公共向けアプリケーションフレームワークの ソフトウェアアーキテクチャの分析

西尾 祐[†]　森脇 康之[†]　渡辺 健一郎[†]　青山 幹雄[§]

[†]富士通コミュニケーション・システムズ(株)

[§]南山大学 数理情報学部 情報通信学科

公共向けアプリケーションフレームワークは，監視制御システム用に開発した各種のコンポーネントを組合せたフレームワーク群である．本システム開発では，システムごとに異なる多様な処理形態を実現する手段としてイベントフローマネージャを用いている．本稿では，このイベントフローマネージャで実現している各種の処理をアーキテクチャの観点で分析する．そして，アーキテクチャの分類を行い，その結果から今後のリエンジニアリングの課題について述べる．

Analyzing Software Architectures of Application Frameworks for Public Service Product Lines

Yu Nishio[†], Yasuyuki Moriwaki[†], Kenichiro Watanabe[†] and Mikio Aoyama[§]

[†]Fujitsu Communication Systems Limited

[§]Dep. of Information and Telecommunication Engineering
Faculty of Mathematical Sciences and Information Engineering, Nanzan University

This article discusses analysis of software architectures of application frameworks for public service product lines. The frameworks are composed with various components for the supervisor and system controls. To flexibly compose different components, an event-based message controller, named Event Flow Manager, is designed. We analyzed various architectural patterns of the behavior of Event Flow Manager. Based on the analysis, we discuss the productivity of framework reuse and future re-engineering of systems.

1. はじめに

公共向けアプリケーションフレームワークは，公共向けシステムをコンポーネントの組合せによって効率的に構築するために開発したフレームワーク群である．公共向けシステムとは，公共団体が所有している各種データや施設を監視制御するシステムおよびシステム構築を行う環境のことで，河川情報システムや道路管理システムなどいくつかの業種(ドメイン)がある．このシステムは，年間数十システムもの需要があり，しかも，要求されるシステムの機能や規模が納入する顧客ごとに異なる．そのため，以前は既存資産を効率良く再利用することが非常に困難であり，開発工程の大半を新規のコンポーネント開発とそれを組合せるシーケンス制御部の開発に費やしていた．

このような背景から，筆者らは 1991 年よりコンポーネントを効率良く組合せることができる独自のミドルウェアプラットフォームを開発し，共通的に再利用可能な機能をコンポーネント化してプラットフォームに組込むことでシステム構築を行ってきた．この開発手法の転換により，システム規模や機能に関係なく，共通的に利用できる各種のコンポーネント群を組合せることによってそれぞれの顧客向けにカスタムメイドするシステム(以後，顧客向けシステムと呼ぶ)を効率良く開発することが可能になった[1].

2. ミドルウェアによるコンポーネント間連携とシステム構築手法

公共向けシステムでは，独自のミドルウェアにより各種のドメインに属するコンポーネントを共通インタフェースの基でプロセス間または他装置間で連携し，さまざまな機能を実現する．各種のコンポーネントは，各々が持つ設定データを基に自己の振舞いを決定し動作する．設定データには，関連する他プロセスのアドレス情報や処理手順，共通データベースの参照情報等の可変要素を持つ．これらのコンポーネントの組合せにより，ドメイン間やコンポーネント間，さらにはコンポーネントの所有するオブジェクト間でそれぞれ連携することができる．公共向けシステムは，これらの組合せを以下のようなアーキテクチャとして分類する．

2.1. 論理アーキテクチャ

図-1 に示すアーキテクチャは，各種のドメインに属するコンポーネントを組合せることにより実現する．公共向けシステムで利用するコンポーネント群は，図に示すような階層構造となっており，業種ドメインごとに図-1 下のような標準的なシステムが構成できる．このような各業種別の標準的な構成を標準フレームワークと呼ぶ．そして業種に応じた処理やサービスを提供するコンポーネントの組合せの構造を論理アーキテクチャと呼ぶ．

