技術レポート

.NET版ワークフローエンジンの開発

産業システム第三事業部
鈴 木 貴 博

1．はじめに

本稿では、ワークフローシステムにおける基本機能である、フロー定義、承認、差し戻し、経路確認などをワークフローエンジンとして、.NETテクノロジーを利用し開発した実際について紹介する。

2．背景

日常の業務には、交通費精算の申請、連絡文書の申請など、特定の伝票や帳票を申請し、特定のルートで承認する定型的な業務が数多く存在する。この定型的な業務を自動的に処理して、業務の効率を向上させるのがワークフローシステムである。

ワークフローシステムは既に一般的であり、LotusNotes[*1]、Microsoft Exchange[*2]等を利用した開発が比較的多く行われている。特に、LotusNotesでのワークフローシステムの構築は、文書データベースという特性上容易であり、多くの導入実績がある。

現在、LotusNotesを導入している企業では、ActiveDirectory[*3]導入によるインフラ統一などさまざまな事情により、Exchangeなどのマイクロソフト製品へ移行するユーザーが多数存在している。

しかし、Exchangeでのワークフローシステムの開発をLotusNotesでの開発と比較した場合、難易度は高くないものの、一般的なカスタムフォームを利用するだけでは、同等の機能の実現が難しい。機能の拡充を図るにはCOM（Component Object Model）[*4]などを組み合わせた、それなりの工数を要する開発を強いられることになっているのが現状である。

もちろん、システムをWebアプリケーション、クライアント/サーバー・アプリケーションとして最初から開発する方法もあるが、開発効率の点でLotusNotesに遠く及ばない。また、パッケージを導入する方法では、業務をパッケージの機能に摺り寄せる必要があり、理想的なシステムの実現という観点からは決して望ましくない。

3．目的

このような背景から、ワークフローにおける基本的な機能を司るエンジン部分を切り出し、さまざまなシステム開発に再利用可能なワークフローエンジン・コンポーネントが必要と考え、その開発を目指したものである。

開発に際しては以下のような目的を置いた。

(1) ワークフローエンジンの開発

本システムでは、拡張に対してオープンなコンポーネントであり、Exchange、ActiveDirectoryとの連携を行え、かつ、Webシステム、クライアント/サーバー、モバイル等のさまざまなシステム形態に対応可能なワークフローエンジンの開発を目的とする。

＊1）ロータス社のグループウェア製品。

＊2）マイクロソフトのグループウェア製品。

＊3）マイクロソフトのディレクトリ技術。

＊4）マイクロソフトが開発した、クライアントとサーバーの両方で利用可能なコンポーネント。

(2) サンプルワークフローシステムの構築

このエンジンを利用したサンプルとして、少なからずニーズのある、オフライン申請が可能なワークフローシステムを実現する。これは、エンジンのWebサービス化と、クライアントとしてのMSOfficeとの連携により実現する。

(3) .NET テクノロジーの利用

拡張に対してオープンなシステムの実現という点から、オブジェクト指向による開発は必須である。また、Exchange、ActiveDirectoryと連携できる技術として、現時点では最適と考えられる .NETテクノロジーを利用し、言語にC#を採用した実装を行う。

さらに、今後の開発をふまえ、.NETを利用した際のオブジェクト指向設計･開発において、基本となるスタイルについて検討を行う。同時に永続化について可能な限りの手法の確立と、必要な各種ライブラリの整備についても、今回の開発の中で行う。

4．システムの概要

4.1 ワークフローエンジンの概要

ワークフローエンジンの機能を表1のように想定し、開発を行った。このワークフローエンジンを利用したシステム構成の3つのパターンを次ページ図1に示す。

4.2 サンプルワークフローシステムの概要

Webサービス化したフローエンジンを利用したサンプルワークフローシステムとして、Excelをクライアントとする、オフラインエントリーが可能なシステムを構築した。これは図1(c)のパターンの実装形であり、より詳細な構成を次ページ図2に示す。

