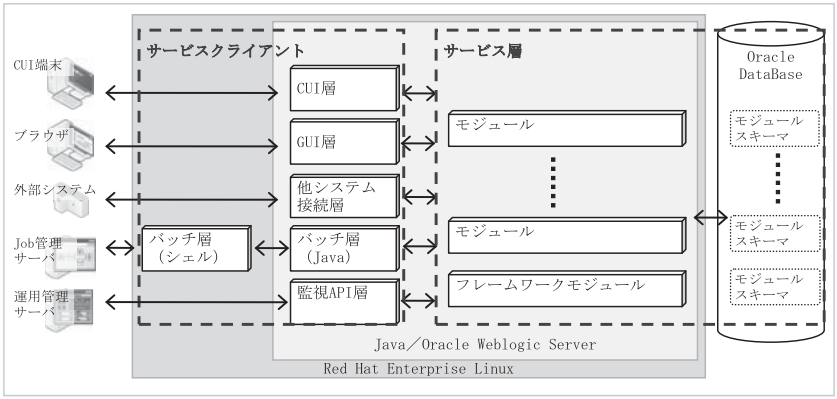

図 1　アプリケーション構造

3. 2　層とフレームワーク

各層はそれぞれ独自のフレームワークを持つ．サービス層はフレームワークをモジュールの一つとして持ち，他のモジュールやサービスクライアント層に対してフレームワーク機能を提供する．サービスクライアントは，フレームワークモジュールのサービスクライアント向け機能を利用できることに加え，インタフェース毎に固有のフレームワークを持つ．

3. 3　サービス層のアーキテクチャとフレームワーク詳細

サービスクライアントから呼び出され，業務処理を行うサービスのアーキテクチャとフレームワークについて詳細を説明する．

3. 3. 1　サービス層のアーキテクチャ

サービスは業務領域によって分割されたそれぞれのモジュールとフレームワークモジュールで構成されている．モジュールは自身の業務領域について，サービス層に対して純粋な業務機

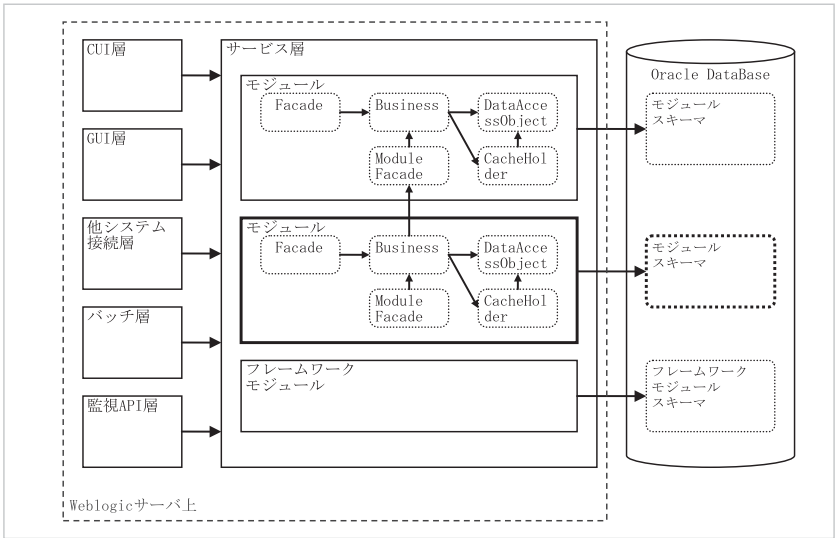

図 2　モジュールの構成

能を提供することを目的とし，対象業務領域の機能と，処理に必要となるデータの管理を司る．モジュールは，図 2 に示す五つの要素で構成される．以下で説明する．

- Facade
  サービスクライアントに対して，業務機能を提供するインタフェースを提供する．このインタフェースをサービスと呼ぶ．Facade は業務処理を司る Business を一つまたは複数呼び出す．インタフェース固有の処理が必要な場合は Facade に実装し，Business をインタフェース固有の要件から独立させる．
- Business
  オブジェクト指向設計にて抽出された特定の業務エンティティの業務操作を目的とするコンポーネントであり，コアな業務処理を担当する．
  また自身が業務処理で利用する複数個のデータベーステーブルの管理責任を負っており，そのテーブル群を更新する唯一のコンポーネントである（図 3）．これは，Business の責務を明確化し，保守性を維持することを目的としている．