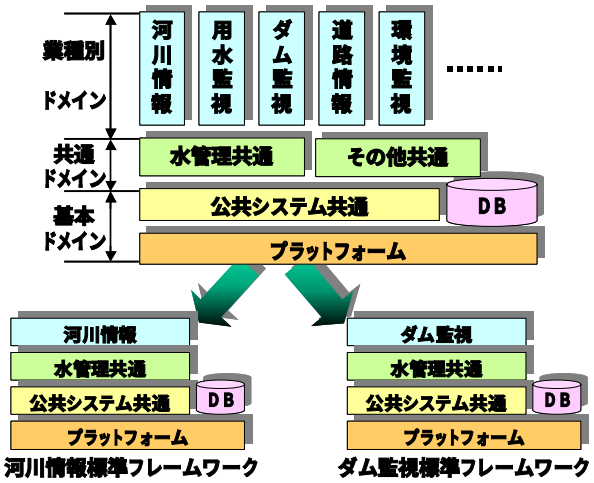


図-1 論理アーキテクチャと標準フレームワーク

2.2. システムアーキテクチャ

図-2 に示すアーキテクチャは，標準フレームワークをベースに顧客向けシステム用にカスタマイズすることで実現する．顧客向けシステムでは，標準フレームワーク以外にも各種の個別サービスや処理機能を持つ[2]．このようにカスタマイズされたシステムの物理的構造をシステムアーキテクチャと呼ぶ．

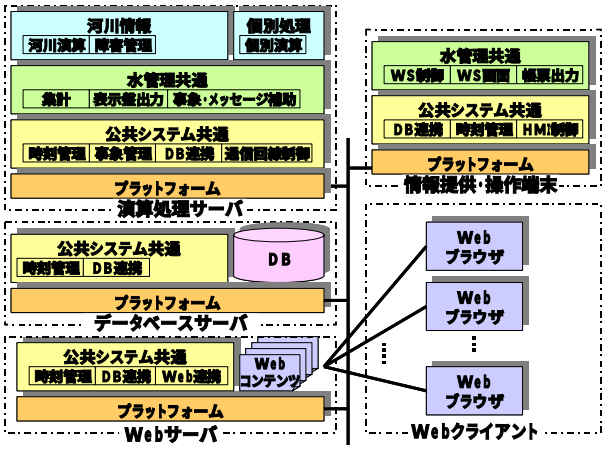


図-2 システムアーキテクチャ例
－河川情報システム－

2.3. 実行アーキテクチャ

ある一連の機能を実現する場合，図-3 のように必要な機能を所有する複数のコンポーネントどうし，あるいはそれらに属するオブジェクトどうしを実行単位で結合する．このようにして目的の機能を実現する実行制御の構造を実行アーキテクチャと呼ぶ．公共向けシステムでは，顧客の要求する機能や規模に関らず，多様な実行アーキテクチャの組合せにより要求される機能を実現する．

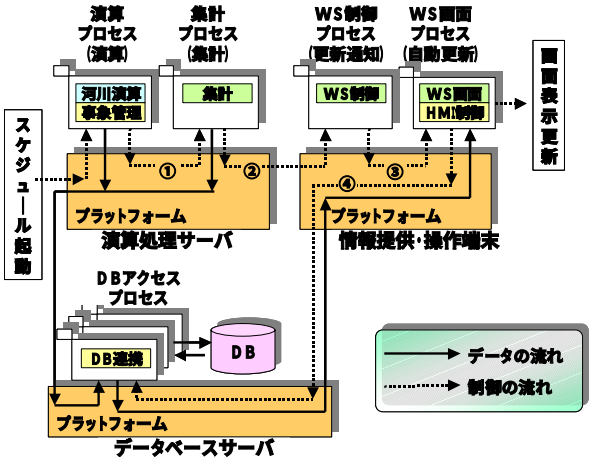


図-3 実行アーキテクチャ例

顧客向けシステムの開発では，コンポーネントの選定･設定データの作成を行い，図-3 のような実行単位で常駐／非常駐のプロセスを作成し，プラットフォームにコンポーネントを組み込む．各コンポーネントの持つ設定データは，関連するプロセスのアドレスや処理手順を定義する．これによりコンポーネントの多彩な組合せが可能となり，結果として多種多様の規模や機能を持つ顧客向け

システムが実現可能となる.

しかし, 各々のコンポーネントが関連する実行プロセスに対して直接的に連携した場合, システムとして要求される機能が増大し, かつ, 複雑になると共に, 各コンポーネントの持つ設定データも複雑になってくる. こうなると, 設定データの作成だけでも膨大な作業量となり, フレームワークの実現も困難となる. さらに, 開発中に仕様変更や機能追加が発生することは日常的であり, 既存コンポーネントの機能追加や関連するコンポーネントへのインタフェースの変更等は頻繁に発生し得る. この結果, わずかな機能追加や仕様変更がシステムに対して大きな影響を及ぼす. よって, プラットフォーム以外にも関連するコンポーネント間の連携を各コンポーネントの機能や動作と独立して制御するための手法がシステム機能として必要となる.

