*C Solution*TM アプリケーションフレームワーク

*C Solution*TM Application Framework

斎藤　悦生　SAITO Etsuo
吉田　和樹　YOSHIDA Kazuki

オブジェクト指向技術はカプセル化や継承などの機能によりソフトウェア部品化技術の本命と目されてきている。事実，数多くのフレームワークと呼ばれるクラスライブラリが登場している。しかし，単にフレームワークを作るだけでは十分な効果が得られないことがわかってきた。近年，“デザインパターン”と呼ばれる技術が注目を集めている。これは，オブジェクト指向技術のさまざまな機能を活用した設計ノウハウ集で，デザインパターンを利用することで再利用性が高まる。*C Solution*TM アプリケーションフレームワーク(APF)は，このデザインパターンを利用したフレームワーク構築のためのコンポーネント体系である。

Object-oriented (OO) technology has several functions such as encapsulation and inheritance which are useful for reusable software systems. Recently, many frameworks have become available. However, these frameworks are not sufficiently efficient for real software development. “Design Patterns” is a catalog of know-hows for OO reusable software design which is currently attracting attention. The *C Solution*TM application framework (APF) uses this to make widely reusable frameworks.

1 まえがき

アプリケーションを効率的に開発するための技術として“ソフトウェア再利用技術”は長年研究されてきた。しかし，期待に反していまだに実効的な技術として確立していないように思われる。一方，近年オブジェクト指向(Object-Oriented：OO)技術のさまざまな機能を積極的に活用した再利用技術として，“デザインパターン[1]”が注目されている。

1.1 部品化・再利用技術は何故失敗したか

部品化・再利用技術は1970年代初期から研究されてきた。当初は，構造化分析・設計手法によりアプリケーションを構築した後，部品として利用度の高いモジュールを切り出すという方法だった。しかし，この方法を実際にアプリケーション開発に適用すると，期待した効果を得るのは難しいことがわかった。技術的な原因として，主に次のような点が挙げられる。

(1) 抽出された部品の流用困難　基本機能は同じであるが，ほんの少しの仕様の差のために使えない。

(2) ベースとなるアーキテクチャが変化　情報技術の進展に伴い，OS，ミドルウェア，言語などの実行環境が変わるため，せっかく作った部品が使えなくなる。

(3) データと機能が分離　構造化手法ではデータと機能が分離されているため，機能とデータ間の依存関係が強く，仕様変更が機能とデータの間で波及しあい，仕様変更に伴う影響が広範囲に及ぶ。

このように，アプリケーションから単にモジュールを切り出すだけでは，広範囲に利用可能な部品(群)は実現できないことがわかってきた。

1.2 オブジェクト指向技術とフレームワーク

OO技術は，カプセル化や継承などの特徴をもち，これを利用すると独立性の高い部品構築や差分開発が可能になるため，部品化の新たな手法として注目を集めた。

OOの機能部品群はクラスライブラリと呼ばれるが，これを発展させて大規模な機能部品体系であるフレームワークが構築されるようになった。特にGUI(Graphical User Interface)ツールとして多くのベンダーから提供されるようになった。これらのフレームワークは継承で利用することが特徴である。しかし，フレームワークはGUIでは効果を発揮することができたが，アプリケーションドメインに対しては必ずしも十分な効果は発揮できなかった。その理由として次の二つがある。

(1) フレームワークの構成要素であるクラス部品の数は普通100以上あり，すべてのクラスを完全に理解して適切に利用することは非常に困難で修得コストが大きい。

(2) フレームワーク利用が継承を前提にしており，機能的に似た部品が継承で数多く作られる。そのため継承のネスト構造が深くなり，上位のクラス構造の変更が下位のクラスへ波及するために，品質に重大な支障をきたす恐れがでてきた。

