ソフトウェア工学 108－10
（1996．3．22）

MVCモデルを利用した分業型オブジェクト指向開発

松澤由香里　山城明宏

（株）東芝　システム・ソフトウェア生産技術研究所

オブジェクト指向開発を実システムに適用するためには、適切なオブジェクトの選択と大規模システムに対応するための分業化指針が重要な要件となる。この2点の要件に対応する手段として、問題分野に特化したパターンの利用と、OMT法を分業開発用にカスタマイズしたOMT-CD法を提案する。本稿では、特にGUIアプリケーションにおいて最適とされるMVCモデルを実システムに適用した具体的手順と、MVCモデルを利用したサブシステム分割による分業開発法OMT-CD法の運用手順を述べ、評価を行なう。MVCモデルを利用したクラス構造は頑強なものであり、またOMT-CD法により設計/実装における作業効率の向上が確認できた。

OMT Cooperative Development Using the MVC Model

Yukari Matsuzawa,　Akihiro Yamashiro

Systems & Software Engineering Lab., Research & Development Center,
Toshiba Corporation

This paper proposes how to use the MVC Model and the OMT-CD(Cooperative Development) method to architect large scale GUI application.
When we develop an actual system, two important guidelines are missing in traditional Object-Oriented development: One is a guide for object's extraction and the other is a guide for cooperative development. To make these guidelines, we firstly applied the MVC model which is one of an effective pattern in GUI domain. Secondly, we constructed a class architecture and then divided the system into subsystems, which are cooperative development unit. The class architecture based on the MVC model is robust and the application of the OMT-CD method can improve the productivity of Object-Oriented software.

1. はじめに

ソフトウェアサイクルを考えた場合、再利用性・拡張性は重要な要件となる。オブジェクト指向によるソフトウェア開発では、手続きとデータを一体化させた「オブジェクト」と呼ばれるグループ単位によりソフトウェア構造を構成することで、オブジェクト毎の再利用、オブジェクト内での変更吸収が可能な柔軟なソフトウェアを構築できる。このようなオブジェクト指向によるソフトウェア開発を導入する際には、オブジェクトとその静的構造・動的振る舞いを決定するためのオブジェクト指向分析/設計/実装という一連のフェーズを支援するための標準アーキテクチャと開発方法論が必要となる。現在我々は、このオブジェクト指向による開発を、理論レベルから実用レベルに引き上げるために、いくつかの分野のシステムを対象にオブジェクト指向開発を試行し実際の効果を見極めるとともに、その問題点の発掘と打開策を検討している。

今回我々が対象としたのは、ユーザインタフェースの事前の試作・動作確認を可能にするラピッドプロトタイピングツールMuseである[1]。このようなGUIアプリケーションに対するオブジェクト指向言語の有効性は既に広く知られており[2]、分析/設計/実装と一貫してオブジェクト指向を適用するアプローチを採用した。オブジェクト指向の開発手法は現在40種類近くの手法が提案されているが、特に分析/設計フェーズにはポピュラーでわかり易いOMT法[3]を、実装言語には普及度の高いC++を用いた。実装規模は20KLOC程度である。

これら分析/設計/実装と一貫したオブジェクト指向開発を通じて、我々は以下の2項目について焦点をあて検討した。

1. パターンを利用したクラス構造
2. 分析/設計/実装と一貫したグループ作業

これまで我々が実システムに分析/設計/実装と一貫適用して得られた知見の1つとして、ハード制約や処理速度向上のため、分析時と設計時ではクラス構造がかなり変化する場合があることが確認されている。そこで、クラス構造にパターンを利用することを検討した。パターンとは問題領域の分野毎に存在する最適なクラス構造の枠組みの指針のことである。今回の対象システムであるGUIアプリケーションにおける最適なパターンを利用することで、より変更・拡張に強いクラス構造を構築できるのではと考えた。また、グループ作業によるオブジェクト指向開発は、これまで少数しか報告されていない。今後さらに大規模システムへの適用を推進していくためには、グループ作業による一貫した開発は必須事項と考えられる。そこで我々はグループ作業において重要になるグループ内の仕様書一貫性保持のための新たなフェーズを設け、OMT法を改良運用したOMT-CD(Cooperative Development)を提案した。