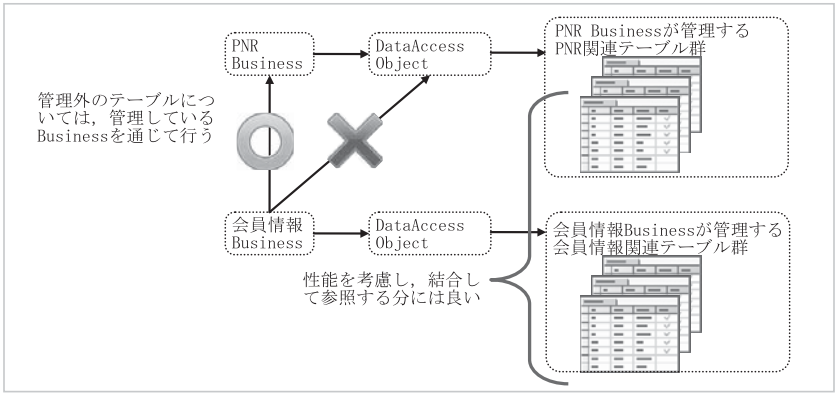

図 3　Business のデータアクセスの例

- Data Access Object（DAO）
  Business から利用され，データベースに対して SQL を発行し，結果を返却するコンポーネントである．Table と View へのアクセスを行う DAO は，開発プロセスに連携したツールで自動生成される．自動生成で実現できない SQL を発行するケースでのみ個別の実装が必要となる．
- Module Facade（MF）
  他モジュール向けに，業務機能を提供するためのインタフェースである．他モジュールの Business や Facade から呼び出される．Facade 同様，このインタフェースもサービスと呼ぶ．Facade 同様に自モジュールの Business をインタフェース要件から守ることだけでなく，自モジュールスキーマの独立性を保つのが目的である．
- Cache Holder
  性能向上を目的として，マスタデータなど参照は多いが更新がほとんどないデータベース上のデータをメモリ上で保持するコンポーネントである．Business から利用され，データベースアクセスなしでテーブルデータが参照できる．

各モジュールの Cache Holder は，キャッシュフレームワークの管理下にあり，キャッシュ内容の参照や定期的あるいは任意のタイミングでのデータリフレッシュ機能を提供している．

■ モジュールスキーマ

それぞれのモジュールは専用のデータベーススキーマを持つ．このスキーマ内に自身の担当する業務領域のテーブルを全て持ち，そのテーブルのメンテナンス責務を負う．他のモジュールが管理するテーブルへのアクセスは，基本的に MF が提供するインタフェースを通じたアクセスに限定し，モジュールスキーマの独立性を保つ．

3. 3. 2　サービスのフレームワーク

サービスに配置されているフレームワークモジュールは，複数ある ANACore のフレームワークのコアのフレームワークである．フレームワークモジュールを支えるベースプロダクト上に，システム全体向けの機能と，サービス層の他のモジュール向けの機能を持つ（図 4）．

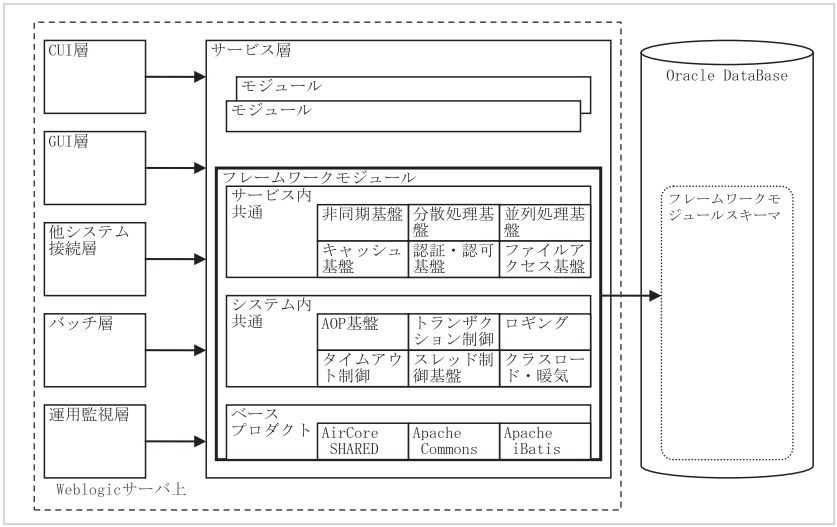

図 4　フレームワークモジュールの構成

■ ベースプロダクト

フレームワークモジュールを支えるベースプロダクトとして，AirCore の SHARED モジュールをコアに利用している．また，永続化機能として自動生成機能を持つ Apache iBatis を利用し，共通部品として Apache Commons を利用している．

