統合的データモデリング支援環境"Darwin"(2)

4E－5　－ツール開発におけるオブジェクト指向の実践適用について－

郡司賢　武田和彦　丹羽隆　茅野康臣

NTTデータ通信株式会社　技術開発本部

1　はじめに

オブジェクト指向方法論の有効性が議論されてから久しく、現在では、実際の開発に適用した話も多く聞かれようになった。しかし、オブジェクト指向による開発への適用を考えた場合、ドキュメントの管理、作業プロセスの明確化、開発体制など、プロジェクトの運営に関する問題が存在している。

今回のツールの開発に先立って、OMT[1]をベースとした具体的な作業プロセス、ドキュメント管理などの開発手順書を作成し、全面的にオブジェクト指向による分析、設計、実装を実践適用した。

本稿では、オブジェクト指向の適用事例として、OMTをベースにした実践的方法論の紹介とその問題点について述べる。

開発の規模、期間および環境は、以下のとおりである。

開発期間	.... 94/4 - 現在
開発人数	.... 6人(プログラマー3人)
適用OS	.... Windows 3.1
開発言語	.... Visual C++ ver 1.51(MFC2.5)
クラス数	 122
ステップ数	.... 33K(+ コメント:20K)

2　OMTの具体化にあたっての問題点

今回の開発は、OMTの適用評価ではなく、製品作りであることから、手順そのものについては、あまり試行錯誤できないのが前提であったため、これまでの試行評価の経験から具体化した手順が必要になった。ベースにした方法論はOMT法であるが、開発プロセスを具体化するにあたって幾つかの問題点を解決した。

問題点1:ビュー設計(画面設計)は何から、どのように決めるか。

解決策1:どの契機に、なにを参考にしてビューを抽出するかという問題は、OMT法では、アクターとオブジェクトから導くと解説している。しかし、オブジェクト図にアクターを入れると、分析が難しくなり、設計との隔たりが大きくなる為、ツールの運用面を表現するモデルとしてワークモデルを考案し、オブジェクト分析と切り離して行うプロセスを加えた。

問題点2: 動的モデル、機能モデルを描くか。

解決策2:経験上、すべてのクラスについての動的モデルは必要ないし、すべての機能についての機能モデルも必要ないことが分かっている。また、OMTのそれらのモデル同士の関係が不明確であることからモデルの検証が不可能なので、描くことに大きな意味は無いと判断した。そこで、複雑なイベントのやりとりを整理する必要があるものだけ動的モデルを描き、複雑な処理をまとめるときのみDFDを描くことに決めた。

問題点3:ドキュメンテーションはどう整理するか

解決策3:「何」を描くかは決まっていても、グループで開発する場合、そのフォーマットや、管理方法を決めないと作業が始められない。そこで、ドキュメントの種類、フォーマット、バージョン管理、ドキュメントIDの付与基準を決めた。また、手作業での管理を前提としている為、修正の影響がなるべくひとつのドキュメントに局所化できるように書き方を簡略化した。

3　開発手順

図1は、今回適用した開発手順の作業の流れとドキュメントの流れの概要である。

3.1　要求分析

ツール開発においては、そのコンセプトを握っている開発者の考え方だけで仕様を決めてしまうと、出来上がったツールが実際にはユーザーに使いにくいものとなってしまうことが多い。今回は、データ分析を実際に行っているグループ(ユーザー)にインタビューを依頼し、データ分析/設計作業におけるさまざまな問題点を問題記述書にまとめた。これは、ワークモデル分析における重要な資料となった。

3.2　オブジェクト分析

オブジェクト分析では、要求分析で得られた問題記述から、オブジェクト/リレーションシップを抽出する。この作業は、一般的な「名詞句と動詞句からの抽出」の作業で行った。ここでは、ERモデルやプロジェクトなどの保存すべきオブジェクトのみを抽出し、UIに関わるアクター(人、マシン)は抽出しない。それらは、ワークモデル分析で抽出する。

3.3　オブジェクト設計

実装に使う言語、クラスライブラリ等を考慮し、リレーションシップの実現方法、データ型、コンテナーを決め、分析モデルを設計モデルへと変換する。分析モデルにアクターを含めないため、この作業は、ほとんど肉付け的な作業で済んだ。

3.4　ワークモデル分析/ビュー設計

ワークモデル分析では、問題記述書から、ツールの運用面を分析し、シナリオとワークフロー図を描く。このワークフロー図には、オブジェクト分析では抽出しなかったアクターが登場する。ワークモデルには、現状の

Integrated Data-Modeling Environment "Darwin"(2)

～Object oriented practical application to development of CASE tools

Masaru Gunji, Kazuhiko Takeda, Takashi Niwa, Yasuomi Chino
Research and Development Headquarters,
NTT DATA COMMUNICATIONS SYSTEMS Corp.