表1　ワークフローエンジンの機能

申請経路定義機能
・経路（各ワークステップの前後関係）に関する定義機能
・線形経路定義機能
・分岐経路定義（並列、条件分岐）
・経路上の各ワークステップに関する設定機能
・各ワークステップでの承認グループ、あるいは、承認ユーザー指定機能
・各ワークステップでの滞留判定条件設定機能
ユーザー・グループ管理機能
・ユーザー定義機能
・グループ定義機能
・グループ･ユーザー関係定義機能
申請機能
・指定した経路定義を元にした申請機能
承認機能
・次ワークステップが一つあるいは複数存在するケースに対応した承認機能
・次ワークステップが複数あり､一つの選択、複数選択､全選択をあらかじめ指定されているケースに対応した承認機能
・次ワークステップでの承認グループが決定しているケースに対応した承認機能
・次ワークステップでの承認ユーザーが決定しているケースに対応した承認機能
差し戻し機能 ・ワークステップとして一つ前に差し戻す機能
・ワークステップとして申請者に差し戻す機能
滞留処理機能
・滞留確認メール機能
代理・自動承認指示機能
・自分が承認ユーザーとして設定されている未承認のワークステップ(先のワークステップ）に対し、代理承認者を設定する機能
・自分が承認ユーザーとして設定されている未承認のワークステップ(先のワークステップ）に対し、自動承認として設定する機能
承認状況確認機能
・申請した特定の申請書の承認状況の画像出力機能
・申請した特定の申請書の一つ先のワークステップを表示する機能
・申請した特定の申請書の、全ての承認、差し戻し、状況を表示する機能
・経路が線形の場合、申請した特定の申請書の先のワークステップ全てを表示する
自分が承認すべき申請書の一覧を表示する機能

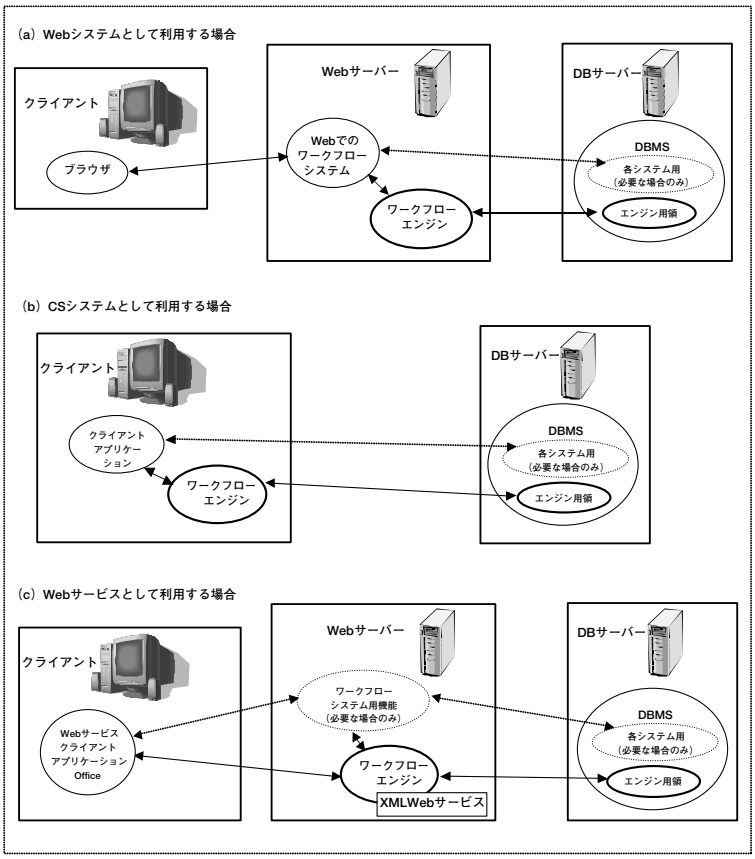

図1　ワークフローエンジンを利用したシステム構成のパターン

5．開発の概要

機能の洗い出しの後、分析フェーズとしてエンティティの抽出を行った。永続化の格納場所としてRDBを前提としていたこともあり、名詞抜き出しなどのオブジェクト指向でのエンティティ抽出手法ではなく、従来のデータ中心的な手法を用い、可能な限り正規化を行った。ORマッピングとしては、基本的に、1エンティティに対し単純に1クラスを割り当てた。