■ システム内共通

アプリケーション処理の流れの基盤となる主要機能（表 1）と，サービス層およびサービスクライアント共通のユーティリティを持つ．

表 1　システム内共通の主要機能

機能名	説明
AOP基盤	業務の実装に影響を与えずにフレームワーク処理を挟み込めるよう，AOPを採用している．AOPの実現には既存フレームワークプロダクトを利用せず，独自に実装を行っている．
トランザクション制御	EJBコンテナ管理のトランザクション処理を採用している．トランザクション制御が必要なタイミングでEJBに処理を通すなどのコントロールを行う．
ロギング	ログ基盤として，Java.util.loggingベースのログ実装を様々に拡張している．処理の開始，終了やSQL，障害時の解析に必要な情報のロギングを行う．ログの出力レベルについては動的に変更することができ，特定の端末からの処理だけログレベルを変更することなどが実施可能．
タイムアウト制御	機能毎にタイムアウト値を設定することで，Java処理およびSQL処理で時間がかかりすぎシステムに影響を及ぼすことを最小限にとどめる．
スレッド制御	JavaEEでは実現できない，動作しているスレッドを強制的に停止する機能を実装している．また，業務上の優先度の低い機能のリクエストによって，優先度の高い機能の処理に影響を及ぼさないよう，機能のグループごとに同時実行数を設定し，その数以上同時に動かないようにコントロールを行う．
クラスロード・暖気	Javaアプリケーションは，クラスロードに時間がかかるため初回処理に時間がかかる．ANACoreフレームワークでは，主要クラスを初期化時にクラスロードさせる機能と，さらに参照系の主要機能を暖気と称して事前実行させる機能を持つ．

■　サービス層内共通

サービス層内の業務処理が利用する機能を提供する（表 2）．

表 2　サービス層内共通の主要機能

機能名	説明
非同期基盤	業務処理を非同期で実行させるフレームワーク．システムとして直列化して実行することや処理の異常終了時はリトライさせることも可能．またキューイングされる処理は，メンテナンス画面にて，停止や削除することも可能．
分散処理基盤	長時間かかるバッチ処理等を 1 台のサーバではなく，複数のサーバで分散並行処理させるためのフレームワーク．
並列処理基盤	処理の重いオンライン処理においても，高速に処理できるよう，1 台のサーバ上の複数スレッドで並列処理させるフレームワーク．
キャッシュ基盤	各モジュールに配置されているCache Holderを管理し，ブラウザ画面を利用したキャッシュ内容の参照，キャッシュの指定間隔での自動リフレッシュ，ブラウザ画面からの全サーバ，特定サーバを指定した強制的なリフレッシュ処理などを実現する．
認証・認可基盤	システムユーザの認証と，利用可能端末，利用可能機能の認可機能を持つ．またサインイン時はセッションの管理機能を持つ．
ファイルアクセス基盤	固定長バイナリファイル，固定長テキストファイル，セパレータ区切りテキストファイルの読み書きを支援する機能を提供する．

3. 4　CUI 層のアーキテクチャとフレームワーク

CUI 端末の処理をハンドリングする CUI 層について，アーキテクチャとフレームワークを説明する．

3. 4. 1　CUI 層のアーキテクチャ

CUI 層は，メインフレームのエミュレータ端末や空港設置の端末から入力されるキャラクタをハンドリングし，リクエストされた機能に該当するサービス（Facade）を呼び出し，返却された結果を受けてキャラクタベースの画面を構築し，端末に返却する．CUI 層は図 5 に示す要素で構成される．以下に説明する．

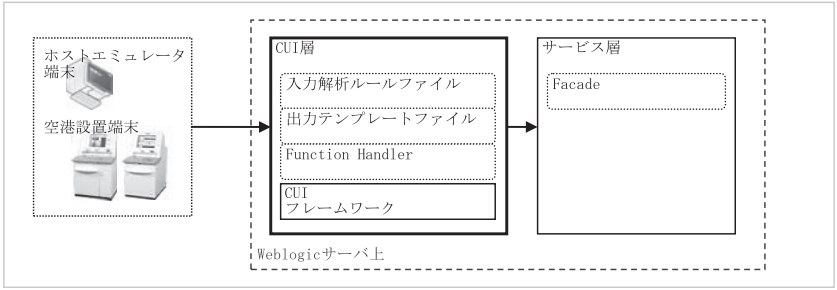

図5　CUI層の構成

- 入力解析ルールファイル
  入力されたキャラクタ情報を，ファンクションと呼ぶ機能の識別子と，そのパラメータに分解する時に参照する解析ルールを格納している．Excelベースの設計書から自動生成する．ファンクションとその入力パラメータは，任意項目を省略するなど様々な入力パターンを持つ．これらのパターンをJavaベースの実装ではなく，外部ファイル化し，さらに設計書から自動生成できるようにすることで，品質及び生産性の向上につなげている．
- 出力テンプレートファイル
  CUI画面に表示するレイアウトを定義しているテンプレートファイルである．Excelベースの設計書から自動生成する．画面上業務データが入る部分は，変数項目として管理し，それ以外の固定部分はレイアウト情報としてテンプレート中に保持している．
  入力解析ルールファイル同様，出力処理はJavaで実装するのではなく，Excelで視覚的にレイアウトする構造とし，品質及び生産性の向上につなげている．
- Function Handler
  入力解析ルールファイルの情報により，論理項目に分解された入力値を基に，サービス（Facade）を呼び出す．サービス呼び出しの結果を基に出力テンプレートを選択して，変数項目を編集し出力画面を生成する．

