ソフトウェア工学 92－4
（1993．5．26）

C++ OMTool の大規模ソフトウェア開発への適用事例

井上 健　　土田 崇　　村田なつめ　　西岡健自
横河電機株式会社 オープンシステム研究所

OMT法によるオブジェクト指向分析／設計をグラフィカルかつインタラクティブに支援するOMTool を用いてグラフィックエディタ開発のためのツールキット GIST (Graphical Interactive Systems Toolkit) を開発した。本稿ではその事例紹介とそれにもとづいて、OMT 法や OMTool の評価を行う。

A Report of Large Scale Application Construction with C++ OMTool

Takeshi Inoue, Takamu Tsuchida, Natsume Murata, Kenji Nishioka
Open Systems Laboratory, Yokogawa Electric Corporation

We developed a toolkit named GIST(Graphical Interactive Systems Toolkit) which supports developing graphical editors. To develop GIST, we used the OMT methodology and a software tool, 'OMTool', which graphically and interactively helps users draw OMT based object models on a screen. In this paper, we report our designing process of GIST and then, we evaluate the OMT methodology and OMTool.

1 はじめに

オブジェクト指向による分析、設計にＧＥ中央研究所（以下 GECRD）のランボーらによるＯＭＴ法が提案されており [1]、GECRD では、この手法にもとづいて、開発支援を行うソフトウェアツール OMTool を開発した。当社では OMTool の開発初期より評価およびエンハンス提案を行い、C++用に改良する際にはその開発にも参加した。さらにそれと同時進行の形で、グラフィックエディタ開発支援ツールキットである GIST(Graphical Interactive Systems Toolokit) の開発を行った。

本稿では OMTool と GIST の簡単な紹介のあとに、OMTool を用いて GIST を開発していった過程、最後に OMT 法や OMTool の評価などを行う。

2 C++ OMTool の概要

OMTool は OMT 法によるオブジェクト指向設計の３つのモデル、オブジェクトモデル、動的モデル、ファンクションモデルのうちのオブジェクトモデル設計を支援する GUI アプリケーションである。

図１に、OMTool によって設計するクラス階層表示の例を示す。OMTool を使用すると、クラスボックス表示、クラス名、メンバ名、メソド名記述が順次行え、マウスで自由にボックスを移動できる。また、クラス間の関連も、スーパークラス-サブクラス、Association（関連）、Aggregation（集約）などを指定し、それぞれ OMT 法で提案された線の形式でクラスボックスが結ばれる。結ばれた線もマウスでグラブして自由に移動することが可能である。マウスやキーボード操作によるレスポンスは非常によく、図形の移動や線分による連結などもスムーズである。またクラスボックスは OMT 準拠の四角形だけでなく、楕円や角の丸い四角形も可能。画面上でのボックスの色つけも可能である。

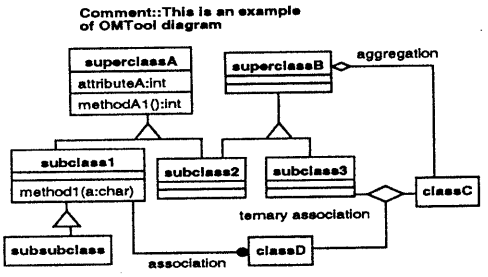

図１ OMTool によるオブジェクトモデル表示例

コメントは、詳細なレベルまで記述可能である。すなわち、モジュール全体、ページごと、クラスごと、メンバごと、メソドごと、クラス間の関連ごとなどにポップアップダイアログに記入する形式で記述できる。また、関連や集約には役割 (role) 記述も可能で、線分の近くに文字表示がされる。