これらの理由により，OOを導入したのにかえって効率が悪くなった，という事例すら見うけられるようになった。

1.3 デザインパターン

OOによる部品化技術は“デザインパターン”の登場により新たな展開を迎えることになった。デザインパターンは，

OOのもつさまざまな機能(例えば，多様性(polymorphism)やオーバローディングなど)を駆使して，再利用性の高い設計構造を実現するための“設計集(カタログ)”である。デザインパターンをフレームワーク構築に適用することにより，より広範囲に適用できる柔軟に利用可能な部品体系が得られる。われわれはこのデザインパターンに着目してフレームワーク構築を目ざしている。

2 *C Solution*TMAPF

2.1 *C Solution*TMAPFとは

C SolutionTMAPFは，ブラックボックス化されたコンポーネントによってアプリケーションごとに“フレームワークを構築する”ための部品体系である。少数の抽象度の高いコンポーネントにより，広範囲のアプリケーションに対してフレームワーク構築を可能にし，従来のフレームワークの問題点である，構成部品が多くなることによる理解のためのコストの増大，および継承を多用することによる安易な部品の膨大化を防ぐことが可能になる。

2.2 APFコンポーネントとは

APFを構成するコンポーネントは，ある規約(API(Application Program Interface)／実行規約など)を満足させるように作られたクラス(群)であり，ベースコンポーネントとカスタムコンポーネントに大別される。

ベースコンポーネントは，アプリケーションの骨格を作る部品で，ベースコンポーネントどうしの組合せが可能である。カスタムコンポーネントは業種／業務に共通の機能をもち，ベースコンポーネントへ組み込んで利用する。ベースコンポーネントとカスタムコンポーネントを組み合わせて，個々のアプリケーションに必要となる機能構造を得る。これをユーザー定義APFと呼ぶ。コンポーネントは，インタフェースだけ公開されており，内部のクラス構造についてはアプリケーション開発者は関知しなくてよい。

コンポーネントの基本的な内部構造を図1に示す。ベース／カスタムそれぞれのコンポーネントは，実装用クラス(群)と開発用クラス(群)から構成される。

開発用クラス群は，後述するAPF開発環境に組み込まれ，コンポーネントのカスタマイズ用のGUI部品になる。コンポーネントのカスタマイズは，実装クラス群のカスタマイズ用インタフェースを通じてユーザー定義APFのインスタンスの定義を行うと，コード生成機能によってユーザー定義APFとしてコード化される。

実装用クラス群は実際にアプリケーションに組み込まれるクラス群で，Java(注1)言語で生成される。個々のコンポーネントは通例複数のクラスにより構成されるが，コンポーネント外部からは，コンポーネントの主体となるクラスであるImpl(Implimentation)クラスと呼ばれる実装用クラスの一つだけが見える。

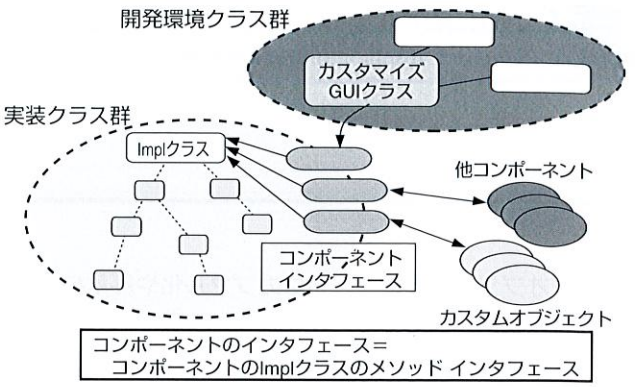

図1．APFコンポーネントの内部構造　ベースコンポーネントとカスタムコンポーネントを組み合わせて個々のアプリケーションに必要となる機能構造を編成する。
Structure of APF components

2.3 APF利用のアプリケーションの構造と開発体制

図2にAPFを利用したアプリケーションの構造を示す。

アプリケーションはユーザー定義APFに，アプリケーション固有の機能を実現するカスタムオブジェクトを追加することで構築される。すなわち，アプリケーションは①ベースコンポーネント，②カスタムコンポーネント，③カスタムオブジェクトから構成される。