本稿では、分析/設計/実装を一貫適用した結果、今後オブジェクト指向開発を実務レベルで進めていく上で重要と思われる上記2点について、開発の経緯を踏まえ報告する。

2. 対象システムと利用技術

本章では適用対象システムであるMulti Modal User-interface design Support Editor(Muse)の概要と、この開発で利用したOO技術マップについて説明する。

2.1 対象システムMuse

Museは、SEが客先と仕様決定を行う際に起こる「仕様検討→試作→仕様決定」というルーチンの期間短縮を狙っており、UIの画面仕様をデザインし、そこに仮想的な操作環境を実現し、デモを行うことで客先との仕様決定を行うためのラピッドプロトタイピングツールである。本ツールは、設計画面のGUI部品によるデザイン及びこれらGUI部品の動作を定義するための「編集機能」と、内部動作を設定した設計画面を実行するための「プロトタイプ実行機能」から構成される。図1に本システムの画面イメージを示す。

尚、以後Museでは設計画面をカード、画面設計に使用するGUI部品をUI-objectと呼ぶ。

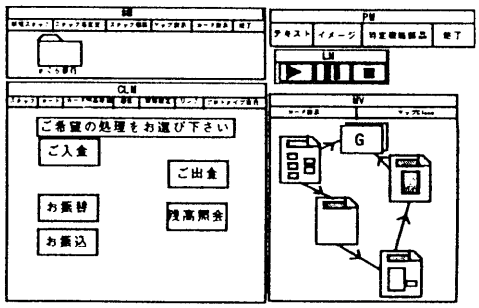


図1 Museの画面イメージ

●編集機能

編集機能には以下の4つのマネージャーが存在する。

(1)SM(スタックマネージャー)
　カードの集合を格納するスタックの管理と編集

(2)MV(マップビューア)
　カード間のリンク情報のマップ表示

(3)CLM(カードレイアウトマネージャー)
　カードとUI-objectの編集とリンクの設定

(4)PM(パーツマネージャー)
　UI-objectの管理

●プロトタイプ実行機能

(1)LM(ログマネージャー)

プロトタイプ実行時にはLM（ログマネージャー）のみが起動し、プロトタイプの実行、一時停止、中止を制御する。

2.2 利用技術

図2にMuse開発で利用したオブジェクト指向技術マップを示す。

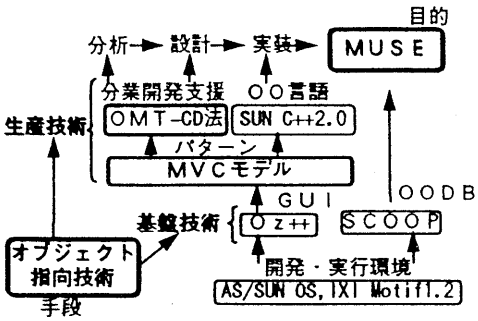


図2 オブジェクト指向技術マップ

図2は、Museを開発する手段としてどのようなオブジェクト指向技術を利用したのかを示している。オブジェクト指向技術には生産技術と基盤技術がある。生産技術は開発対象アプリケーションの分析/設計/実装を支援する技術である。また、基盤技術とは、開発・実行環境におけるプラットフォームを示す。

Museの開発では、生産技術として分析/設計において分業開発を支援するOMT-CD法を用い、実装にはOO言語としてSUN C++2.0を流用した。また、開発を通し、一貫してMVCモデルというパターンに準拠したアーキテクチャを採用している。基盤技術としては、GUIクラスライブラリ：Oz++[4]と自社で開発されたOODB：SCOOPを用いた。