設計が進み、クラスが増えすぎて初期のページの大きさにはいらない場合、ページそのものを大きくしてもよいし、複数のページに分けることも可能である。OMTool はクラスの名前を管理しており、複数ページに同じクラス名が存在すると、その内容を一方で変えても自動的に他方のページに反映する。この機能と、クラスボックス表示を、クラス名のみまたはメンバやメソドも表示、など選択可能な機能により、複雑なクラス階層をもつソフトの設計においては、最初のページにクラス名のみを使ったクラス関係を表示し、２ページ目以降に各クラスの詳細（メンバおよびメソド）記述を行うという方法が効率的である。

設計した OMT の図面は独自のフォーマットでアスキー文字列として保存される。また PostScript などの形式に OMT 図を出力してプリンタに印刷することも可能である。

設計者が OMTool を使用して設計をすすめる一般的なプロセスは、以下の２～６を繰り返しながら行う形式になりうる。

1. OMTool を起動する。
2. 空のクラスボックスを数個適当に作る。
3. クラスボックスを埋めてゆく。
4. クラス間の関係を規定する。
5. クラスの詳細記述（属性定義、メソド定義）を行う。
6. クラスがふえすぎたら、新しいページを作って、いくつかのクラスを次のページにコピーする。
7. レイアウトを整える。
8. C++ソースコードを出力する。

C++ソースコードは、ヘッダファイル (.h ファイル) と、ソースボディファイル (.cc、.C, など拡張子は選択可能) の２つのファイルが生成され、ソースボディの方は各メソドの、関数宣言部を含む枠組のみが生成される。

設計後の実装では、クラス変更やメソドのインタフェース変更などはすべて OMTool 上で行えば、ソースファイルも自動的に変更される。つまりヘッダファイルに関してはプログラマが直接編集することは許されず、すべて OMTool 側で変更したものを自動反映するという方式をとっている。また、OMTool 上の変更はヘッダファイルだけでなく、ソケットで OMTool につながれた emacs 上のソースボディにも自動反映され、プログラマはそこへ実行コードを記入してゆく。この形式で設計／編集を繰り返す限り、ヘッダファイルとソースボディ間でメソドインタフェースの矛盾は起きず、プログラマは OMTool と emacs 上のソースボディのみを扱えばよい。

3 OMToolを用いて開発したGISTの紹介

GISTはもともと、OMToolのすぐれたユーザインタフェースが出発点となり、OMToolのようなグラフィックエディタを簡単に作るためのツールキットを作りたいという発想から生まれた。GISTを使うことによって、幾何図形を描画／操作するグラフィカルでインタラクティブなエディタを容易に開発することが可能である。

GISTはOMT法によるオブジェクト指向に基づいて開発されたC++のライブラリであり、その開発にはOMToolの本格的な利用があった。

本稿では、OMToolの利用を紹介するための例としてGISTを挙げるため、GISTの構成を中心に紹介し、機能的な特徴などの紹介は省略する。

GISTの構成面からみた特徴は以下のとおり。

- Subject-View-Imageモデル

  アプリケーションが使用するデータをSubject、表現をImageとし、それをViewクラスが連結する方式をとった。これによりデータを多様な形式で多様な出力に一貫した形式を保って表示できるようになった。

- 図形オブジェクトの構造化による表示管理

  Imageクラスでグラフィック木を管理し、表示オブジェクトの表示／隠蔽などが効率よく確実に行える。

- 図形間の意味的な関連が記述可能

  図形オブジェクトを連結するためのPadオブジェクト、Jointオブジェクトを定義することにより、図形の生成／消去／移動／回転／変形などの変化に対して自動的に他オブジェクトへの影響を計算し、管理するシステムを実現。

- EIL(Event Interpretation Language)の導入

  ユーザインタフェースや、アプリケーションの振舞いはイベントをトリガとする状態遷移を考え、EIL言語を用いて記述できるようにした。

- Xウィンドウ／Motifとのインタフェース

  GISTの中核となるクラスはいかなる出力とも独立に作り、そこへXウィンドウやMotifとのインタフェースをもつクラスを用意して実際に表示できるようにした。

GISTを使ったアプリケーションとしてはさまざまなグラフィックビルダやグラフィックエディタ、シミュレーションツールなど考えられる。GISTの規模はクラス数が約２００、全ソースコードが１０万行以上である。４人のエンジニアにより約１年半の期間で作り上げた。