3.4.2　CUI層のフレームワーク

CUI層のフレームワークは，入力解析と出力処理に特化した機能を持ち，Function Handlerの実装を支援する．具体的には表3に示す機能を持つ．

表3　CUIフレームワークの主要機能

機能名	説明
入力パラメータ解析・フォーマット検証	キャラクタ入力された文字列に，入力解析ルールファイルを適用し，業務的な項目として，Function Handlerが取得可能とする． 全ての入力解析ルールファイルにない文字列については，入力フォーマットに沿っていない旨のメッセージを画面に出力する．
項番変換・日付変換	表形式で出力された画面では，行を特定する項番を指定した入力が可能である．そのようなケースでは，指定した項番が指し示す情報をFunction Handlerが取得できるようCUIフレームワークにて，業務情報に変換している． また日付などは本日の日付を示す識別子を利用することで入力者の負担を減らす考慮がなされている．そのようなケースでもFunction Handlerとしては，識別子ではなく日付として取得できるよう変換処理を行う．
出力テンプレート処理	画面出力内容について，出力テンプレートを選択し変数部分を渡すことで，出力内容を組み立てる．
ページング処理	出力内容が複数の行になる場合は，自動的にページに分割し，画面からの次ページ表示指示を受けて，次のページを画面表示する．
入力履歴管理・入力補完	画面からの入力を履歴管理している．機能によっては，その機能の前に叩かれたパラメータを省略することが可能である．そのような機能については，管理している履歴情報からパラメータを補完し，Function Handlerに情報を渡す．
ホストエミュレータ電文処理	CUI端末はM345というホストエミュレータ用電文形式を扱う．また文字コードもホスト固有のLETS-J（レッツ・ジェイ）という形式である．CUIフレームワークでは，画面からの入力および画面への出力電文を変換する機能を持つ．

3.5　GUI層のアーキテクチャとフレームワーク

GUI操作に利用されるブラウザの画面入出力をハンドリングするGUI層について，アーキテクチャとフレームワークを説明する．

3.5.1　GUI層のアーキテクチャ

GUI層は，ブラウザベースのGUI機能を提供する．ブラウザからリクエストされた機能に該当するサービス（Facade）を呼び出し，返却された結果を受けてブラウザ画面を構築し，端末に返却する．GUI層は図6に示す要素で構成される．以下に説明する．

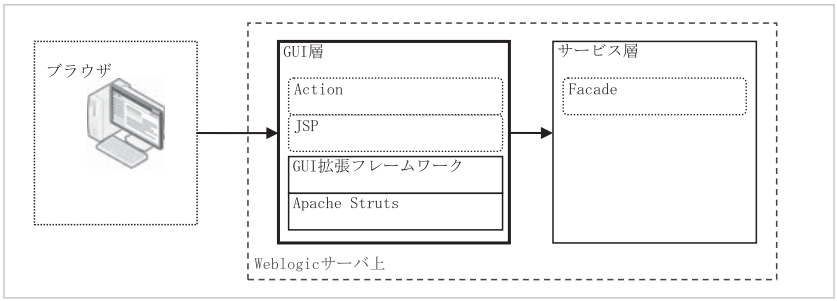

図6　GUI層の構成

- Action
  ブラウザ画面から入力された項目を検証したうえで，サービス（Facade）を呼び出す．サービス呼び出しの結果をハンドリングし，利用するJSPを選択して，画面出力操作を実施する．
- JSP
  JavaEEで定義されるブラウザ画面に表示するHTMLのテンプレートを持つソースコードである．Actionより渡されたパラメータをJSP中で変数部に設定する処理を実装する．

3.5.2　GUI層のフレームワーク

GUI層のフレームワークは，Javaの Webシステム開発では一般的に利用されている Apache Strutsをベースとして，生産性向上の観点でフレームワークを独自に拡張している．具体的には表4の機能を持つ．

表4　GUIフレームワークの主要機能

機能名	説明
入力チェック支援	入力チェック部品の提供および，チェックの結果NGとなった場合，入力値をコーディングレスで再表示したうえで，画面のNG項目を赤色にし，メッセージを表示する
HttpSession制御	HttpSessionのライフサイクル管理を行う．OutOfMemoryの原因となり得る不要オブジェクトの自動破棄も行う．
ファイルアップロード・ダウンロード	ファイルのアップロード・ダウンロード時，ファイルのハンドリングを支援する．
画面表示支援	画面標準に従う標準レイアウトや，画面項目出力部品を提供する．