APFを利用したアプリケーションの開発において，開発体制はどのように変わるだろうか。従来の開発とのもっとも大きな違いは“アプリケーション開発部隊が2極化”することにある。すなわち，アプリケーションの開発は，APFコンポーネントを開発するAPF開発部隊(以下，開発部隊と略記)と，APFを利用してアプリケーションを開発するAPF適用部隊(以下，適用部隊と略記)とに分かれる。

開発部隊は，コンポーネント開発に加え，適用部隊が使うAPFの選択やカスタムオブジェクト開発の支援を行う。

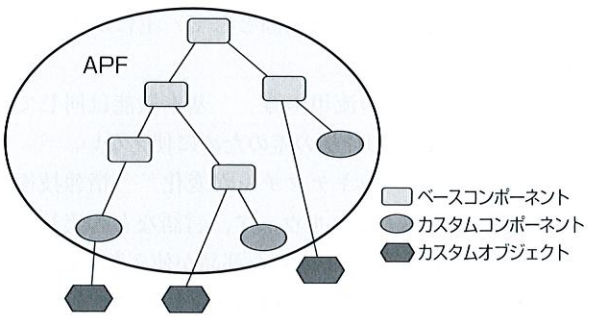

図2．APF利用時のアプリケーション構造　ユーザー定義APFにアプリケーション固有の機能を実現するカスタムオブジェクトを追加することで構築する。
Structure of application using APF

(注1) Javaは，米国SunMicrosystems社の商標。

適用部隊は，開発対象のアプリケーションにフィットするユーザー定義APFを構築し，アプリケーション固有機能をカスタムオブジェクトとして開発する。併せて，コンポーネントの仕様に関するフィードバックを開発部隊へ提供し，より良いコンポーネント開発を支援する。それぞれの部隊は別個に開発を進めるが，開発に当たっては両部隊の共同開発と密接な連携プレイが重要になる。

2.4 APF利用のアプリケーション開発に必要なシステムの構成

図3に，APF利用のアプリケーション開発に必要なシステム構成を示す。

最小限必要となる環境は，ユーザー定義APF開発のためのAPF開発環境とカスタムオブジェクト開発のためのアプリケーション開発環境(Java開発環境)である。

APF開発環境は，ユーザー定義APFを構築するための環境で，コンポーネントリポジトリとカスタマイズGUIとで構成される(図3)。構築したユーザー定義APFからコード生成機能によりJavaコードにすることができる。

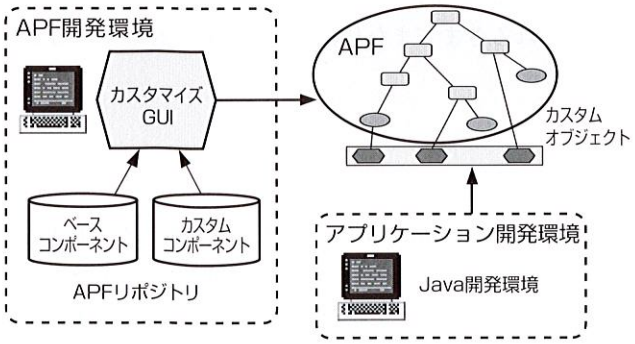

図3. APF利用のアプリケーション開発に必要なシステムの構成
最小限必要となる環境はAPF開発環境とアプリケーション開発環境である。
System configuration of development environments using APF

ユーザー定義APFの構築手順は，ユーザー定義APFを構成するのに必要となるコンポーネントのアイコンをドラッグ＆ドロップで配置し，それぞれのコンポーネントを結線することで基本構造を決定する。個々のコンポーネントのカスタマイズは，コンポーネント側で提供される開発クラス群を呼び出し，そのGUIを通じて行う。ビジュアルにユーザー定義APFが構築できることが大きな特長である。構築したユーザー定義APFはコード生成機能によってJavaコード化される。図4にAPF開発環境の画面例を示す。