4 OMToolを用いたGISTの開発過程

GISTの開発過程について述べる。要求の分析から設計まで約半年を要した。ここでは、分析／設計がどのようなものであったのかではなく、オブジェクト指向による分析／設計の過程での特徴的なことがら、評価に重点をおく。

分析と設計はどちらも同じメンバで作業したため、明確な分離、区別は意識しなかったが、オブジェクトとオブジェクト間の関係の決定までが分析であり、そのあとの、各オブジェクトの詳細設計にはいるところから設計フェーズにはいったという認識をもっている。もちろん設計の段階で分析に戻って考え直す作業はたびたび起きたが、OMToolのおかげでそれらの作業も画面上で自由に行えたため、紙と鉛筆、あるいは一般的な絵を描くためのツールを使う場合に比べて飛躍的に作業効率は上がっている。

4.1 開発の進め方

開発は以下の順序で進んだ。

1. ウィンドウシステムに関する勉強会、世間の動向調査
2. 全体のオブジェクトモデルを考え、動作の大まかなシナリオを検討
3. オブジェクトモデルの詳細化
4. 詳細設計作業とコーディング
5. 結合テストとデバッグ

設計は以下の方式で行われた。まずプロジェクトリーダのオフィスに4人が集まり、オブジェクトモデルやシナリオを話し合ったり、黒板を使ってデータのながれなどを検討した。オブジェクトモデルが検討されてゆく過程でリーダがワークステーション上でOMToolを使ってクラスの階層や定義を決めてゆき、クラス間の関係やクラス内部の定義もどんどん付加されてゆく形式で形成されていった。

4.2 オブジェクトモデルによる設計例

開発を始めるにあたって我々がGISTのゴールとした項目は以下のとおり。

1. グラフィック図形を自由に描けるアプリケーションの開発効率を高める。
2. グラフィックエディタとしての機能を果たすため、扱うデータと表示するグラフックとの関連の定義／管理を完全に行える。
3. エディタのユーザインタフェースを簡単に定義できる。

1番目の目標を達成するために行ったのが、Xウィンドウシステムの Xlib ライブラリを利用した、グラフィック描画のためのクラスを開発することであった。2番目のために GIST ではサブジェクト-イメージモデルを導入するところから始まった。3番目の目標のためには EIL の導入があった。

GIST は巨大なツールキットであり、１０００にもおよぶクラスがどのように作られていったかを述べられないので、ここでは上の2番目のサブジェクト-イメージモデルに例をとり、その開発過程の一部を紹介する。

ＯＭＴ法でいうオブジェクトモデルはシステム内のオブジェクトを決定し、オブジェクト間の関係や、オブジェクトの属する各クラスの属性、操作を定義するものである。今回の開発では分析段階でオブジェクトの決定（クラスの決定)、オブジェクト間の関係定義が最初に決定され、各クラスの属性、操作定義は設計フェーズで行われた。またサブクラスの定義やそれにともなう継承の定義は属性／操作を定義している過程で適宜行われ、開発の初期段階では殆んど問題にしなかったことに特徴がある。

アプリケーションで扱おうとするあるデータを、さまざまな表現法で表示できるようにしたいというのが、まずはじめの要求であった。ここでは Smalltalk のＭＶＣモデル [2] や、InterViews[3] でもとりあげられている Subjects-Views モデルなどでも議論されている問題を GIST でどう考えたかということを例に設計の過程を紹介する。