3 開発で利用したOO技術の特徴

本章では、Muse開発で着目したOO技術である、パターン化技術と分業型OO開発方法論の2点についてそれぞれ述べる。

3.1 パターン化技術

オブジェクト指向分析において常に障害となる問題が、何をオブジェクトとするのか、どんな基準でオブジェクトを探せば良いのかという事である。このような時に1つの手助けとなるものがパターンである。パターンは、オブジェクト一式とその構造の抽象的な枠組みを与えてくれる。開発するソフトウェアの問題領域からいきなりオブジェクトを抽出するのではなく、パターンを前提に置き、そこにあてはまるようにオブジェクトを抽出していく方法である。このパターンを利用することでオブジェクト抽出のための指針を得ることができるとともに、柔軟なクラス構造を構築できる。

GUIアプリケーションの分野ではSmalltalkにおけるMVCモデルが代表的なパターンである。今回我々は、このMVCモデルをMuse開発において利用した。

3.1.1 MVCモデル

MVCモデルは、Model/View/Controller(MVC)の3種類からなるオブジェクト分類観点と、その関係を示したもので、Smalltalk-80において利用された枠組みである[5][6]。この3種類のオブジェクトとその関係を図3用いて説明する。

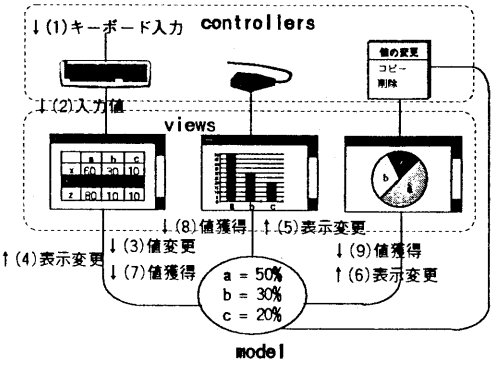


図3 MVCモデル

図3は1つのModelオブジェクトと3つのViewオブジェクト、3つのControllerオブジェクトを示している。Modelオブジェクトはあるデータ値を有し、このデータを3つのViewオブジェクトが様々な形式で画面表示する。また、各Viewオブジェクトは値を変化させるためのユーザ入力方式を定義する Controllerオブジェクトを持っている。Controllerオブジェクトがユーザからの入力を受け付けると、Viewオブジェクトを通してModelオブジェクトに値の変更が通知される。Modelオブジェクトは所有しているデータに変更が生じると、依存関係にあるViewオブジェクト全てにデータの変更を通知する。各Viewオブジェクトは、自分の表示形式に合わせて変更されたデータを表示する。このMVCモデルに見られる特徴は次の2点にある。

1.Modelオブジェクトに割り付けた複数のViewオブジェクトを矛盾なく動作させることができる。
(ModelとViewの依存関係)
2.ユーザ入力方式をControllerオブジェクトとして抽出することで、ユーザ入力方式を容易に変更できる。
(ViewとControllerの独立関係)

上記2点の特徴により、MVCモデルを利用したGUIアプリケーションは、画面仕様や入力方式の変更に強い拡張性のあるものとなる。

3.1.2 MuseへのMVCモデルの適用

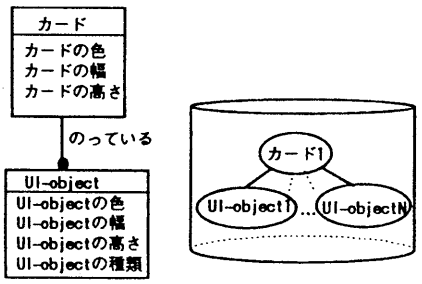
Museのオブジェクト指向開発では、分析/設計手法にOMT法を、またプラットフォームとしてGUIクラスライブラリとOODBを用いている。実装言語はC++を用いた。これを前提にMuseにおけるMVCモデル適用のための具体的手順を示す。

手順1：Modelクラスの抽出

●永続データを抽出する。
●対多関連を目安に永続データをカプセル化し、Modelクラスとする。

永続にすべきデータを抽出する。例えばユーザがMuse上で作成する画面(カード)仕様の色、種類、座標、画面遷移情報等が永続データである。

次にこれらの永続データをカプセル化してModelクラスとする。この目安として対多関連を利用する。画面仕様を考えた時、図4(a)に示される対多関連が永続データ間に成り立つ。またこの時のOODB内のオブジェクトの格納状態を(b)に示す。