このようなことから, 筆者らはコンポーネント間の連携制御を行うイベント駆動型のフレームワークとそれを制御するイベントフローマネージャと呼ぶコンポーネントを開発し, 顧客向けシステム開発に適用してきた.

本稿では, このフレームワークのアーキテクチャを分析した結果を紹介する.

3. イベントフローマネージャを用いたコンポーネント結合

従来の部品化技術では, ライブラリやモジュールを利用するプログラムを作成することにより部品を結合してきた. 近年のオブジェクト指向のプログラムにおいても, クラスのパッケージ化や継承により, 既存クラスに手を加えなくとも新規機能を既存機能に追加し, そのまま再利用することができるようになった.

しかし, クラスを再利用する際には, 何らかのプログラムの作成が必要であり, 機能の追加や変更が生じた場合, プログラムの変更や改造は常に付きまとってくる. このことは, コンポーネントの実体(プログラム)の単位でも起こりうることであるが, 特に, システム単位での変更要求は 1 つの変更や機能追加が複数のコンポーネントに影響を及ぼすことがある.

このような背景から, 公共向けシステムでは, 連携するコンポーネント間を仲介して制御する機能としてイベントフローマネージャを開発し, 柔軟なコンポーネント間連携を実現した. 従来のコンポーネント間連携とイベントフローマネージャによるコンポーネント間連携は図-4 のように異なる.

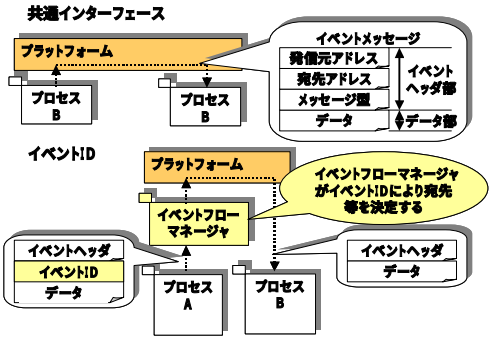


図-4 共通インタフェースとイベント ID

図-4 a)は公共向けシステムの共通インタフェースである. イベントヘッダと呼ぶ部分は, イベントメッセージの発信元／宛先アドレスや同期型／非同期型／応答型といったイベントメッセージの型が設定される. データ部は, イベントを発信するコンポーネントが任意に使うことができる領域で, 基本的にイベントを受信するコンポーネントが処理できる型のデータを設定する.

図-4 b)は, イベントフローマネージャを利用する場合のコンポーネント間の基本的な連携方法である. 基本的には, a)に示す方法で各種のコンポーネント間を連携することができるが, この方法だと, コンポーネントどうしが直接結合することになる. そこで筆者らは, 単にコンポーネント間を連携するだけでなく, 後に発生するであろう仕様変更や機能追加に柔軟に対応できるように, 連携するコンポーネント間を仲介し制御する機能としてイベントフローマネージャを用いている. イベントフローマネージャは, データ部先頭のイベント ID と呼ぶ振舞い方(シナリオ)を決定するための番号から自己の設定データで示されるシナリオを割り出し, シナリオ通りの宛先にイベントメッセージを発信する. イベントフローマネージャの設定データには, イベント ID ごとに処理を行うためのシナリオを設定する.

3.1. イベントフローマネージャの機能

イベントフローマネージャは以下のような機能を共有しており, この機能を組合せることにより各種のコンポーネントの結合や連携形態(実行アーキテクチャ)を実現する.

3.1.1. イベントフロー制御機能

イベントフロー機能は, 図-5 のように 1 つ以上のイベントを複数の対象に振り分けたり, 複数のイベントを待合せ, 1 つ以上のイベントを出力したりすることができる.

