ソフトウェア工学 101－1
（１９９４．　１１．　１８）

構造化オブジェクトモデリング環境ＳＯＭＥ

－ＳＯＭＭに基づくＯＯＡ／ＯＯＤ－

原田　実　　澤田　隆史　　藤沢　照忠

青山学院大学

ＯＭＴをはじめとする既存のオブジェクト指向分析／設計方法論では、実規模システムに適用した場合に設計図が巨大化／複雑化してしまうという問題点が指摘されはじめている。我々は、この問題を解決するために設計図を階層的に構造化した表記法を用いるオブジェクトモデリング技法：ＳＯＭＭ（Structured Object Modeling Method）の提案と、このＳＯＭＭを支援する構造化オブジェクトモデリング環境：ＳＯＭＥ（Structured Object Modeling Environment）の開発を行った。本稿では、このＳＯＭＭ及びＳＯＭＥの概要について述べる。

Structured Object Modeling Environment:SOME

- OOA/OOD based on SOMM -

Minoru HARADA　Takafumi SAWADA　Terutada FUJISAWA

Aoyama Gakuin University

There are some problems in applying usual object oriented analysis/design (OOA/OOD) methodologies such as OMT to an actual large-scale system development. The most serious problem is that its design schema becomes too huge and complex to be displayed on a screen or to be printed on a paper .

To solve this problem we propose a "Structured Object Modeling Method : SOMM", which adopts a hierarchically structured design schema, and also have developed "Structured Object Modeling Environment : SOME" supporting SOMM. In this paper, we describe the outline of SOMM and SOME.

1　はじめに

ＯＭＴ[1]をはじめとする既存のオブジェクト指向分析／設計方法論では、実規模システムに適用した場合に設計図が巨大化／複雑化してしまうという問題点が指摘されはじめている。本研究では、この問題を解決するためにオブジェクト図やイベントトレース図などを階層的に構造化した表記法を用いるオブジェクトモデリング技法（Structured Object Modeling Method：ＳＯＭＭ）を提案し、このＳＯＭＭを支援する構造化オブジェクトモデリング環境（Structured Object Modeling Environment：ＳＯＭＥ）を開発した。このＳＯＭＥは単なる作図支援ツールではなく、設計対象を表現する全設計図を統合管理する設計ベースシステムである。

2　ＳＯＭＭの基本的な考え

オブジェクト指向分析／設計（ＯＯＡ／ＯＯＤ）では、設計対象に必要なクラスの構造や振る舞い、クラス間の関係等を記述してゆくことが中心となる。ＯＭＴをはじめとする方法論の多くは、これらの情報を図（グラフ）によって表現するものがほとんどである。

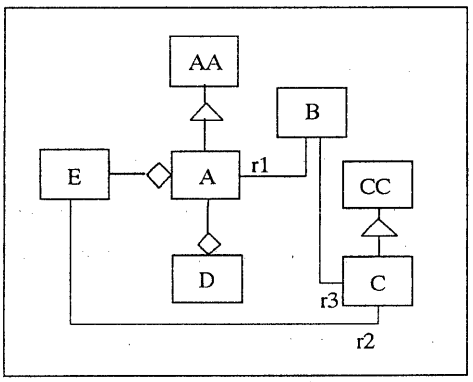

図１　ＯＭＴのオブジェクトモデル

ＯＭＴのオブジェクトモデルを例に取ると、図１のようにクラスがグラフを構成するノードとなる。実規模システムではクラス数が１００を越えることもあり、これを表現するためにはノード数１００以上の巨大なグラフを扱う必要がある。又、クラス間の継承、集約、関連など異なる種別のクラス関係が、ノードを結合するアークとして表現されるので、たとえそれらが異なる形状のアークであっても設計対象の大規模化に伴い、アーク数が激増してしまう。従って、ＯＭＴのオブジェクト図は、その豊かな設計記述能力が仇となり、かえってグラフ表現の一番のメリットである「設計情報の直感的理解」を減少させているといえる。

ＳＯＭＭでは上記のようなグラフの複雑化を解消するために、以下の方法を用いる。

①設計図内のノード数を減少させるために、階層的に設計図を分割する。

②設計図内のアーク数を減少させるために、１つの設計図に使用するアークの種別を１種類に限定する。

なお、①における設計図の分割方法は、設計図の直感的な理解を促進させるよう、クラス間の集約関係を階層関係とした。詳細は後述するが、これにより単に設計図の視覚性を向上させるだけでなく、クラスがそのまま設計単位となるため、自然なモデリングを行え、さらに実装工程へのスムースな移行を可能にする。