今回の開発は、総勢2名が、2002年12月～2003年4月の5ヵ月間、週に約1日を費やしたプロジェクトであり、アジャイル的な開発*5にならざるを得なかった。そのため、主要な機能に対し、抽出したエンティティと、考えられるバウンダリー、コントロール*6を付加したシーケンス図、およびクラス図を作成した段階で設計フェーズ完了とし、実装に取りかかった。

永続化に関するパターンとしては､エンティティクラスそのものと永続化を司るクラスを分離し､ファクトリメソッドを取り混ぜたパターンを作成して適用した。また、ADO.NETを利用する際のパターンとして､プロバイダの抽象化を行うパターンを作成し､適用した上で実装を行った。

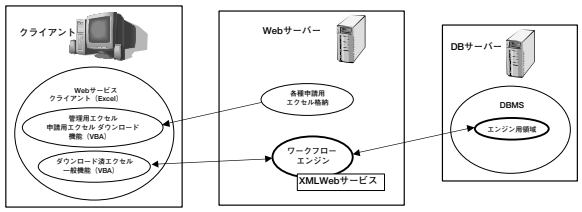

図2　Excelをクライアントとする､オフラインエントリーが可能なシステムの構成

6．開発における考慮点

6.1　ワークフローエンジン開発における考慮点

(1) オブジェクト指向での開発

永続化部分の開発に関しては、パターンの適用が功を奏し、ほとんど手戻りもなく開発を終了することができた。

---

*5)「大量のドキュメントよりも動作するソフトウェアの方が重要」「計画に頑固にしがみつくよりも積極的な変更を受け入れるべきだ」といった、ユーザーの満足度を最大にする開発手法の総称。

*6) ソフトウェアアーキテクチャであるヤコブソンのBCE（Boundary－Control－Entity）モデルにおける、Boundary、Control。

また、ADO.NETプロバイダの抽象化により、RDBを途中から切り替えた際にも、変更点は、変更後のRDB向けの永続化クラスの追加のみとなり、オブジェクト指向での抽象化による効果を得ることができた。

(2) リファクタリング実施

最初の実装直後における永続化以外の処理に関しては、設計どおりの実装ではあるものの、適切なメソッドの配置とは言い切れなかった。そのため、見直しとして、リファクタリング[*7]を行った。リファクタリングの効果は大きく、以降これを繰り返す方法を用いた。結果として、拡張性、再利用性は向上したが、クラスへのメソッドの配置については当初の設計と大きく異なり、かつ、想定しなかったメソッドも多数作成することになった。

(3) オブジェクト指向開発におけるリファクタリング

最終的には理想的な形になったものの、設計段階で理想形に至らなかった要因として、設計フェーズの時間を十分に取れなかったことが最も大きいと考えられる。しかし、それを差し引いても「設計の段階で決して発見できなかったのではないかと考えられる本来あるべきメソッド」が存在したのも事実だ。

個人的な見解だが、オブジェクト指向での開発では、設計フェーズは従来と同じく最も重要な位置を占めるが、可能であれば、リファクタリングについても採り入れるべきではないかと考えるに至った。

とはいえ、今回のような少人数でのアジャイル的な開発とは異なる規模の大きい開発、すなわち、クラスごとに各メンバーが実装を行うような場合には、メソッドのクラス間での移動が頻繁に発生するリファクタリングは、適用が難しくなるという問題もある。