Java開発環境では，カスタムオブジェクトをJavaで開発し，読み込んだユーザー定義APFのJavaコードと組み合わせてアプリケーションを構築する。これをJava実行環境で動作させるとアプリケーションが実行される。

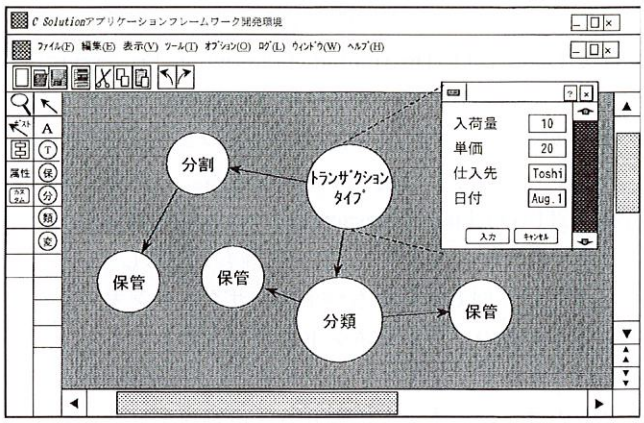

図4. APF開発環境のGUI画面イメージ　アイコンを使ってビジュアルに開発環境を構築する。
Example of APF development environment display

3 APFの実例：Accounts

APFの例としてAccountsを紹介する。

3.1 Accountsの概略構造

Accountsは，“会計業務”などのトランザクション処理を対象としたAPFで，“トランザクションタイプ”“保管”“分類”“分割”“変換”“マージ”の6種類のベースコンポーネントからなる。トランザクションタイプは，業務トランザクションのデータ構造とデータ投入の入り口として機能する。また，分類，分割，変換，マージは再帰的に組み合わせて，最終的には保管でデータを管理する。

Accountsを利用したアプリケーション開発においては，データと処理の流れを“トランザクション”という単位で考える。アプリケーションで処理すべきトランザクションの基本データ構造を定義し，トランザクションの処理方法を，分類，分割，変換，マージの組合せで実現する。

Accountsにおけるトランザクションの流れは次のようになる。トランザクションは“トランザクション オブジェクト”としてトランザクションタイプで生成され，次にツリー状に構成されたトランザクション処理コンポーネントに“ポスト(処理の依頼)”され、そこで次々と振り分けられ，最後に保管コンポーネントで管理される。以下にそれぞれのベースコンポーネントの詳細について述べる。

3.2 Accountsを構成するベースコンポーネント

“トランザクションタイプ”は，業務トランザクションのデータ構造の定義とデータ投入の入り口として機能する。“トランザクションオブジェクト”はここで生成されて，ここを基点にトランザクション処理が開始される。

“保管”は，トランザクションオブジェクトを管理するコンポーネントで，データはRDB(Relational Data Base)に保管される。事務処理系システムにおいては業務は並行処理されることが多いが，そのときのデータの一貫性を保持す

るトランザクション処理機構が重要である。“保管”においてはX/Open DTP(Distributed Transaction Processing)モデルのXAインタフェースに準拠した簡易トランザクション処理機能をもっており，XAのインタフェースをもつリソースマネージャとして使うことができる。したがって，適当なOLTP(On Line Transaction Processing)製品をつなぐことで本格的なトランザクション処理も可能にしている。

“分類”は，同じ構造をもつトランザクションオブジェクトを扱う複数のコンポーネントを子どもとしてもち，ポストされてきたトランザクションオブジェクトの内容を見て，子どものコンポーネントに振り分ける処理を行う。条件分岐に相当する機能を提供する。

“分割”は，異なる構造のトランザクションオブジェクトを扱う複数のコンポーネントを子どもにもち，ポストされてきたトランザクションオブジェクトを子どものコンポーネントへ分割して振り分ける処理を行う。“分類”との違いは，トランザクションオブジェクトの内容を組替え編集することにある。