最初に考えたオブジェクトモデルは、図２に示すような、Subject-Image モデルである。SubjectElement のクラスのインスタンスとしてデータを用意し、そのデータに関連させていくつかの表現のための ImageElement を作っておけば、最初にデータが与えられ、それをウィンドウ上に表現する場合に、データに関連づけられた ImageElement の数分の表現で行われる。ImageElement では、図形を抽象的にとらえて表現を規定し、実際の描画のためには図のように ImageElement に関連させた GraphicElement クラスを用意し、すべての幾何計算を行う。

図２ 初期の Subject-Image モデル

設計が進行するうちにこのモデルだけでは単純すぎて、アプリケーションの要求が複雑になると首尾よく対応しきれないことに気づいた。おもな要求は２つの点にまとめられる。

ひとつはデータの変更の反映である。変更は SubjectElement 側のデータそのものの変更がトリガーになることもあるし、ユーザがウィンドウ上の情報をキーボードやマウスで変更し、その ImageElement の変更を SubjectElement は勿論、他の ImageElement へも反映するメカニズムが必要になる。図１のモデルで、これを一般的に扱おうとすると SubjectElement が単に ImageElement への関連を認識するだけ（ポインタをもつだけ）では不十分であり、ImageElement とのインタフェースも知らないといけない。ImageElement とのやりとりが高度になるにしたがい、SubjectElement に要求される ImageElement に対する知識も多くしなければならず、サブジェクトというデータ構造が「自分をどう表現するか」を意識したものとなってしまう。これはデータに独立性をもたせたいという我々の要求と相反するものになる。

もうひとつはインスタンスのダイナミックな生成／消去への対応である。SubjectElement が生成／消去されたとき、または、ImageElement のうちのひとつが生成／消去されたときの処理、管理をするオブジェクトが必要になる。これは SubjectElement または ImageElement 自身にさせるべき仕事なのであろうか。それとも独立なオブジェクトが必要なのだろうか。

第一の問題点を解決すべく導入されたのが ViewElement オブジェクトであった。ひとつの ViewElement のインスタンスは SubjectElement のひとつの表現を「管理」するもので、対応する ImageElement との橋渡しを行い、どちらかの変更を他方に伝える。今や SubjectElement は ViewElement の集合への関連をもち、変更が生じた場合には ViewElement に対して「変更反映せよ」というメッセージを送るだけでよい。このメッセージを受けとった ViewElement は自分が管理する ImageElement に対して所定の変更手続きを起動する。

この問題を考える際、OMT モデルの「リンク属性」または「関連をクラスとしてモデル化する」方法が考えられた。すなわち、ViewElement は SubjectElemet と ImageElement の関連がオブジェクトと化したととらえ、図３のように関連オブジェクトとして定義する方法である。OMTool にはこの関連オブジェクトの記述も用意しているため、これを用いた設計上の議論もしやすかった。結局 GIST の場合、SubjectElement、ImageElement 双方ともに、関連オブジェクトを介して相互に参照しあうことも、メッセージ通信することもなく双方の直接の関連が無視しうるため、この方法は採用されていない。

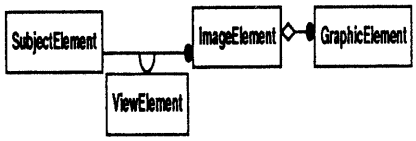

図３ ViewElement の関連オブジェクト化

第二の問題点はコンテナ（入れもの）オブジェクトの導入によって解決した。SubjectElement, ViewElement, ImageElement, GraphicElement それぞれのインスタンスの集合を管理するためのクラ

スがそれぞれ Subject, View,Image,Graphic として用意され、生成／消去のダイナミクスを管理する。ある Element オブジェクトが作られるとまずそれは対応するコンテナオブジェクトに登録され、関係各位に所定の情報が伝わって処理をする。