また、たとえリファクタリングの効果が大きいとしても、従来どおり設計として完璧なものを目指し、リファクタリングは考えずに行うべき、という意見もあるだろう。

しかし、いくら精査した完成度の高い設計を行ったとしても、マーチン・ファウラーが著書『リファクタリング－プログラムの体質改善テクニック』[*8]で述べているように、設計段階では隙がある。また、ここからは私見だが、結果として設計段階では完全にはなりえず、たとえそれが可能だとしても、それにかかる工数には見合わないのではないか、というのが、今回の経験から得た結論だ。

厳密な設計は重要だが、同時に、いかに上手くリファクタリングのフェーズを取り入れるか、という点が、今後、ある程度の規模のオブジェクト指向開発における課題になるのではないか。これについては今後も十分に検討していきたい。

(4) エンジンのWebサービス化

.NETを利用したWebサービス化は非常に容易であり、エンジン部分のWebサービス化についてはなんら問題は発生しなかった。

(5) .NETでのグラフィックス機能の効能

これまでは非常に手間のかかったグラフィックス関係機能の取り扱いが、.NETでは格段に容易になった。従来、こうした機能は工数の増大につながるため、費用の問題から採り入れない傾向にあったが、.NETでの開発ではその認識を改める必要があるだろう。今回は、この機能を利用して、経路の確認、承認印などを視覚的に表示する機能を付加した。単なる見栄えだが、わずかな工数で効果は大きく、今後もこうした機能については拡張していく予定だ。

6.2 サンプルワークフローシステムの開発における考慮点

Webサービスを利用するクライアントとして、今回はExcel2002を採用した。また、Webサービスに対応するために、Office XP Web サービスツールキットを採用した。これによりプロキシーの自動生成が可能になり、かつ、かなり適切なプロキシークラスが生成されるため、開発の大部分でWebサービスをそれほど意識する必要がなく、通常のVBAでの開発と同様に行えた。また、書式情報なども含めたデータのXML化の効果は大きく、しかも容易に実現が可能だった。

7．システムの特徴

7.1 ワークフローエンジンの特徴

ワークフローエンジンの特徴を以下にまとめる。

(1) 機能のWebサービス化

今回のエンジンは、一般的なワークフローに必要な機能をひととおり備え、かつ、各機能がWebサービスとしても提供されている点が特徴だ。その結果、通常のライブラリとしての利用はもちろん、Webサービスとしての利用も可能となっている。

(2) 各種RDBへの対応

永続化の際、ADO.NETプロバイダを抽象化して利用することにより、永続化クラスを追加し、各種RDBへの対応を容易にしている。

(3) 承認状況などの画像出力

承認状況などの確認機能では、承認状況等をテキストだけでなく画像として出力することも可能にし、ワークフローシステムにおけるインタフェース開発の負担を削減した。

＊7) 外部から見たときの振る舞いを保ちつつ、理解や修正が簡単になるように、ソフトウェアの内部構造を変化させること。

＊8) マーチン・ファウラー著、児玉 公信、友野 晶夫、平澤 章、梅澤 真史 訳『リファクタリング－プログラムの体質改善テクニック』ピアソン・エデュケーション（2000）。

(4) 申請内容のワークフローエンジンへの保存

申請内容をワークフローエンジン内に格納できるようにした。そのため、稟議書などのように内容に関して集計した結果を必要としないシステムであれば、エンジンの他にRDBなどの永続化資源を必要とせず、ワークフローシステムを構築可能となる。