3　設計図の構造化

3.1　再帰グラフ

通常、システムをグラフで表現する場合、そのシステムのサブシステムをノードで表し、それらの間の関係を全てアークで表す。しかし、

このノードも構造や性質を持つシステムであり、サブシステムを持つ場合がある。このような、ノードやアークが再びノードの中に現れる「再帰構造」を表現するのに、ノードの内と外とを接合するインタフェースを持つ再帰グラフ[2]を用いる。これは、設計図の諸概念の意味を形式的に定義するためである。なお、再帰グラフ上のノードにそのサブシステムを表す別の再帰グラフを埋め込む操作をズームイン操作と呼び、これを設計プロセスの形式化に用いる。

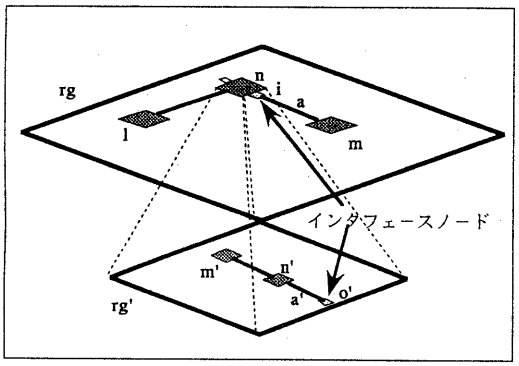

図2　再帰グラフ

3.2　クラス間の集約関係

クラスは幾つかの構成要素の集まりであるとみなすことができる。そして、その構成要素もまた何らかの型（クラス）[1]をもつ。このようなクラス間の集約関係は、クラスが構成要素をサブシステムにもつ再帰構造となることから、再帰グラフによる表現が有効である。

ＳＯＭＭでは、クラス構造等の静的な情報を表現するためのオブジェクト図と、クラス間のイベント送受を表現するイベントトレース図に再帰グラフを用いる。つまり、設計対象を表現する全てのクラス毎にオブジェクト図、イベントトレース図を作成し、各設計図のノードをそのクラスの構成要素とすることで、設計図の巨大化／複雑化を回避するわけである。また、設計図内のノード数は設計対象の規模には左右されず、クラスの規模を反映するものであるため、万一設計図が巨大化するような場合はクラスの肥大を示すものであり、そのクラスの更なる抽象化の必要性を教えてくれるのである。

1　本論分では、特にことわりのない場合、「型」と「クラス」は同等の意味を持つとみなし、特に区別せずに使用する。

4　ＳＯＭＭ

ＳＯＭＭでは、設計対象の分析／設計視点として以下の４モデルを提案する。

●派生図（Derivation Diagram:DD）
クラスの定義とその派生関係を表現する。

●オブジェクト図（Object Diagram:OD）
クラスの構成要素の定義と、その構成要素間（内外インタフェースを含む）の関連を表現する。

●イベントトレース図（Event Trace Diagram:ETD）
クラスの構成要素間（内外インタフェースを含む）でのイベントの送受信と、構成要素間へのイベントの送信を表現する。

●状態遷移図（Statc Transition Diagram:STD）
クラスのライフサイクル内での状態遷移を表現する。

4.1　派生図（ＤＤ）

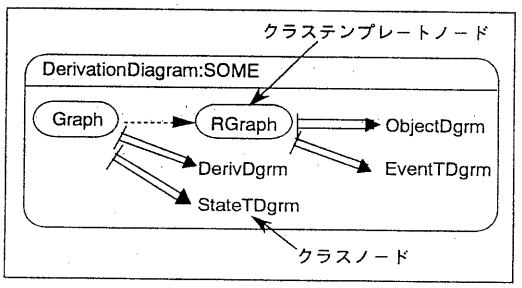

図3　派生図

図３のようにＤＤでは設計対象に必要とされる全クラス名をノードとして登録する。この時、C++を開発言語として意識している場合、通常のクラス以外に図中で○型のノードで示すクラステンプレートを指定できる。クラステンプレートは、C++で言うところのテンプレートを表すもので、その構成要素の型として実体的なクラス名ではなくパラメタ化されたクラス名を用いることができるものである。[1] これにより設計情報の再利用性を高めることができる。