(a)画面仕様内　　(b)OODB内

図4　永続データの対多関連

以上によりModelクラスを抽出する。

手順2：Viewクラスの抽出

●Modelクラスに依存するViewクラスを抽出する。
●画面仕様に記述されるGUI部品をViewクラスとして抽出する。

Modelクラスの所有するデータを画面上にさまざまな仕様を用いて表示するものがViewクラスである。従ってまず、Museの画面仕様からModelクラスのデータを表示するViewクラスを抽出する。次にその他の画面仕様であるメニューバー、ウィンドウ、スクロールバーといったものをViewクラスとして抽出する。図5 にMuseの画面仕様とModelクラス、Viewクラスの対応関係の一部を示す。

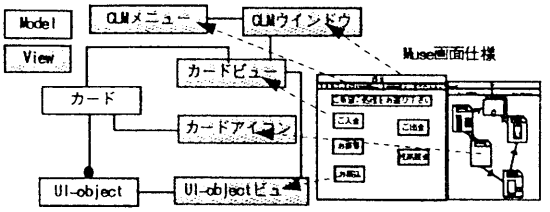


図5　画面仕様とModel、Viewクラスの対応

手順3：Controllerクラスの抽出

●機能単位にサブシステムに分割する。
●サブシステムの状態を管理するControllerクラスをサブシステム毎に追加する。

MVCモデルにおけるControllerクラスの定義は、図3で表わされているように、ユーザ入力に対する応答メカニズムを与えるものである。しかし、我々はControllerクラスをより広義にとらえた。図3でControllerクラスは、1つのウィンドウ内で提供される機能を管理しているものと考える。Museを対象とした場合、図1に示されるようにウィンドウ毎に提供される機能がグループ化され、役割が明確である。従って、1ウィンドウを目安としてサブシステム分割を行い、サブシステム毎に1つのControllerクラスを追加した。我々の定義したControllerクラスの役割は次の通りである。

Controllerクラスの定義：

●1サブシステムを管理するものであり、各サブシステム間のインタフェースの窓口となり得るものである。

サブシステム内の各機能は、それを管理するControllerクラスに依頼することで実行される。Controllerクラスの追加により機能実行の役割分担が明確化され、サブシステム間で一貫した機能の共有がはかれる。Controllerクラスを追加したMuseのMVCモデルによるクラス図を図6に示す。

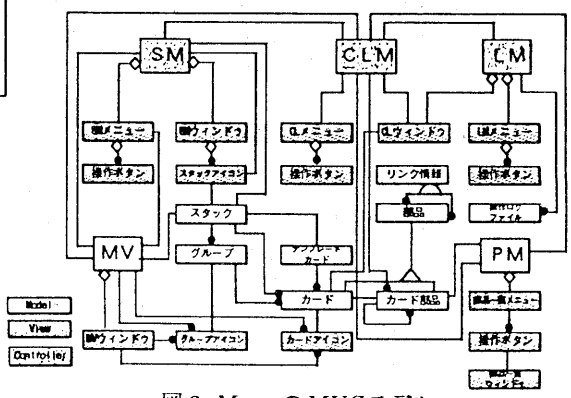
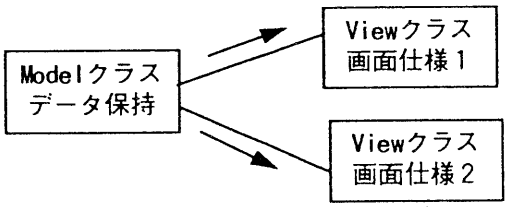


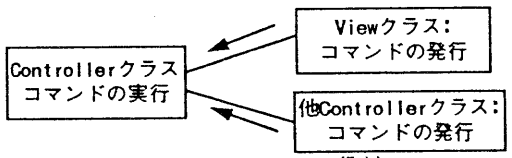
図6　MuseのMVCモデル

3.1.3 MVCモデル利用の妥当性

以上のような手順により、MuseというGUIアプリケーションに特化したMVCモデルによるクラス構造を構築した。Smalltalk-80で提唱されるMVCモデルの2つの特徴をMuseに合わせてカスタマイズすることによって、より柔軟性ある構造とした。ModelとViewの切り分けと依存関係は維持し、かつサブシステムを管理するControllerクラスによりサブシステム間のインタフェースの統一ができる。これにより、ユーザ入力方式が変わってもControllerクラスの提供するインタフェースを使うことでその他のクラスへの変更波及が生じない。この2つの特徴を動的側面から示したものが図7である。