(5) 拡張性の維持

エンジンの利用者側ではなく開発者側にとっての特徴だが、抽象化を行った実装を意識したため、結果として比較的容易な機能の拡張を可能としている。

7.2 サンプルワークフローシステムの特徴

開発を行ったサンプルワークフローシステムの特徴を以下にまとめる。

(1) クライアントとしてExcelを利用

エントリーに関しては、カットアンドペーストなどの機能を利用し、過去に申請した内容からの貼り付け等を可能としている。

(2) オフラインでの利用

申請用のテンプレートシートをダウンロードしておくことにより、オフラインでの申請入力を可能としている。また、申請途中での保存や申請前のデータの保存についても、Excelの保存機能をそのまま利用できる。

(3) データ表示に関するExcelの書式設定の利用

データ表示に関しては、Excelの書式設定をそのまま利用することにより、条件付書式設定等を使って特異データの強調表示などを可能としている。

(4) 承認済申請データの集計

申請内容をXML化してエンジン内に保管しており、申請データの集計を行う場合には、最終承認済の申請データをエンジン内より抽出すれば、必要なフィールドの集計を行える。

(5) 申請データのファイル出力

エンジン内に保管してある申請データは、表形式でのファイル出力が可能。

8．今後の展望

(1) ワークフローエンジンの拡張

今後のワークフローエンジンの機能拡張として、ActiveDirectory、Exchange等との密接な連携を予定している。その際、エンジン開発における目的の一つだった「拡張に対してオープンなコンポーネントとする」との点について、より詳細な達成度の確認を行っていきたい。

(2) .NETでの設計及び開発スタイル

マイクロソフトの製品を利用した従来の開発における、UI制御ロジック、ビジネスロジック、永続化ロジックの配置に関しては、一般的に次のような方法が考えられる。

クライアント/サーバー・アプリケーションを例とすると（ActiveServerPageを利用したWebアプリケーションでもほぼ同様だが)、以下の2つの方法がある。

①クライアントであるVB等に全ロジックを配置、あるいは、ストアドプロシージャを作成してDBMS上に配置し、クライアントから呼び出す形で構築する、いわゆる「コンポーネント化をあまり意識しない方法」。

②VBにてビジネスロジック、及び永続化ロジックをCOMとして作成し、クライアント上ではUI制御ロジックと、COMへのメッセージにより構築する「コンポーネント化をそれなりに意識した方法」。

.NETでは上記いずれの方法でも実装が可能であり、共に .NETでのさまざまなメリットを享受できる。ただし、①の「コンポーネント化を意識しない方法」では、拡張性と再利用性により高い生産性を実現するという、.NETが本来持つポテンシャルを引き出すことは難しくなる。このポテンシャルを最大限に引き出すためには、②の「コンポーネント化を意識した方法」での開発が必須だ。

また、②においても「共通ライブラリをコンポーネント化する」といった方法では、再利用性が多少実現できる程度で、「オブジェクト指向での適切な抽象化を取り入れた設計により、ビジネスロジック、永続化ロジックをコンポーネント化していく」という方法での開発が必要となる。

今回の開発では、ワークフローエンジンが、ビジネスロジックと永続化ロジックを司るコンポーネントとなり、ExcelがUI制御とコンポーネントへのメッセージを司る形となった。エンジンに関しては、当初からコンポーネント化を意識した設計を行い、必要と考えられるオブジェクト指向での抽象化を行った結果、拡張性・再利用性・生産性について、いずれも満足のいく成果を得る事ができた。

今後の .NETを利用した開発でも、引き続き今回と同様の方法を用い、更なる改善を加えつつ、開発を行っていく。

(3) Webサービスの利用

Webサービス化は容易であり、その効果も非常に大きいと考えられる。これは.NETでの利点の一つだ。

特に、今回のような、Excelなどのリッチクライアントを対象としたシステムであれば、従来難しかったエントリーを容易にしたシステムの開発が可能となり、印刷関係の問題も解決すると考えられる。

こうしたさまざまな利点から、今後、「Webサービス＋リッチクライアント」という形態は、どのようなシステムでも非常に有効と考えられる。今回のケースを参考にし、他のシステム開発においても、Webサービス＋リッチクライアントについて検討・展開していきたい。