これらの関連を OMTool で表したものが図 4 である。

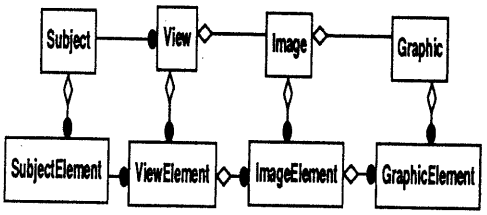

図 4 最終的な Subject-View-Image モデル

OMTool を使った分析／設計ではこのような新しいクラスの定義が非常に楽であり、さらに図 4 のモデルに対して属性、操作を定義しながら設計が進められた。SubjectElement, ViewElement の属性／操作は GIST を使ったアプリケーション自身に依存する部分が多く、GIST としては基本属性、基本操作のみを定義してアプリケーションを開発する際のスーパークラスとして機能する。一方 ImageElement クラス以下は GIST が用意するグラフィックライブラリを駆使するためのものであり、ImageElement クラスを定義しながらさまざまな図形や動作に応じたサブクラスを OMTool 上で設計していった。

4.3 最終的な GIST の全オブジェクトモデル

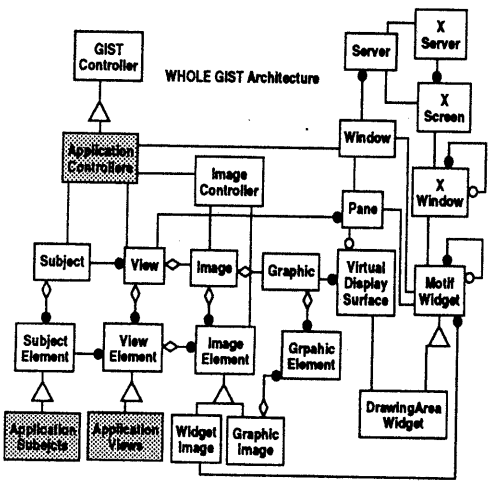

図 5 GIST の全体象

図 5 に GIST 全体のクラス階層を示す。暗色のクラスは GIST のユーザが開発する部分である。

GIST を使ったアプリケーションでは、SubjectElement, ViewElement のサブクラス定義によるデータ処理と表現、Controller クラスのサブクラス定義と EIL による動作定義による振舞い、ユーザインタフェース規定によって開発が行われる。

4.4 オブジェクトモデルによる分析／設計の特徴

我々の分析、設計の流れはおおまかに以下のようであった。

1. 「まず」何をオブジェクトにするのか、主役をつとめるいくつかのオブジェクトを考え、クラスボックスを作る。
2. 主役たちの関連、結び付きを考え、OMT 法に基づく表記法による線で結ぶ。(関係するオブジェクトの多重度 (multiplicity) はまだ考えない)
3. 主役たちを管理／操作するオブジェクトが必要になったら適宜定義する。
4. 関連そのものに単純でない役割があるとわかった時点で、関連オブジェクトまたは、独立したオブジェクトに昇格させ、それらを主役たちと関連づける。
5. 各オブジェクト自身について考え、属性、操作、サブクラス化などを定義する。この間に随時、関係の multiplicity などもつめて定義してゆく。

頭の中にあるオブジェクトの概念をオブジェクトモデルで表現することは、我々にとって非常に自然に行える作業であった。Coad/Yourdon の方法 [4] にある、'is a' 関係や'a part of' 関係は OMT 法ではそれぞれ汎化 (generalization)、集約 (aggregation) と表されるが OMT 法で挙げている関連 (association) は我々にとってより意味深く、初期の段階で定義されることによって、設計が手際よく行われたことを強調したい。また、汎化、集約も並列に存在する 2 つの「関係」ではなく、集約を定義したあとに汎化を考えつめてゆくのが有効な方法に感じられた。

関連と集約は明確に区別しにくい場面もあったが、まず関連として定義し、のちに 2 つのオブジェクトの包含関係や主従関係を調べながら集約関係になおすことによって、困難なく定義することが多かった。