“変換”は，“分割”の機能をさらに進めて，トランザクションの編集機能をさらに高度化したものであり，具体的にはカスタムコンポーネントで定義された編集加工処理に基づき処理を行う。

“マージ”は“分割”の逆で，複数の“保管”で管理されるトランザクションオブジェクトを取り出し，キーの一致に基づいてトランザクションオブジェクトどうしをマージして，これを下位のコンポーネントへ振り分ける処理をバッチ形式で行う。

3.3 Accountsを利用したアプリケーション例

Accountsを利用してアプリケーションを構築する例として，医療システムの料金計算を対象に構築したプロトタイプについて説明する(図5)。

APFの対象は処置を行った後の費用計算処理で，具体的には，処置対象の患者の病状，処置内容などによりポイント計算を行う。この算出方法は計算処理の内容が大幅に変更されるごとにシステムの改造が必要となり，その対応方法が重要な課題となる。また，患者のステータスが変われば計算式が変わるため，柔軟な対応方式が望まれる。

これをAccountsで実装すると図5のようになる。ユーザー定義APFの構造は，“変換”コンポーネントに処置データからポイントを割り出す計算式を組み込み，さらに計算対象処置と対象外処置ごとに“保管”を用意し，“保管”にはポイントから基本利用料の計算などを行うカスタムコンポーネントを組み込む。

このようにして構成したシステムで，例えば患者のステータスが変わった場合を見る。患者のステータスはある日付を境に変わるが，このとき“保管”コンポーネントはタイムスタンプをベースに計算処理を行うため，患者のステータスを変えるだけで処理が行える。つまり，このための新たな作業は必要ない。同様に法改正対応も，法改正後の規則をタイムスタンプで起動できるため，法改正後の規則用カスタムコンポーネントを新たに組み込むだけでよい。

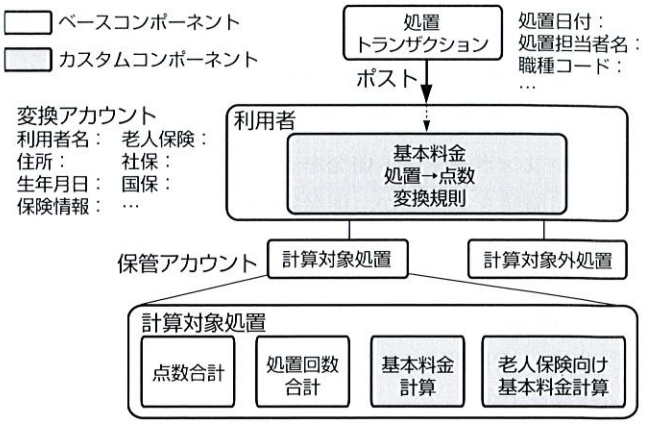

図5．Accountsの適用事例　Accountsを利用した医療システムの料金計算を対象に構築したプロトタイプの構造。
Example of application using Accounts

Accountsによるシステム開発では，このようにシステムに要求されるさまざまな仕様変更に柔軟に対応できることがわかる。

4 あとがき

ここでは，***C Solution***$_{TM}$APFについて述べた。***C Solution***$_{TM}$APFは，デザインパターンを利用したコンポーネントベースのフレームワーク構築技術である。ベースコンポーネントとカスタムコンポーネントの2種類のコンポーネントによって，対象となるアプリケーションにフィットしたフレームワークを柔軟に構築できる。APFの例として，Accountsを紹介した。Accountsでは6種類のコンポーネントによって，即時更新型のトランザクション処理を可能にしている。

文　献

(1) E. Gamma, et al. Design Patterns-Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

斎藤　悦生　SAITO Etsuo
SI技術全社支援センター オブジェクト指向技術支援担当主査。オブジェクト指向技術の全社普及推進およびアプリケーションフレームワークの開発に従事。
SI Technology Support Center

吉田　和樹　YOSHIDA Kazuki
SI技術全社支援センター オブジェクト指向技術支援担当主務。
アプリケーションフレームワークの開発に従事。
SI Technology Support Center