3.6　他システム接続層のアーキテクチャとフレームワーク

外部システム接続の電文をハンドリングする他システム接続層について，アーキテクチャとフレームワークを説明する．

3.6.1　他システム接続層のアーキテクチャ

他システム接続層は，外部システムとの接続と，バイナリを含めた多種多様な電文形式をJavaプログラム中で同じように操作可能とするための電文・Javaの相互変換，外部システムの接続状態の管理等を行う．他システム接続層は図7に示す要素で構成される．以下に説明する．

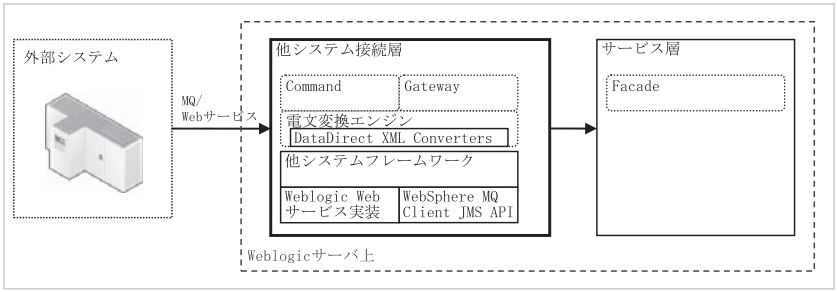

図7　他システム接続層の構成

- 電文変換エンジン
  外部システムとやりとりするバイナリやテキストの様々な電文フォーマットに対して，ANACore内で統一的に操作できるよう，Javaのオブジェクトに相互変換を行う．IATA定義のEDIFACT電文については，DataDirect社の XML Converters というプロダクトを使って変換している．プロダクトを更新することで，EDIFACT仕様の更新の取り込みが可能である．
- Command
  外部システムからの入力電文を受信すると呼び出され，電文で受信したパラメータを基にサービスを呼び出し，結果を再び外部システム側に返却するコンポーネントである．相手システムによっては，電文通番を利用した電文の抜けの確認なども行う．

■ Gateway

ANACore より外部システムに対して問い合わせをする際に，それを中継するコンポーネントである．結果が受信できない場合などのタイムアウトの制御や，一度タイムアウトしたシステムへの送信を取りやめる等の制御も行う．

3.6.2　他システム接続層のフレームワーク

他システム接続層のフレームワークは，MQ，Web サービスを用いた外部システムとの物理的接続と，外部システムのステータス監視などの支援機能を持つ（表 5）．

表 5　他システム接続層のフレームワークの主要機能

機能名	説明
通信機能	MQおよびWebサービスを利用した通信機能を持つ．通信機能として具体的に以下を実装している． ・通信経路の監視 ・電文の送受信 ・通信経路フェイルオーバ，フェイルバック ・問い合わせ応答同期 ※問い合わせ応答同期：MQは一方向の通信であるため，問い合わせに対しての回答を受け取れないことにより，アプリケーションが複雑になってしまう．問い合わせ応答同期機能は，他システムフレームワークがANACoreから外部システムへの問い合わせとその回答を，同期に変換する仕組みであり，一方向を意識せずにアプリケーションを実装することが可能となる．
タイムアウト制御	ANACoreからの問い合わせを行う通信にて，外部システムからの回答待ち時間の制御を行う．
閉塞／スタブ呼び出し	ANACoreから問い合わせを行うシステムに対しては，テスト中むやみに飛ばさないよう閉塞やスタブプログラムを呼び出す機能を持つ．
外部システムステータス管理	Command，Gateway向けに，外部システムのステータスを管理するAPIを提供する．
通番管理	外部システムによっては，電文の抜けや重複を制御する目的で，電文中に通番を埋め込む場合がある．通番はCommand，Gatewayで付与しており，フレームワークは通番の管理APIを提供する．

3.7　バッチ層のアーキテクチャとフレームワーク

JOB 管理サーバ起動されるジョブネットを処理するバッチ層について，アーキテクチャとフレームワークを説明する．

3.7.1　バッチ層のアーキテクチャ

ANACore のバッチは，ジョブ管理サーバからのジョブ実行指示を受けて起動するシェルスクリプトと，業務処理を行う Weblogic 上のサービス（Facade）で構成される．バッチ層はシェルスクリプトとサービスを呼び出す仕組みをさす．バッチ層は図 8 に示す要素で構成される．以下に説明する．