ＳＯＭＭでは、図４のようにＤＤ上でノードとして表わされたクラスに対し、ＯＤ、ＥＴＤ、ＳＴＤの３つの視点から詳細設計を行う。これによりＯＤ、ＥＴＤ、ＳＴＤは設計対象を表現するクラスの個数だけ作成することになり、個々の設計図は複雑になることはない。

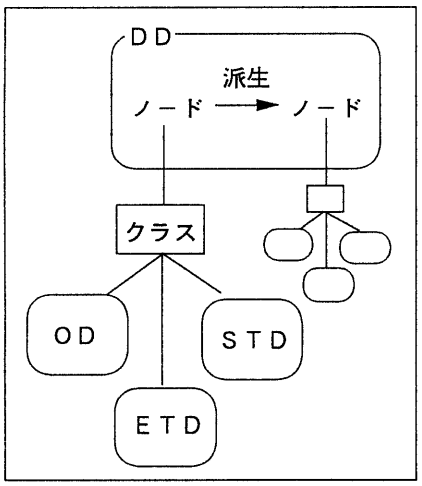

図４　ＳＯＭＭの概念図

ＤＤ上でクラス間の派生関係はアークで表わされる。ＤＤでは、派生を厳密な継承だけでなく、基本クラスと良く似た構造を持つ派生クラスを作成することであると拡大解釈し、表１のように５種類の派生を用意し、これを表現する。なお、ＤＤは他の３つの設計図と違い、プロジェクトに１枚のみ作成するものであるが、クラス間の派生関係は木構造となるので、クラス数が膨大であっても可読性を損なうことはない。

表１　派生の種類

派生の種類	表記法	Ｃ＋＋での意味
継承	A→B	class B : A {.....};
テンプレート化継承	A→(B)	template < class T > class B : A {.....};
テンプレート継承	(A)→(B)	template < class T> class B : A<T> {.....};
カスタム化	(A)⇒B	T→CCの時 #define B A<CC>
カスタム継承	(A)⇒B	T→CCの時 class B : A<CC> {.....};
展開	A (A) ⋯→ B (B)	コピー後に編集

クラスを派生させることは、派生クラスが基本クラスの構造を受け継ぐことであり、具体的にはＯＤ、ＥＴＤ、ＳＴＤを受け継ぐことである。しかし、派生の種類によってはコピーされた設計図への編集作業に制約が加わる。例えば、「継承」において派生クラスは、基本クラスの特徴を引き継いでいるものの、派生クラス独自の属性や操作を追加できるが、「カスタム化」においては、生成されるテンプレートクラスはクラステンプレート内のパラメタを実体に置き換えるだけであることから、属性や操作の追加は行えない。

４.２　オブジェクト図（ＯＤ）

ＯＤではクラスの属性、構成（集約）要素、構成要素間の関連を図５のように表す。この構成要素は、任意の型を持つインスタンス変数であり、四角いノードで表現する。また、整数

1　クラステンプレート定義内のパラメタに実体のあるクラス名を当てはめ生成されるクラスをテンプレートクラスと呼び、クラステンプレートと区別する。

型、文字列型などのプリミティブな型をもつ要素は、属性であるとみなし、構成要素とは区別して図の右側に列挙する。

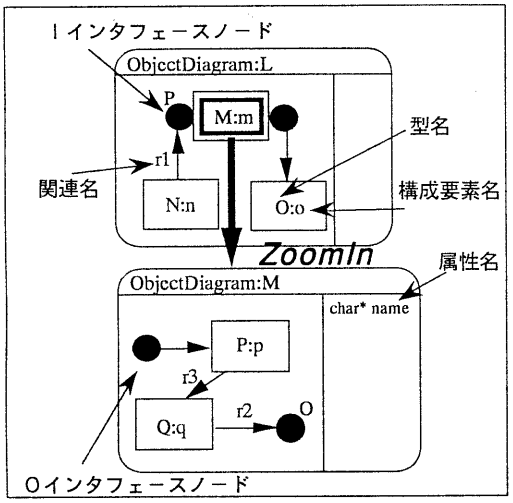

図5　オブジェクト図

構成要素が、同一スコープ（グラフ）内に存在しないクラス、すなわち同じ親の構成要素でないクラスと関連を持つ場合は、インタフェースノードを用いて表現する。インタフェースノードにはグラフの外とのインタフェースと、ノードの内とのインタフェースがある。