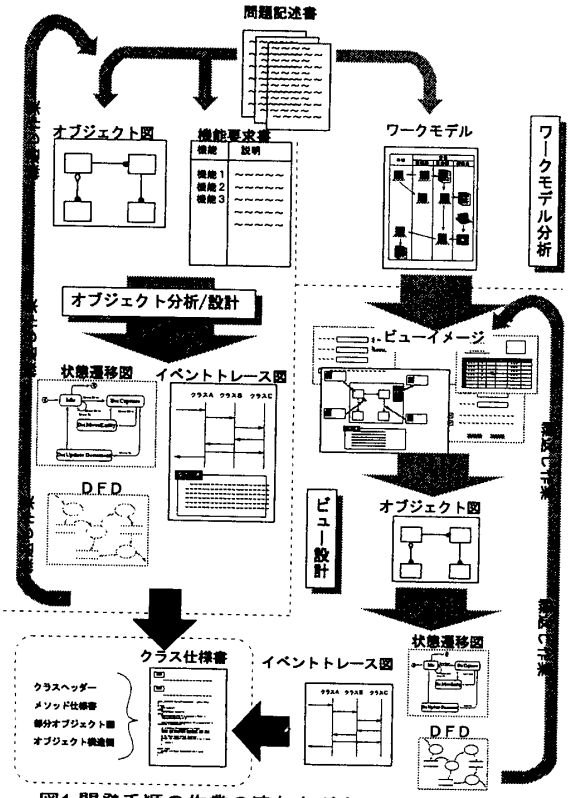

図1 開発手順の作業の流れとドキュメントの流れ

モデルと改善後のモデルの2つを描く。しかし、このアプローチは初めてだった為、フォーマリズムのあるモデルは描けずにビュー設計に移った。

ビュー設計では、シナリオ/ワークフローを元に、実装する環境に合わせた画面設計を行う。ここには、明確な変換規則というものは無く、経験に頼るしかなかった。

3.5 コーディング

コーディングは、分析の段階からプロトタイプ的に並行に進めた。分析の正当性と、適切な設計の為に、コーディング/実行を行い、トライ&エラーを繰り返して、モデルを最適化していく。

3.6 ドキュメント管理

予め定めたドキュメントの種類、ID付与基準に則り、ドキュメント管理ツールで表形式にまとめた(図2)。スプレッドシートの縦方向にドキュメントの種類で分け、横方向でバージョンの順番で管理する。このツールは、スプレッドシート上からオープン、印刷、削除などの操作ができるようにした。

モデルは、社内開発のOMTツールとシェアウェアの汎用的なドローイングツールを組み合わせて用いた。

クラス仕様書(クラス毎の役割や機能の仕様書)は、プログラムの修正との整合性を保つのに大変な労力が必要である。ひとたび、その労力を怠ったため整合性が失われると、「プログラムのみが真実」という結果に陥る。今回は、プログラムのコメントから仕様書を自動生成するプログラムを利用しこの問題を解消した。プログラムのコメント文が正しければ、常に約800ページのクラス仕様書は真実になる(図3)。

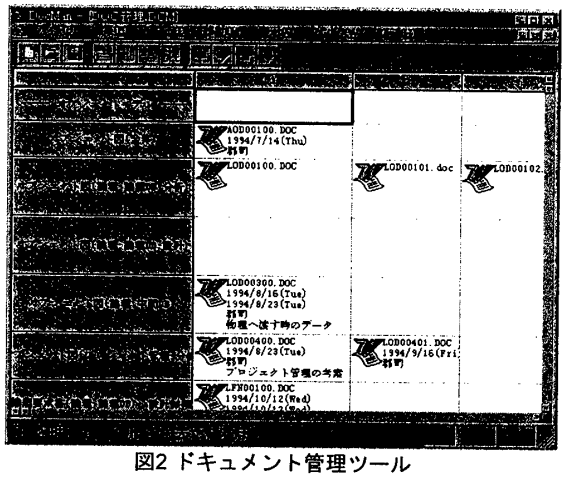

図2 ドキュメント管理ツール

```
////////////////////////////////////////////////////////////
// @mfunc ポリラインの各線分に対するヒットテストを行うコールバック関数です。
//
//    <mf DPolyline::HitLineTest> メンバ関数の内部からコールされます。
//
// @comm
//    呼び出し側であらかじめ HITRESULT 構造体を初期化しておく必要があります。
//
//    計算は、デバイス座標系で行います。
//
// @ex HITRESULT構造体の初期化 |
// HITRESULT htResult;
// htResult.ptClick = ptTemp;              // ヒット位置（デバイス座標系）
// htResult.dwLineWidth = SELECT_LINE_GAP;// ヒットしたとみなす距離（デバイス座標系）
// htResult.nHitResult = HT_NONE;    // ここに結果が返ってくるのであらかじめ初期化しておく
//
// @xref <mf DPolyline::HitTest>, <mf DPolyline::HitEachPtTest>, <mf DPolyline::HitLineTest>
//
void CALLBACK EXPORT DPolyline::HitLineTestProc(
    int xPos,  // @parm 線分の任意のドットの x 座標デバイス座標系で指定します。
    int yPos,  // @parm 線分の任意のドットの y 座標デバイス座標系で指定します。
    LPARAM lpData) // @parm ヒットテストする条件を ptClick, dwLineWidth に指定します。
                   //      ヒットテスト結果が nHitResult に格納され返されます。
{
                    :
                    :
}
```

図3 プログラムのコメント文

4 おわりに

今回のツール開発は、それほど大きなプロジェクトではないが、複雑で、ソフトウェアそのもののクオリティーが重視される。完成しても使われない(使えない)ツールでは意味が無い。今回使ったワークモデルは、まだまだフォーマリズムの点で未熟であるが、ユーザーの運用面を分析し、より使えるものを開発する上で重要であると認識した。また、ドキュメント管理においては、より強力な関連性を管理できるツールが必要であると実感した。

今回の開発手順は、OMTやプログラムの知識等の開発者の経験を前提として、それらをまとめる方法を示しているに過ぎない。これに加えて、方法論そのもののフォーマリズムが向上すれば、より明確な開発プロセスを定義できるだろう。また今後、分散開発を考慮した開発プロセスも作っていきたいと考えている。

参考文献

[1]J.ランボー他著 (羽生田 栄一 監訳)
「オブジェクト指向方法論OMT」