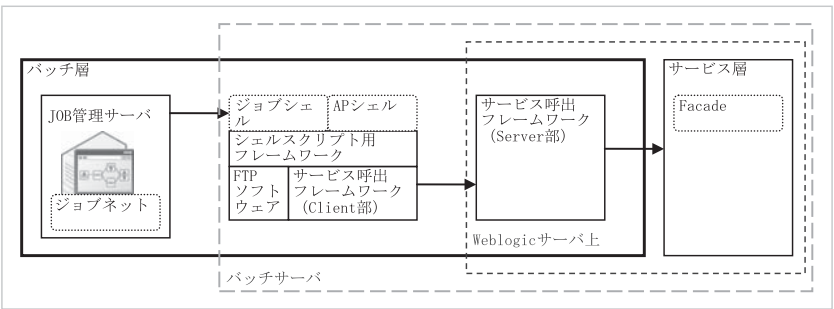

図 8　バッチ層の構成

■ ジョブネット

ジョブ管理サーバ上に実装されるバッチジョブのフローである．ANACore では，バッチの機能毎にネストジョブネットを作成し，ネストジョブネットの最上位のジョブネットで処理の流れを制御する（図 9）．またバッチを以下に分類し，それぞれジョブネットの構成テンプレートを用いてジョブネットをデザインしている．

・定時バッチ：定められた時刻に起動する
・イベント起動バッチ：FTP ファイル受信を契機に起動する
・即時バッチ：オペレータが手動で起動する
・リクエストバッチ：CUI 等のオンライン処理からオンデマンドで起動される
・常駐バッチ：5 分間隔など一定時間間隔で起動する

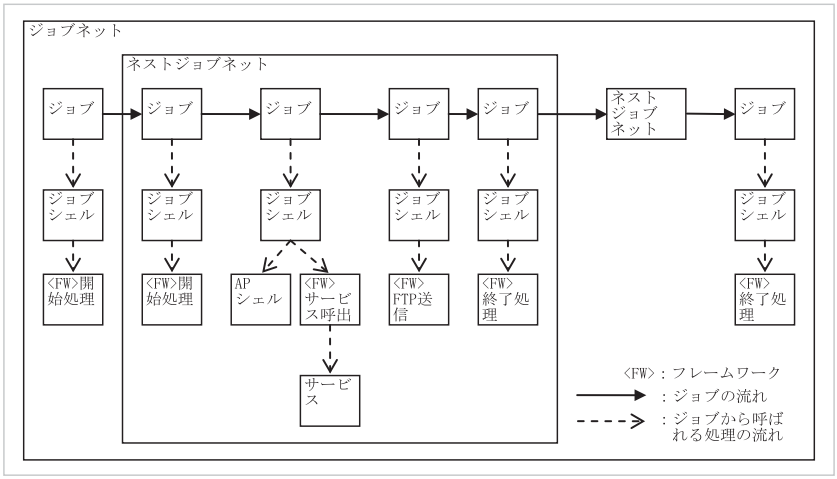

図 9　バッチ層の構成物の関係

■ ジョブシェル

ジョブ毎に用意するシェルである．ANACore では，ジョブが処理する内容を 10 種類程度にメニュー化しており，設計書にてジョブが実施する処理をメニューから処理順に一つ以上選択することで，そのジョブが実施すべき作業を全て実装したジョブシェルが自動生成できる．自動生成することで，ジョブシェルの品質向上を図るとともに，ログ出力等の定型処理についてはアプリケーション開発者に意識させることなく実施できる．ジョブシェルが選択できる主要処理は以下である．

・ジョブネット開始宣言：ジョブネット開始時に必ず入れるジョブで呼ぶ処理
・排他開始：同時に動いてはいけないバッチ処理やオンライン処理がある場合に，開始可能かのチェックと可能であれば排他情報を登録する処理
・排他終了：取得した排他情報を削除する．削除することで他の処理が動作可能になるように呼び出す処理
・サービス呼び出し：サービスを呼び出すジョブで呼ぶ処理
・FTP 送信：FTP 送信を行うジョブで呼ぶ処理
・AP シェル呼び出し：メニューで実現できないシェルスクリプトを用いた処理を実施した場合に呼び出す AP シェルを呼び出す処理

・ジョブネット終了宣言：ジョブネット終了時に必ず入れるジョブで呼ぶ処理

■ AP シェル

メニューを選択することで作成するジョブシェルでは実現できない処理を行う場合に作成するシェルスクリプトである．ジョブシェルのメニューで呼び出す AP シェルを指定することにより，自動生成したジョブシェルに AP シェルの呼び出し処理が実装され，呼び出される．

3.7.2　バッチ層のフレームワーク

バッチ層のフレームワークは，ジョブシェル，AP シェルの処理を補助するシェルスクリプトと，サービスの呼び出しを支援する Java 部分とに分かれる．具体的には表 6 の機能を持つ．

表 6　バッチ層のフレームワークの主要機能

機能名	説明
FTP送信支援	FTP送信ソフトウェアを拡張し，送信前に文字コード・改行コードの変換，圧縮を行う．
ロギング	シェルスクリプト上の処理でも，サービス側のログファイルにログが出力できる． またジョブネット，ジョブは開始終了時にログ出力を行う．これらバッチの進捗と，バッチの処理件数を出力する専用のログファイルを持つ．
オンデマンドバッチ	CUI等のオンライン処理からジョブネットを起動する仕組みを提供する．
排他機能	バッチとバッチ，バッチとオンラインで同時に実行させないよう排他の仕組みを持つ．
単純リラン情報管理	ANACoreのバッチ設計は，基本的に単純リランで設計することが求められる．そのため，処理済みの情報を残すための支援機能を提供する．