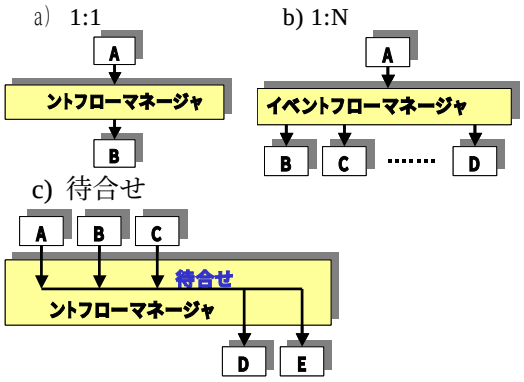


図-5 イベントフロー機能

3.1.2. イベント編集機能

イベント編集機能は，図-6 のように入力された複数のイベントメッセージを編集して出力することができる．この機能により，イベントメッセージ中に付加されたデータを編集して新たなイベントを作成することができる．また，結合するコンポーネント間にインタフェースの違いがあっても，編集することにより差異を吸収できる．

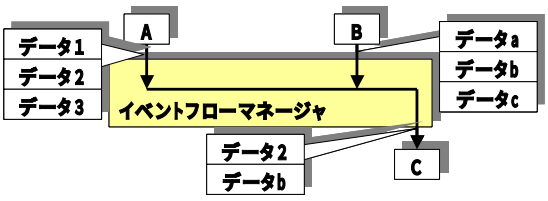


図-6 イベント編集機能

3.1.3. イベントフロー監視機能

一連の機能を実現する場合，いくつかの処理を順に行う必要がある．この処理の流れをシナリオと呼び，イベントフロー監視機能は，シナリオ単位で以下のような処理状態を管理する．

①シナリオの有効
　起動要求により動作可能な状態である．
②シナリオの無効
　起動要求があっても実行しない状態である．
③シナリオの実行中
　一連の処理が実行中である．
④シナリオの停止中
　一連の処理を待合せている．
⑤排他中
　排他指定されている処理が実行中である．
⑥排他による実行待機中
　排他処理の終了を待っている．

また，上記のシナリオの状態に対して，有効化／無効化／強制停止／強制終了／保留等の操作も行うことができる．

上記のような機能は，システムのあらゆる場面で使用されている．

4. アーキテクチャの分析／パターン分類

公共向けシステムでは，イベントフローマネージャによりシナリオを一元管理することができる．この機能により，インタフェースの変更や機能追加が発生しても，シナリオを再編集するだけで既存のコンポーネントに影響を与えることなく変更や追加分を容易にシステムに取り込むことができる．

以下は，公共向けシステムで利用されるコンポーネントの結合／連携形態について Shaw らによるアーキテクチャスタイルの例[3]に基づいて分析し，実行アーキテクチャのパターンとして分類した結果である．

4.1. データフローパターン

データフローパターンは，図-7 のように一連の処理手順をデータの受け渡しまたは参照により逐次行っていく．図-3 に示した例はすべてこのアーキテクチャパターンで実現する．

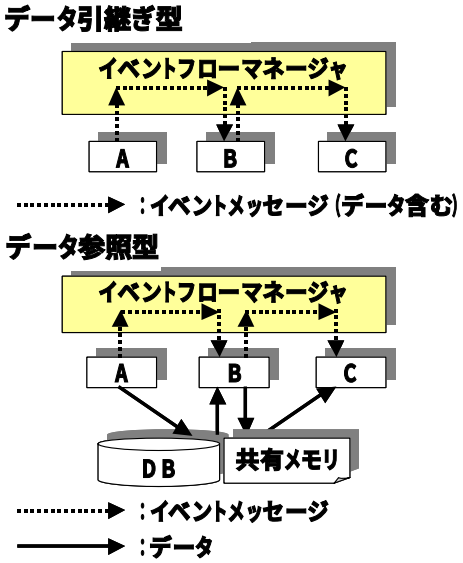
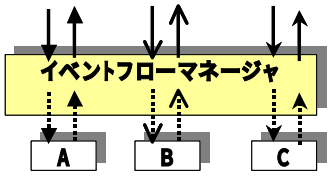