例えば、図５において、クラスＬの構成要素であるＮ型のｎは、同じく構成要素であるＭ型のｍのスコープ内の（つまりクラスＭの構成要素である）Ｐ型の構成要素と関連ｒ１を持つということを表現している。

構成要素は型を持つ以上、その型のオブジェクト図をサブシステムに持つ。従って、ズームイン操作を行い、その整合性を確かめなければならない。ズームイン操作は、設計対象を表現する全クラスの構成要素に対して順次適用して行き、構成要素を持たない属性のみを持つクラスに行き着くまで続けられる。これが再帰グラフの展開の終了条件となり、意味的には一枚の大きな設計図を作成したのと同様となる。

４.３　イベントトレース図（ＥＴＤ）

図６のように、ＥＴＤではＯＤで定義した構成要素間での、イベントの送受をアークで表現する。構成要素は、棒状のノードで表す。また、クラスの構成要素へのイベント送信を表現するために、このクラス自身を表わすセルフノードを左端に記述する。例えば図６中のｅ１は、ａの型Ａのある任意の構成要素から、クラスＣに対して送信されるイベントを表している。

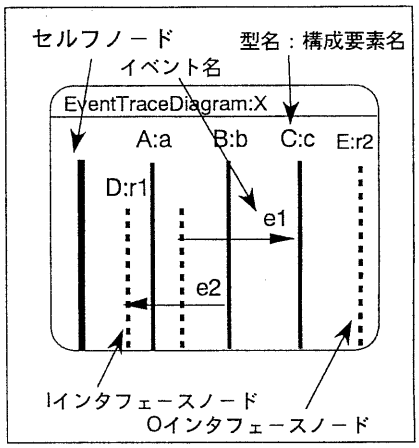

図６　イベントトレース図

ＥＴＤのノードはＯＤのそれに依存するため、インタフェースノードが必要になる。従ってＯＤと同様の理由で、各ノードに対してズームイン操作を行わなければならない。

４.４　状態遷移図（ＳＴＤ）

ＳＴＤでは、図７のようにクラスの状態をノードで表わし、状態を遷移させるイベントをアークで表わす。また、受け取っている（Received）イベントと、送っている（Sending）イベントが、クラス内部のＥＴＤとズームイン先のＥＴＤから得られるので、これを右端のフィールドにまとめる。なお、ＳＴＤで扱うイベントは全て他のクラスから送られて

くるものであるため、ＳＴＤは再帰グラフである必要性はなく、通常のグラフとなっている。

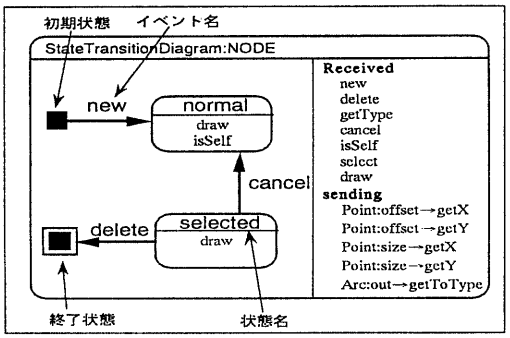

図７　状態遷移図

また、各状態の時にどのReceivedイベントを受け付けるのかを明示するために、状態ノードの中にそのようなイベント名を記述する。また、状態を遷移させるイベントについては、アークのラベルとする。さらに、全てのReceivedイベントについて発火時のアクション、戻り値の型、引数の型などの詳細定義をイベント仕様書として作成する。このアクションの候補として、他のクラスへのSendingイベント、あるいは、自分自身へのSendingイベント（これはReceivedイベントとして表現されている）が候補になる。なお、ＳＯＭＥではこのイベントの詳細定義を行うために、図８のようなダイアログを用意している。例えばこの図では、イベントisSelfはBool型の返り値、Point型の引数を持ち、その発火時のアクションとしてPoint型のoffsetにイベントgetX等を送っていることを表している。