3.8　運用監視層のアーキテクチャとフレームワーク

運用監視向けの API を提供する運用監視層について，アーキテクチャとフレームワークを説明する．

3.8.1　運用監視層のアーキテクチャ

運用監視層は，システム運用機能向けに，ANACore の状態の返却や変更を行う機能を提供する．JMX（Java Management eXtensions）を用いた API を運用管理サーバ上のクライアントに対して提供し，クライアントのリクエストを受けてサービス層で管理している情報を返却する．この層は全てフレームワークで構成されており，また呼び出すサービスもフレームワークモジュールに属するサービスを呼び出す．運用監視層は図 10 に示す要素で構成される．以下に説明する．

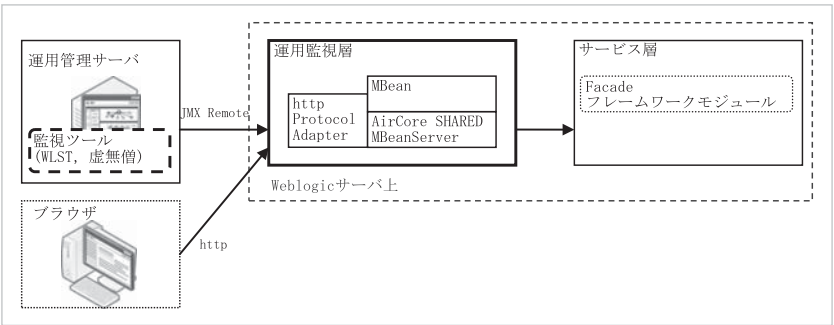

図 10　運用監視層の構成

- MBean（Managed Bean）
  JMX で定義されるクライアントに対して，運用管理機能を提供するコンポーネントである．管理内容毎に MBean を作成し，クライアントは MBean が提供する API をリモートで呼び出すことができる．MBean では，クライアントの処理を受け付け，フレームワークモジュールが提供するサービスを呼び出し，結果をクライアントに返却する．
- AirCore SHARED MBeanServer
  MBean を管理する MBean Server を AirCore のフレームワークである SHARED モジュールが提供している．MBean は，サーバ起動時に，MBean Server への登録処理を行うことで，クライアントからの呼び出しが可能となる．
- http Protocol Adapter
  MBean へ，ブラウザでアクセスするアダプタコンポーネントである．
- 監視ツール
  監視ツールは，外部ツールであり，運用監視層には属していない．WLST（WebLogic Scripting Tool）や虚無僧という JMX クライアントツールにて，運用監視層に接続する．

3. 8. 2　運用監視層のフレームワーク

運用監視層は，全てフレームワークで構成されている．ここでは MBean として実装されている主要な運用監視機能を紹介する（表 7）．

表 7　運用監視層のフレームワークの主要機能

機能名	説明
オンライン処理流量制御キュー滞留数通知	オンライン処理の流量制御のキューの現在のメッセージ数を返却する．
非同期処理キュー滞留数通知	非同期処理基盤が持つキューの現在のメッセージ数状態を返却する．
平均処理時間通知	ANACoreでは，各サーバのオンライン処理件数と累計処理時間を15秒毎に3時間分蓄積している．このAPIは，最新15秒間のオンライン処理の平均処理時間を全サーバ分通知する．
スループット通知	上記取得情報により，最新15秒間の1秒あたりのオンライン処理件数を通知する．

4.　プロジェクトを終えて

前章でフレームワークが持つ機能を紹介したが，本章では，ANACore の開発と稼働後の保守の経験を踏まえ，今回選択したアーキテクチャと実装したフレームワークについて考察する．

4. 1　アプリケーション開発に関する評価

開発着手前に検討したアーキテクチャを変えることなくシステムの開発を終えることができたという点は，選択したアーキテクチャ自体が正しかったという証明だと考えている．特にアーキテクチャが規定しているサービス層のモジュールと，サービスクライアントの層構造は，複数の開発チームが責任分界点を明確にし，開発を分担することを可能にした．サービスクライアントとサービス間の Facade，サービス層内の業務別モジュール間の Module Facade というインタフェース方式は，チーム別の開発を可能にしただけでなく，各チームの責任範囲を非常に明確にすることができた．

フレームワークについては，開発時のログに AOP を利用して引数や戻りを出力したこと，本番稼働時に利用される機能ではないが，アプリケーションのテスト用に DB のメンテナンス