(a) ModelクラスとViewクラスの関係



(b) Controllerクラスの役割

図7 動的側面からのMVCクラスの役割

図7で矢印はメッセージ通信の向きを示している。(a)において、Modelクラスは、保持しているデータに変更が生じた場合、依存するViewクラス全てに変更通知をすることでデータ表示の一貫性を保持する。(b)において、複数のViewクラスや他Controllerクラスから同一のコマンドが発行されても、実行するContorollerクラスはただ1つに限定されているため、実行順序による実行結果の無矛盾性を保証する。この場合Controllerクラスの役割を、あるViewクラスに担わせることも可能であるが、ModelとViewの関係のように分離させることで、View仕様変更による影響をControllerクラスが受けずに済む。

このように、Model、View、Controllerという静的側面の定義にもとづき、問題領域からオブジェクトを抽出する。次に、抽出されたオブジェクトに対して図7に示されるようなModel-View-Controller間の依存関係を動的側面として設定する。これによりMVCモデルを利用したクラス構造は、柔軟性・拡張性・一貫性のあるものとなる。

3.2 分業型OO開発方法論

本節ではMuseのOO開発における2つめの着眼点である分業開発について報告する。

3.2.1 OMT-CD法

Museでは分析/設計/実装と一貫してグループ作業による分業開発を進めてきた。基本的にはOMT法を利用し、それを分業開発用にカスタマイズして運用した。このカスタマイズした運用方法をOMT-Cooperative Development(OMT-CD)法と名付けた。作成したモデルは、オブジェクトモデルと動的モデルのみである。本来作成するべきである機能モデルは、動的モデルによりその仕様書の役割を代替できるとして省略した。以下に分析/設計/実装の各フェーズにおけるOMT-CD法を既存のOMT法と比較して述べる。

3.2.2 分析フェーズ

図8が通常のOMT法における仕様書の作成手順と、OMT-CD法による作成手順を比較した図である。OMT-CD法の基本手順は既存のOMT法にしたがった。

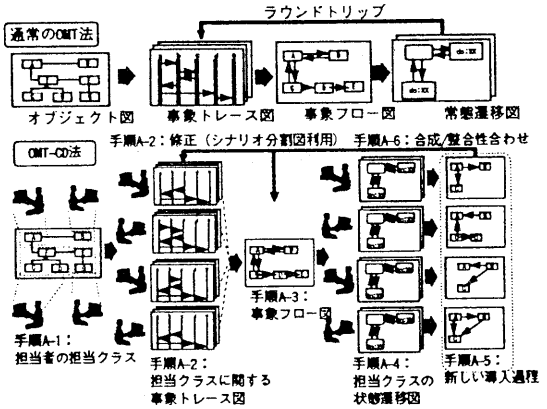


図8 OMT法とOMT-CD法の比較（分析）

OMT-CD法の分析では図8に示されるA-1～A-6までの6つの運用手順を用い、ラウンドトリップに作業を進める。この中で、特に分業開発を効率よく進める上で重要な、分業単位を決めるためのフェーズである手順A-1と、担当者間の一貫性を保持するフェーズであるA-5、A-6について次に説明する。

3.2.2.1 分業単位決め(A-1)

分業開発では、どのタイミングで分業を始めるか、またどのような開発単位で分業を進めていくかということが重要になる。我々は、分業単位をサブシステム単位として開発すると、オブジェクト間の