4.5 動的モデル／ファンクションモデル

現在の OMTool が提供するのはオブジェクトモデリング支援の機能だけであり、動的モデル／ファン

クションモデルによる分析／設計はツールを使わずに行わざるをえない。

GIST 自身は、上のセクションに述べた、Subject-View-Image モデルにおける、変更時のオブジェクトの動きや、図形描画の際に問題になる図形ハイライト、ダメージオブジェクトによる再描画処理などで動作が問題になったが、GIST ではこの動作は、オブジェクト自身の複雑な状態変化を考えたイベントドリブンな動的モデルまで考えず、単純なフラグ操作的な処理で片付けた。より問題になったのはデータの流れであり、このデータの流れが複雑なことは、ファンクションモデルの必要性を強く訴えた。

ファンクションモデルに正確に従ったわけではないが、我々の設計では黒板を使って DFD が描かれ、話し合いが行われ、DFD を変更したり、WS 上に表示されている OMTool 上のオブジェクトモデルを変更したりという作業をすすめながら設計が進行した。

ファンクションモデルが重要な場合に、それをオブジェクトモデルとうまく結合させるようなツールの必要性も感じたことを付け加えておく。

また、GIST 自身の問題ではないが、GIST はアプリケーションを構築するためのライブラリおよび、アプリケーションが作るクラスのスーパークラスとなるものを定義しており、アプリケーションの動作に依存する多くの部分はアプリケーションまかせである。GIST 自身、EIL 言語によってアプリケーションで動的モデルを構築しやすい環境を整えており、これは今後のアプリケーション開発への応用による評価を待ちたい。

5 GISTを利用したシステム

当社のシステム技術部では GIST を利用して、プラント制御システムのための画面構築用グラフィックビルダ 3 種類を開発した。これらはやはり百以上のクラスから成る大きなシステムであるにもかかわらず、オブジェクト指向や C++は初めてのエンジニアばかりで大きな困難もなく設計を行うことができた。これには以下の理由が考えられる。

- GIST での経験をつたえることにより、ノウハウが伝わった。クラス設計段階でのコンサルティングは十分に行った。
- OMTool の利用により、オブジェクト指向の概念に馴染みやすかった。
- GIST のクラスのサブクラスを作る形式で設計されたため、「何をオブジェクトにするか」という問題に真正面から取り組まずに済む場面が多かったし、GIST のクラス関係図を OMTool から知ることによって、オブジェクト指向の設計例を学んだ。

6 オブジェクト指向による開発、OMT 法について

オブジェクト指向による開発を今回行ってみて、オブジェクト指向開発のパワーを実感した。複雑で大規模なソフトウェアといえる GIST をしかもグループによる分担作業で行うためにオブジェクト指向のパラダイムを用いなければ、より慎重で時間のかかる設計やモジュール分けが必要になったことであろう。しかしながら今回の開発を通じて、オブジェクト指向や OMT 法のさまざまな問題点があることも明らかになった。ここでは筆者らが感じた点を列挙する。

- オブジェクトモデルの強力さ。
  頭の中にあるオブジェクトの概念をオブジェクトモデルに表現することは、我々にとって自然に行えた。またできたモデルによって、システムの構成がわかりやすくなったことも挙げられる。
- 動的モデルの有用性。
  GIST 自身の設計にはあまり利用されなかったが、GIST のアプリケーションを開発する際に必要になった。ただ、どのオブジェクトについて動作を考えるかという点が今一つあいまいである。GIST では Subject や View などのオブジェクトに情報を伝えたり、操作を施すコントローラというオブジェクトを定義してその振舞いを動的モデルで定義するようになっているが、一般的には、インテリジェントで能動的なオブジェクトであるいわゆる'エージェント'的なオブジェクトに対して動的モデルを考えればよいのであろうか。
- 汎用クラス開発のむづかしさ。
  ツールキットのようなアプリケーションのためのスーパークラス、あるいは、汎用のクラスを作る場合には、相当慎重でなければならないことを痛感した。GIST のユーザがサブクラスを作るとき、親である GIST のクラスの振舞いが希望通りでない場合に、C++の場合、メソドが virtual でなかったり、親クラスのメンバが private のためアクセスできなかったりという問題は頻繁に起こった。ここで問題なのは、そういった不都合が起きたことよりも、不都合を解決するために、親クラス自身を変更しなければならない事態が生ずることである。親クラスの変更は他のすべてのサブクラスに影響してしまう。
- 深いクラス階層の問題点。
  厳密に問題を考えすぎるとクラス階層は非常に深くなり、メソドの実装もクラスに応じて細やかに対応できる。継承やポリモーフィズムはオブジェクト指向における強力な武器として導入され、実際の設計／実装に役立っている。しかしながら [5] をはじめ、多方面で議論がなされているように、深いクラス階層におけるこれらの多用はかえって混乱を招くだけであり、ソフ