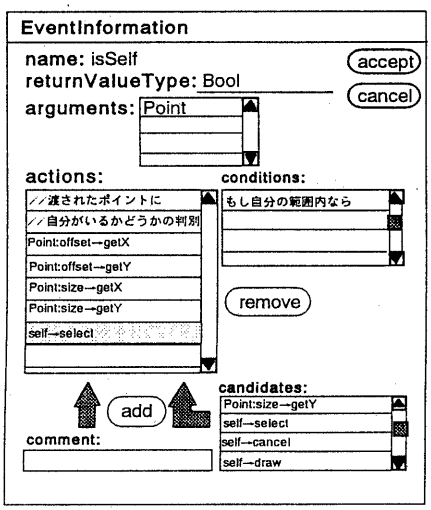

図８　イベント定義ダイアログ

５　ＳＯＭＥ

ＳＯＭＥは、ＳＯＭＭの４モデルの作図支援を行うだけでなく、プロジェクト全体として設計に矛盾をきたさない様、全ての設計図を統合的に管理する設計ベースシステムである。ＳＯＭＥはオブジェクト指向データベース（ＯＯＤＢ）により実装された設計ベース部と、グラフエディタ、ダイアログ等からなるインタフェース部に大別できる。

５.１　ＳＯＭＥの設計ベース

ＳＯＭＥの設計ベースは図９のような３つの物理的なストレージ階層をもつ。すなわち、設計ベース全体にあたるＳＯＭＥ層（フィデレートデータベース）の中では設計対象別にプロジェクト層（データベース）に設計情報をまとめ、更にプロジェクト層の中ではＤＤ、ＯＤ、ＥＴＤ、ＳＴＤの各設計図毎にグラフ層（コンテナ）に設計情報を格納する。

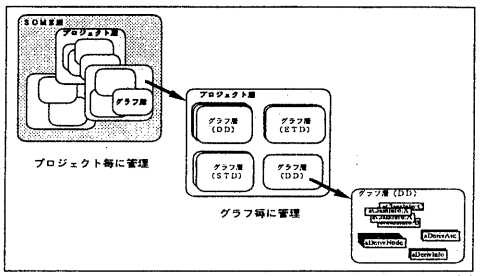

図９　ＳＯＭＥのストレージ階層

ＳＯＭＥの設計ベースはＯＯＤＢを利用している。ＯＯＤＢでは、格納されるデータは全て属性や操作をもつオブジェクトであるため、設計ベース内に生成されたオブジェクトの内、論理的に緊密な関係にあるもの同士を図１０のように双方向関連を持たせている。例えばこの図では、ＯＤとＥＴＤにおけるノードはそれぞれ全く同じ実体を異なる視点から表現しているだけであり、一対一に対応するものであることから、ObjectDgrmのObjNode型の構成要素と、EventTDgrmのEventTNode型の構成要素の間に双方向関連を持たせている。

これによりデータであるオブジェクトに対して何らかの処理を依頼した際に、自発的にその影響圏にある他のオブジェクトと互いに情報の参照・更新等を行うことを可能にしている。すなわち、個々のデータが互いに監視しあうことにより、全体設計を矛盾のない頑強なものとしているのである。具体的には次節に述べるような効果を期待できる。

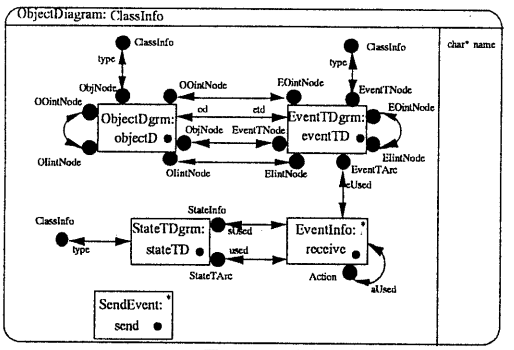

図１０　双方向関連の例（ClassInfo）

５．２　設計情報の統合管理

前述したようにＳＯＭＥは設計ベースに蓄えられた設計情報を統合的に管理している。その際の具体的な処理として以下のものが挙げられる。

①ズームイン操作におけるインタフェースの整合性検査
：ＳＯＭＭにおいて形式的に定義されたズームイン条件に基づいて型名、アーク数等の整合性検査を行い、論理的な矛盾を発見した場合はユーザに警告を発処理を中断する。