メッセージ通信がある程度サブシステム毎にパッケージ化できるとともに、担当者が理解し易いと考えた。そこで、オブジェクト図を作成した段階でサブシステム分割し担当範囲を決めた。オブジェクト図の作成指針については第3 章で説明したように、MVCモデルに基づいて構築した。この中で3.1.2節でControllerクラスをサブシステムの管理役とし設けた。従って、Controllerクラスを中心とするサブシステムを１人分の担当範囲とすれば良いことがわかる。図9 において太線で囲まれた部分がサブシステムを表わす。これが１人の開発担当範囲となる。

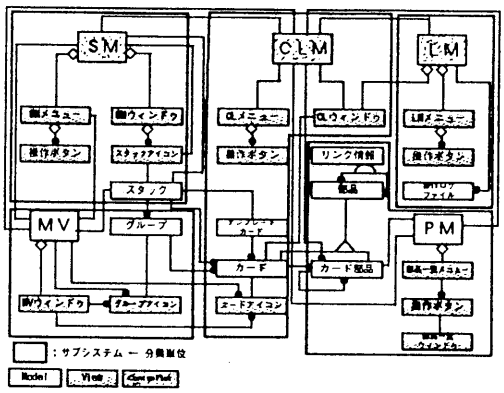


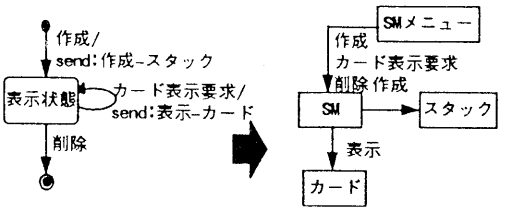
図9 Museのサブシステム分割

サブシステムの分割単位は、Controllerクラスを中心に、それが管理するViewクラスとModelクラスを１つのサブシステムとし、切り分けを行っている。例えば、ControllerクラスがSMクラスの場合、それが管理するViewクラスにはSMウィンドウ、スタックアイコン、SMメニュー、操作ボタンが相当する。また、Modelクラスにはスタックが相当し、これらが１つのサブシステムを形成する。

以上のように、サブシステムを分割し分業単位を決める。

3.2.2.2 担当者間の整合性保持(A-5, A-6)

分業により仕様書の作成を進めていった場合、どこかで担当者間の矛盾をなくすために、整合をとるフェーズが必要となる。今回我々は、この整合性保持のための仕様書として、状態遷移図を作成後、担当クラス間の事象フロー図を作成することにした。図 10に状態遷移図から事象フロー図への変換過程を示す。



(a)SMクラスの状態遷移図　(b)事象フロー図

図10 状態遷移図から事象フロー図の変換

各クラスの状態遷移図におけるイベントとアクションに注目して事象フロー図を記述する。イベント(作成、カード表示要求、削除) はクラスに受信されるメッセージとなる。アクションに記述された「send:xx-yy」はyyクラスにxxというメッセージが送信されることを示す。すなわち、スタッククラスへ「作成」、カードクラスに「表示」というメッセージが送信されることを示す。以上により、(a)の状態遷移図から(b)の事象フロー図が作成できる。

このように担当範囲のクラスの事象フロー図を作成しそれらを合成し整合性を保持する。図11は担当者A,Bが作成した事象フロー図である。これを比較すると、CLMクラスからSMクラスへのメッセージ「スタック獲得」が担当者B の事象フロー図に無いという矛盾が発生していることがわかる。この矛盾について担当者間で調整する。

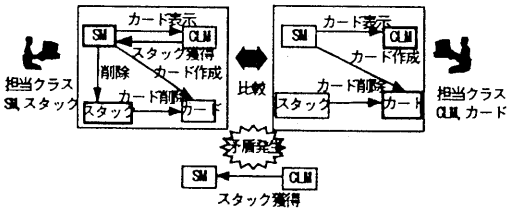


図11 事象フロー図による担当者間の整合性保持

各担当者が作成した事象フロー図を全て合成し、整合性を保持する。最終的にシステム全体のメッセージフローを記述した１枚の事象フロー図を作成する。

3.2.3 設計フェーズ

図 12に設計フェーズにおけるOMT法とOMT-CD法の手順について示す。従来のOMT法では作成していた設計用の事象トレース図をOMT-CD 法では記述しない。これは分析段階でオブジェクト間のメッセージパッシングを十分検討したためである。必要であれば記述する。