トウェアの内容をソースから理解するのは大変困難になる。これはデバッグ／保守フェーズに大きな工数を要求することになる。

7 OMToolの評価

我々はGISTのプロジェクトを１９８９年にスタートさせた時点で、C++コード自動生成機能のない初期版のOMToolができており、このツールを使ってみることになった。それまでOMT法は紙の上に鉛筆で図を描くことによって使われる、「考えるための手助け」としての教科書のひとつであったが、OMToolが出現した時点で手法そのものが開発支援になり、問題点の分析からグループによる設計まで非常に有用であることが確認できた。さらにC++のソースコード生成機能により、もはやC++のヘッダファイルをエディタで手書きする必要が全くなくなり、OMToolユーザはソースを書く目的においては、C++のクラス定義に関する詳細な文法をマスターする必要がない。

ここではオブジェクトモデルによる分析／設計、およびコード生成による実装支援という２つの場面にわけてOMToolの筆者らの評価を行う。

7.1 分析／設計支援ツールとして

- OMT法に準拠しており、その方法論を学ぶこと／学んだあとの実践共に最適。
- レスポンスよいユーザインタフェースは思考を妨げない。
- MotifのLook&Feelに準拠しており、なじみやすい。
- モデルの他モジュールへのコピーができない。
- 日本語化がされていない。

7.2 実装支援ツールとして

- コメントがソースに反映されるのはよい。
- emacsとつなげて開発でき、変更の反映も自動なので設計とコードの情報のずれがなく、安心。
- モジュール分け、モジュール統合を自動的にできない。
- 生成されたC++のボディ内で、メソドの順序がばらばら。
- C++以外の言語への対応が遅れている。

8 まとめ

GISTおよび、GISTを利用したアプリケーション開発において、OMT法およびOMToolの有用性が確認できた。設計論を支援するツールとしてOMToolを見たときの有用性は勿論であるが、OMT法そのものには無関心な場合にOMToolをC++プログラミング支援ツールとして見た場合でも、OMToolの強力さは明らかであった。すなわちクラス階層図からソースコードを自動生成し、それをもとにOMToolと通信でつながったemacsで実装をすすめるパラダイムに一度慣れてしまうと、ヘッダファイルとソースボディの矛盾に気をとられることなく安心して開発できる。我々のグループ以外でもC++で開発をしているところはOMToolの利用が普及しつつある。

[参考文献]

[1] J. ランボー、他. オブジェクト指向方法論OMT:トッパン 1992

[2] L.J. ビンソン、R.S. ウィナー. Smalltalk:オブジェクト指向プログラミング:トッパン 1990

[3] M.A. Linton, et al. Composing User Interface with InterViews.: Stanford UniversityよりFTPで入手。(1988)

[4] P. Coad, E. Yourdon. Object-Oriented Analysis—second edition:Prentice Hall 1991

[5] N.Wilde, R. Huitt. Maintenance Support for Object-Oriented Programs: IEEE Transactions of Software Engineering. Vol 18, No.12 December,1038-1044