図-7 データフローパターン

4.2. コール･リターンパターン

コール･リターンパターンは，図-8 に示すようにイベントフローマネージャが各コンポーネントの起動制御をイベントごとに行い，イベント発信元に応答を返すことで実現する．また，これは一連のシナリオ単位でも可能であり，起動依頼を受けると一連の処理を行い，その後にシナリオの終了を発信元に返すことで実現する．

a) コンポーネント単位



b) シナリオ単位

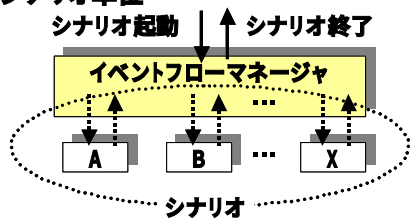


図-8 コール･リターンパターン

4.3. インタープリタパターン

インタープリタパターンは，図-9 に示すように，イベントフローマネージャへイベントを再帰させることによって状態の変化に応じた処理を行うことができる．

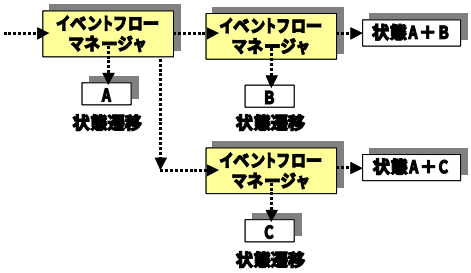


図-9 インタープリタパターン

4.4. イベント駆動パターン

イベント駆動パターンは，イベントフローマネージャが受け取ったイベントにより，対応する宛先に対して非同期でイベントを発信する．これにより，発信元や宛先のコンポーネントは，イベントの個別制御が不要になる．Javaなどのように各コンポーネントが必要なイベント拾って動作する方法と異なり，インタフェースが変更されてもイベントフローマネージャが差異を吸収できる．

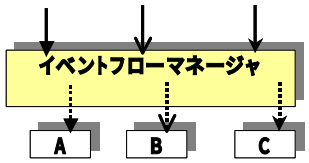


図-10 イベント駆動パターン

4.5. データ共有パターン

データ共有パターンは，図-11 に示すように，処理の制御をイベントフローマネージャが行い，各コンポーネントはＤＢや共有メモリのデータを逐次参照／処理を行う．検索や複数の集計処理を行う場合に有効で，イベントフローマネージャは，データ処理の流れを制御する役目となる．

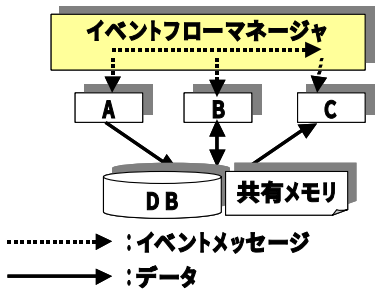


図-11 データ共有パターン

4.6. 階層パターン

階層パターンは，図-12 に示すように，インタープリタパターンを階層構造化したパターンである．イベントフローマネージャは，コンポーネントの状態変化によってイベントの発信先を別の状態変化を行うイベントフローマネージャへ再帰させることで実現する．

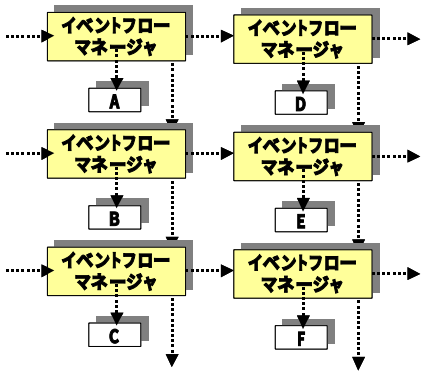


図-12 階層パターン

上記のようにイベントフローマネージャは各種の処理の制御を一貫して行う．これらは処理シーケンスの一例で，顧客の要求する機能の種類により，多様な組合せを実現できる．

5. コンポーネントとフレームワークを用いたシステム開発プロセス

これまで，コンポーネントの組合せ方について述べてきた．次にコンポーネントとフレームワークを用いた顧客向けシステムの開発プロセスについて述べる．

顧客向けシステムの開発は，3つのプロセスに分けられ，次のような流れで開発を行っている．

(1) システム設計と基本フレームワーク構築
(2) 処理フロー(シナリオ)構築

(3) コンポーネント開発

この開発手法では，最初にシステムの基幹となる標準フレームワークの選定を行い，カスタマイズすべき機能単位のフレームワークや新規に開発すべきコンポーネント，およびそれらの結合パターンの設計を行い，その後で機能を実現する処理フローとコンポーネントの開発をおこなう．以下にそれぞれのプロセスについて説明する．