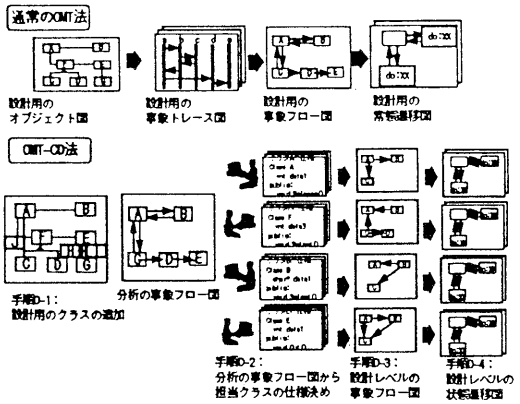


図12 OMT法とOMT-CD法の比較(設計)

OMT-CD法の設計では図12に示されるD-1～D-4までの4つの運用手順を用い、ラウンドトリップに作業を進める。手順D-1は、従来のOMT法と同様の手順であるが、分析から一貫した分業開発を維持するためには、クラスの追加によりサブシステム分割が影響を受けない必要がある。また、手順D-3、D-4については分析時と同様に、担当者毎に担当範囲のクラスについて設計用の事象フロー図と状態遷移図を記述する。担当者毎の事象フロー図は、サブシステム間のメッセージパッシングのインタフェースを定義し、担当者間の整合性を保持するのに役立つ。ここでは特に、サブシステム分割に影響する可能性がある手順D-1について次に説明する。

3.2.3.1 設計用クラスの追加(D-1)

分析で作成したオブジェクト図に実装を考慮した設計クラスを追加し設計用のオブジェクト図を作成する。

本対象システムではプラットフォームとしてOODBとGUIクラスライブラリを利用した。設計では特にこのプラットフォームを考慮して、クラス追加とクラス毎の基底クラスの決定を行った。設計用オブジェクト図の作成のポイントを以下に示す。

設計用オブジェクト図作成ポイント
- ●Modelクラス間の対多関連の間にOODBで提供されるコレクションクラスを追加する。
- ●エラーメッセージ、ワーニング等のメッセージダイアログの追加と、メニューの表示方式(メニューバーか、ポップアップメニューか)等のGUI概観の決定をする。
- ●各クラスにおいて利用可能なクラスライブラリを決定する。

上記3つのポイントを考慮して作成した設計用オブジェクト図を図13に示す。斜線の四角がGUI概観のためのクラス追加、網掛けの四角がコレクションクラスの追加となっている。また、四角内の上部に記述されている英語のクラス名が、利用できる可能性のあるクラスライブラリ名となっている。

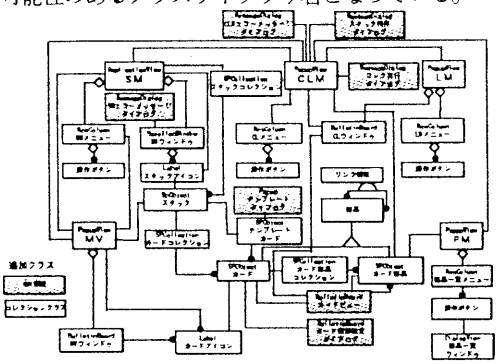


図13 設計用オブジェクト図

設計用に追加されたクラスは、MVCモデルにおけるModelクラスとViewクラスにあたる。これらの追加クラスは、各サブシステム毎に追加されるため、サブシステム分割単位に影響を及ぼすことはない。従って、分析/設計と一貫した分業開発を進めることができる。

4 評価

本章では、今回開発したMuseに対する評価を以下の2つの観点から行なう。

(1)MVCモデルを用いたクラス構造の安定性
(2)分業開発による生産性評価

上記2点について順次述べていく。

4.1 MVCモデルの安定性

MVCモデルの安定度を分析/設計/実装の各フェーズに追加されたクラスの内訳を見ることで評価する。図14は分析時に抽出されたクラスに、設計/実装時にどのようなクラスが追加されたかを示している。