機能をフレームワーク自身に持たせたことで，外部ツールを使わずともテーブルの参照や更新，CSV でのダウンロードができた点，ランタイムパラメータという動的なパラメータ管理により，環境を再起動せずともログレベルやテストの状況を変更できた点は，アプリケーション開発の生産性を向上させるのに効果があった．また開発プロセスに組み込んだ設計書からのソースコード等の自動生成ツールの利用は，生成されたコードが検証済みの品質であるため，生産性だけでなく，品質の安定化に大きく寄与している．このような仕組みを構築できたのも，アプリケーション固有ロジックをパラメータで定義可能としたフレームワーク機能の成果であると考えている．

4.2　運用保守に関する評価

稼働時の障害に起因した暫定運用や，稼動直後の臨戦体制中のスピードが求められる障害対応では，フレームワーク上に再現した，メインフレームに由来する機能が非常に役に立った．

各 AP サーバの平均処理時間や処理件数をリアルタイムで把握できるようにしたことにより，不安定な AP サーバの把握や，特殊なオペレーションを行う場合などの状況注視が可能となった．また全サーバの動作スレッドでどの機能が動いているかもブラウザ画面で確認できたことにより，意図せず長時間動いている機能を把握した上で動的にスレッドを停止させることができた．

アプリケーションログについても，メインフレームが障害時に出力するダンプを意識した結果，大量のログから障害箇所を探さなくとも障害状況解析ログだけである程度の状況の解析が可能となり，障害解析のスピードアップに繋がった．

4.3　追加開発に関する評価

2013 年 10 月現在，既に ANACore システムを異なるインタフェース層から利用するという追加開発が動き始めている．試験的に実装した結果，ANACore で CUI 向けに開発したサービスが，他システム接続層のみ実装することで CUI 端末以外にもサービスとして提供できることが実証できた．サービス層から独立したサービスクライアント層という構成や，SOA をソフトウェア基盤として採用したことが間違っていないと証明できたと実感している．

5.　お　わ　り　に

ProjectAI のアーキテクチャ検討およびフレームワーク開発の初期段階では，メインフレーム上で動作している able-D が持つ，稼働させたままダイナミックに実施可能な様々な運用機能や管理機能を Java で実装可能か疑わしく，戸惑いながら追加・改修することも多かった．しかし今ではメインフレームとは全く同じものではないものの，Java の特性も活かした良いアーキテクチャと，ある面ではメインフレームを超えたダイナミックさを持つフレームワークを作り上げることができたと考えている．

本稿で紹介したのは主要機能だけであるが，フレームワークは非常に多機能となり，アーキテクチャの検証段階から数えると丸7年，専任チームで開発を行っている．開発途中から上がってくる要件や要望を見込みのある複数の実施可能なアイデアに絞り込み，チーム全員の技術を総動員して実装に落とし込むという作業を地道に繰り返した中で，日本ユニシスは非常に多くのノウハウを蓄積できた．

現在このフレームワークのノウハウを基に，今後の大規模ミッションクリティカルシステムの構築に向けて，日本ユニシスの標準フレームワークである MIDMOST for JavaEE Maia（Maia）の改修が行われている．Maia の実プロジェクトへの適用に際しては，今回の経験を活かし，さらなる生産性の向上と高い性能，品質の実現により，顧客システムの価値向上に貢献していきたい．

＊ 1　ANACore は，米国 Unisys 社のエアラインパッケージ AirCore をベースに開発した，全日本空輸株式会社における新国内線旅客システムの開発コード名である．
＊ 2　ProjectAI は，全日本空輸株式会社の新たな国内線旅客システム“ANACore”の開発プロジェクト名である．
＊ 3　able-D は，全日本空輸株式会社における旧国内線旅客システム（1988 年～ 2013 年 2 月）のシステム名である．
＊ 4　AirCore は，米国 Unisys 社が開発した航空業界向けの各種システムのパッケージである．

参考文献　［1］ 石井 真，阿島 哲夫，（株）カサレアル，「カンタン Struts1.2 改訂版」，秀和システム，2005 年 4 月
［2］ Timothy M. O’Brien，長瀬 嘉秀，株式会社テクノロジックアート，「Jakarta Commons クックブック」，オライリージャパン，2005 年 8 月
［3］ 伊藤忠テクノソリューションズ株式会社，「JP1 によるジョブ管理の実践ノウハウ」，日経 BP 社，2008 年 3 月
［4］ 伊藤忠テクノソリューションズ株式会社，日本オラクル株式会社，「Oracle WebLogic Server 11g 構築・運用ガイド」，翔泳社，2010 年 12 月

執筆者紹介　鮫　島　荘　介（Sousuke Samejima）
2001 年日本ユニシス（株）入社．JavaEE 関連開発の技術支援，開発プロセス策定支援，フレームワーク開発支援に従事．2006 年よりエアライン関連システムのアーキテクチャ策定・フレームワーク開発を担当．現在は公共システム本部 エアラインサービス一部に所属．