5.1. システム設計と基本フレームワーク構築プロセス

一般的にシステム開発において最初行うことは基幹となるフレームワークの選定である．顧客向けシステムを開発する場合も同様で，既に開発済みである各業種の標準フレームワークの中から顧客要求に最も適するものを選択する．また，2 章で紹介した論理アーキテクチャにより，必要に応じて標準フレームワークに該当ドメインに属するコンポーネントを組込む．ここで用意できたフレームワークを基本フレームワークと呼ぶ．

基本フレームワークの構築では，顧客要求の規模や新規に要求されている機能の実現手段を検討する．具体的には，顧客要求の規模に応じた基本フレームワークのカスタマイズを行い，その後に機能単体の処理フロー(以後シナリオと呼ぶ)や新規に開発するコンポーネントの設計を行う．

5.2. シナリオ構築プロセス

シナリオ構築プロセスでは，まず装置ごとの役割の決定やコンポーネント･プロセスの配置･構成を決定することでシステムアーキテクチャとなる基本フレームワークのカスタマイズを行う．ここでは，要求される規模や機能に対するデータ項目やデータ数，そして分散環境のカスタマイズを行う．その後，基本フレームワーク中の再利用可能なシナリオの選択を行う．再利用可能なシナリオとは，表-1 の適用例に示す機能であり，これらは 4 章で分類したアーキテクチャパターンを実装したものである．

表-1 アーキテクチャパターンの適用例

パターン	適用例
データフロー	演算，集計，スケジュール起動，再演算，データ更新通知
コール･リターン	ＤＢ連携，データ表示，UI
インタープリタ	データ収集，事象管理，排他管理
イベント駆動	UI連携，制御，データ配信
データ中心	演算，集計，検索
階　層	機器制御

さらに，再利用可能なシナリオだけでは実現できない機能のために，新しいシナリオを構築する．具体的には，イベントフローマネージャの設定データの実行アーキテクチャとイベントメッセージの編集を行う．ここで構築したシナリオは，コンポーネントと同様に再利用が可能である．よって頻繁に使用されるシナリオは，標準フレームワークの一機能を実現するフレームワークとしてフィードバックされ，後に開発する顧客向けシステムでは，再利用可能なシナリオとして扱われる．

5.3. コンポーネント開発プロセス

シナリオの構築プロセスで実現できなかった機能は，新規または個別コンポーネントとして開発する．ここでは，新規のコンポーネント開発について述べる．

公共向けシステムで使用しているコンポーネントは，2 章で説明した階層の他に，それぞれの階層が MVC(Model-View-Controller)に分類できる．図-13 は，本システムで使用している主なコンポーネントの分類である．

ドメイン		View	Controller	Model
業種別	河川情報		河川演算 集計処理 障害監視 OLACLE連携	
共通	水管理共通	VB画面作成 HMIアプリ Excel連携 印字記録 外部装置出力作表 音声出力 音声応答	随時DB管理 LAN制御 共通ライブラリ RAS管理補助 集計演算 WS制御	
基本	公共システム共通	HMI制御 帳票出力 汎用Web	業務管理 事象管理 二重化制御 DB管理 時刻管理	通信管理 回線制御
基本	プラットフォーム		コンポーネント制御 バッファ管理 基本ライブラリ RAS管理	

図-13　コンポーネントの階層分類

図のコンポーネント群は，プラットフォームを除き，他と独立して動作することができる．これらのコンポーネントは，他のオブジェクトやコンポーネントとの連携に関して以下のような規約の下で開発されている．

(1) オブジェクトは独立した機能単位とする．
(2) 他のコンポーネントとのインタフェースは，必要なデータをすべて付加する．
(3) コンポーネントと連携する場合は対象のアドレスを可変とする．

(1)はコンポーネントとしての基本であり，(2),(3)に関しては，データの流れや処理フローをイベントフローマネージャのような外部要因に委ねることである．この規約により，コンポーネント開発者

は他のコンポーネントとの関連を意識することなく比較的容易にコンポーネント開発ができる.

以上の開発プロセスは, イベントフローマネージャがコンポーネント間の連携から標準フレームワークの構築, そして顧客向けシステム構築に至るまでを一貫して行えることに起因している. 筆者らは, このイベントフローマネージャによって構築することができるフレームワーク群を公共向けシステムのアプリケーションフレームワークとし, 次章に紹介する顧客向けのシステムを開発している.

6. アプリケーションフレームワークによる顧客向けシステムの構築事例

ここでは, 実際に開発しているアプリケーションフレームワークの例として河川情報システムの構造を紹介する.