②ズームイン後での情報の更新を拒否
：ズームインは設計の整合性を確かめるものであることから、すべての処理（ノード、アーク等の追加、削除、変更など）は拒否される。

③ＯＤの情報からの、ＥＴＤノードの自動生成／削除
：ＯＤで必要とされる構成要素は、対応するＥＴＤにおいても必要とされることは自明であることから、ＥＴＤノードの生成／削除はＯＤのそれに連動して行われる。

④ＥＴＤにおけるイベントアーク生成時の関連の有無チェック
：アクセスパスを持たないオブジェクトに対してはメッセージを送ることはできないので、これを検査し論理エラーを防止する。

⑤レシーバクラスでのイベント情報管理
：イベントは、そのレシーバクラスのＳＴＤにおいて使用されるので、このイベントをレシーバクラスの管理下に置く。

⑥ＥＴＤ、ＳＴＤでのイベント情報の共有
：ＥＴＤにおけるイベントの削除が、このイベントに関係する全てのＳＴＤに伝えられる。

⑦派生指定時のＯＤ、ＥＴＤ、ＳＴＤコピー
：派生アークの種別により、基本クラスから派生クラスへ設計情報を編集複写する。例

えば「継承」においては、基本クラスの持つＯＤ（ＥＴＤ、ＳＴＤ）のグラフ構造等を派生クラスのそれに追加し、「カスタム化」においては前述の処理の過程で基本クラス内のパラメタを実体に置き換えるためにユーザにクラス名の入力を要求し、この情報を用いて派生クラスのグラフを生成する。

⑧派生クラス、基本クラスでのイベント情報の共有／非共有
：イベントのオーバロードを指定可能。

⑨派生クラスの各設計図に対する操作制限
：４．１参照。

⑩派生後の基本クラスへの操作をロック
：派生後に基本クラスを変更する場合、この変更点を下位クラス全てに伝える必要があるが、現時点では派生後の基本クラスへの全ての変更操作を拒否することによってクラス間の整合性を保っている。

5.3　ＳＯＭＥのインタフェース

ＳＯＭＥは専用のグラフエディタをもち、このエディタからＳＯＭＭ形式で設計ベースへ設計情報を入力する。このグラフエディタはユーザからのイベントを検知し、設計ベース内に存在する特定のグラフオブジェクトに伝える「コントローラ」の機能と、キャンバスに四角形、円、線分等のプリミティブな図形を表示する「ビュー」の機能を兼ね備えている。すなわち、抽象クラスGraphのサブクラスである任意の永続グラフオブジェクト（ex. ＤＤ、ＯＤ、ＥＴＤ、ＳＴＤ）を「モデル」とすることのできる汎用グラフエディタである。これにより、拡張性、ソースレベルでの一貫性、統一的な操作環境を提供している。

6　おわりに

ＳＯＭＭの提案ではオブジェクト図、イベントトレース図を階層的に構造化したことにより、設計図の視覚性を向上させることができた。また、ＳＯＭＥの開発により、プロジェクト全体を通しての設計から矛盾をなくすことができ、またＳＯＭＭの長所を１００％引き出すことができた。今後はＳＯＭＥで行った設計から、プログラムを自動生成することを考えている。

謝辞

本研究の一部は東芝エンジニアリング㈱からの受託研究費を基に行われました。同社技術本部の辻井さん、岩田さん、竹内さんに感謝いたします。

＜参考文献＞

[1] Ｊ．ランボーら（羽生田栄一監修訳）、「オブジェクト指向方法論ＯＭＴ」、プレンティスホールトッパン、1992

[2] Minoru Harada, Tosiyasu L.Kunii : "A RECURSIVE GRAPH THEORY",IEEE WORKSHOP ON VISUAL LANGUAGES,1984,pp.124-135,Hiroshima.

[3]Coad,P.,Yourdon,E.,『Object-Oriented Analysis』, Prentice Hall,1991.

[4]Grady Booch,『Object-Oriented Design With Applications』,Benjamin/Cummings,1990.

[5]Chen,P.,『The Entity-Relationship Model - Toward a Unified View of Data』,ACM Trans.Database Systems,Vol.1,No.1,pp.9-36,1976.

[6]B.Henderson-Sellers,『A Book of Object-Oriented Knowledge』,Prentice Hall,1991.