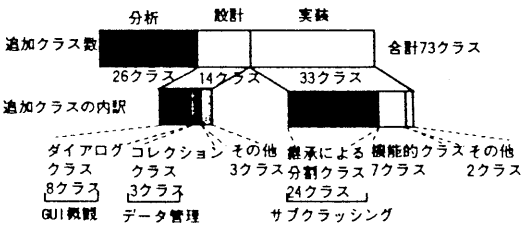


図14 設計/実装時に追加されたクラスの内訳

図14から、分析時には問題領域のクラスが抽出され、設計時には、エラーダイアログやワーニングダイアログといったGUI概観と操作性のためのクラスが追加される。実装時には、主に設計までのクラスから派生クラスを作り出すサブクラッシングによりクラスが追加された。すなわち、分析時に構築されたMVCモデルによるクラス構造は設計/実装時にも維持され、安定していることがわかる。

4.2 OMT-CD法を用いた開発工数の評価

本節ではOMT-CD法による分業開発の生産性について評価する。図15は他GUIアプリケーションとMuseの分析/設計/実装の一貫適用によるオブジェクト指向開発の作業工数と作業人数を表わしている。縦軸に対象アプリケーション、横軸に作業月数を示している。黒が分析、網掛けが設計、斜線が実装の作業工数を示している。分析、設計、実装のそれぞれの下に示された数が作業にあたった作業人数となっている。

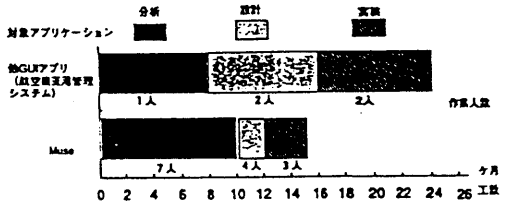


図15 作業人数による作業工数の比較

図15に示されるように、分析フェーズでは作業人数が1人から7人に増えても全体の作業工数は上がらないどころか、却って悪くなっていることがわかる。分析作業はシステムの初期段階のため、この時期に担当者レベルを含めた作業を行うと、グループ内のコンセンサスをはかることが困難になり、単独作業で行う場合よりも作業効率が悪くなる。一方、設計、実装時には明らかにグループ作業による分業化により作業効率が上がっていることがわかる。すなわち、分析時はチームリーダレベルの少人数で作業を行いシステムの基礎固めをしたのち、担当者レベルにその情報を伝達し設計/実装作業の分業化をはかることが、システム開発全体の効率化につながるといえる。

5 まとめ

本報告では、MuseのOO開発を、パターン化技術、分業型OO開発方法論という2点に着目して報告した。MVCモデルというパターンを利用したクラス構造は、現実的な物理構造と対応が良く、分析/設計/実装フェーズを通して一貫して維持され安定していることを確認できた。さらに、MVCという枠組みでのサブシステム化により、分業単位を明確にでき、これを利用したOMT-CD法は担当者間のインタフェースの統一と、設計/実装における作業効率を向上する手段として有効であることが確認された。

今後は、フレームワークを前提としたパターン指向によるオブジェクト指向開発及び分業化のマネージメントとツールによる支援について検討し、最終的に基盤技術の利用を前提としたオブジェクト指向分析/設計法の確立とその分業化支援を目指す。

参考文献

[1]Hiroyuki Kamio et al."A UI DESIGN SUPPORT TOOL FOR MULTIMODAL SPOKEN DIALOGUE SYSTEM, "ICSLP, Japan, Scp.pp.1283-1286, 1994

[2]青木著, 「例題による！！オブジェクト指向分析設計テクニック」, SRC, 1994.3

[3]ランボー他著, 羽生田監訳, 「オブジェクト指向方法論OMT-モデル化と設計-」, トッパン, 1992.7

[4]新井著, 「Motif/C++ Programming」, HBJ出版局, 1994

[5]G.E.Krasner and S.T.Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Journal of Object-Oriented Programming, August/September, pp.26-49, 1988

[6]Erich Gamma他著, 本位田, 吉田監訳, 「デザインパターン」, SOFTBANK, 1995.10