図-14 は近年開発した河川情報システムのシステム構成である. 本システムでは, 雨量･水位･潮位･ダム諸量の各種のデータを管理局が収集し, 中央統制局が全管理局分の収集データの演算･集計や水位･潮位の予測等を図にような分散環境で行っている. 収集･演算されたデータは, 図や表などに加工して, 中央統制局の簡易端末やプラズマディスプレイへ情報表示を行っている. また, 外部へは Web コンテンツやデータ配信により, 情報提供を行っている.

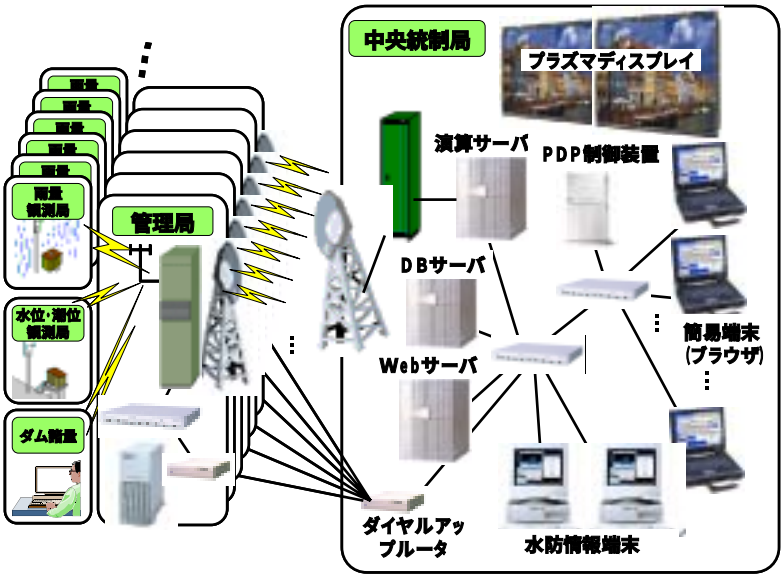


図-14 河川情報システム システム構成

ここでは, 河川情報システムにおける演算サーバのアプリケーションフレームワークを例として, 使用している実行アーキテクチャを解説する.

河川情報システムで使用しているコンポーネントは, 個別処理機能を除くと, 前に説明した図-2 のような構成･配置である. 本システムは, 本稿で解説しているイベントフローマネージャを使用しており, これにより各種の処理フローである実行アーキテクチャを実現している. 図-15 は, データの収集, 演算そして配信までの処理フローを簡略化した例である. 実際の処理フローはもっと複雑であるが, この簡略化した処理フローにも以下のパターンが存在している.

(1) スケジュール起動(イベント駆動)
(2) データ収集(インタープリタ)
(3) 共有メモリアクセス(データ共有)
(4) データ収集待合せ(データフロー)

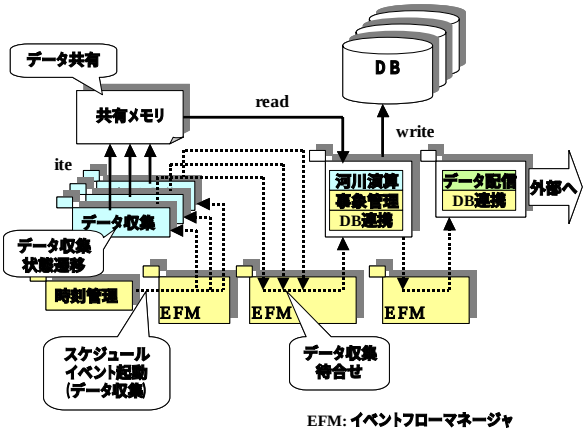


図-15 河川情報システムの実行アーキテクチャ
－ データ収集･演算･配信 －

図-15 のような一連の処理は, 複数のパターンを結合することにより実現している. この例は, 顧客向けシステムに要求される多数の機能中の一つであり, システム全体では, 上記のような処理フローが数多く存在する. よって一連の機能を実現する実行アーキテクチャの組合せをフレームワーク化することが非常に重要であり, 本システム開発では, 5 章で述べた開発手法によりこれを実現している.

7. システム開発の評価

公共向けシステムの開発手法は, イベントフローマネージャを用いることにより, 処理フローのカ

スタマイズや再利用が可能となり，開発効率の向上が実現できた．ここでは，イベントフローマネージャ導入による開発効率の評価を行う．

7.1. アーキテクチャパターンの割合

図-16 は，河川情報システム中の各アーキテクチャパターンの利用率である．システム A，B，C は共に同じ標準フレームワークより構築した顧客向けシステムであるため，使用しているアーキテクチャパターンの割合に大きな違いは無い．よって，アーキテクチャパターンは比較的再利用性が高いことがわかる．

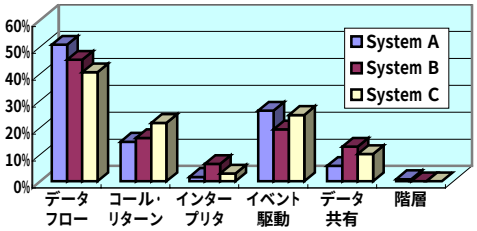


図-16 システムにおけるパターンの割合

7.2. 個別コンポーネントの割合

図-17は，顧客向けシステムの中での個別コンポーネントの割合である．図のようにシステム規模が大きくなるに従い，個別コンポーネントの割合は大きくなる．これは，システム規模が大きくなることでユーザインタフェース等の表示／出力に関する顧客要求が複雑になるためである．システム規模が大きくなっても個別コンポーネントとしての開発規模を小さくするためには，市販の汎用コンポーネントの組込みを可能にするなどの対策が必要であると考えている．

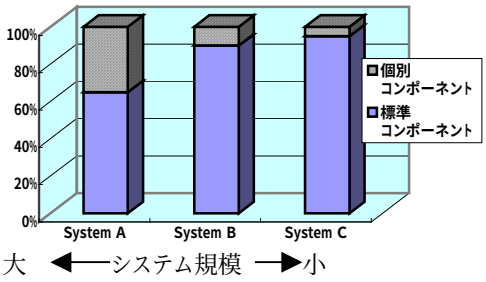


図-17 個別コンポーネントの割合

7.3. 開発工数の比較

図-18 は，1991 年に行った開発手法の転換前と現在の開発工数の変化を示す．BD と ST については，転換前後で大きな変化はなかったが，その他の工程の工数は大幅に削減できており，全体で約60%の工数が削減できた．これにより，5章で述べた開発プロセスが，実績ある既存のフレームワークやコンポーネントを効率良く再利用できていると言える．

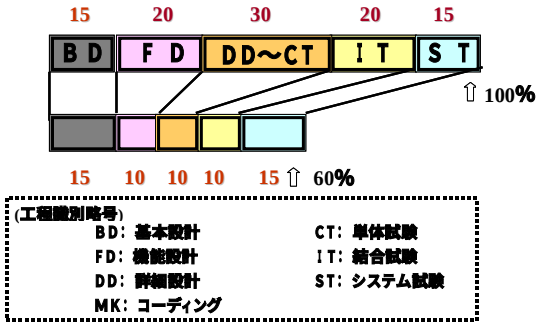


図-18 工程別開発工数の変化

8. 今後の課題

本稿では，公共向けアプリケーションフレームワークのアーキテクチャを分析した．フレームワークを使用する以前と比較し，開発工数が半減できるなどの効果を確認した．

しかし，最近の顧客向けシステムの開発では，顧客要求に対応するために市販の汎用コンポーネントの組み込みが求められている．このような汎用コンポーネントの組み込みを可能にすることが課題である．また，システムのカスタマイズの設定は開発者の手作業によっているので，ツールによる自動化等による支援も課題である．

9. まとめ

本稿では，公共向けシステムにおけるアーキテクチャパターンとそれを利用したフレームワークについて分析してきた．本稿で使用している手法は，手法の転換をしてから既に 10 年になっており，更なる転換に迫られつつあるのが現状である．

よって，本稿で分析したアーキテクチャパターンおよびフレームワークの構築手法を継承しつつ，新しい開発手法を検討している．

謝辞

本研究を実施する機会を頂いた富士通コミュニケーション・システムズ(株)粟路社長並びに，中山開発推進室長に感謝いたします．

参考文献

[1] 青山幹雄，中所武司，向山 博(編)，コンポーネントウェア，共立出版，1998.

[2] 社団法人：ダム・堰施設技術協会，ダム情報処理設備の設計仕様，ダム・堰施設技術協会,1994.

[3] M. Shaw and D. Garlan， Software Architecture, Prentice Hall, 1996.