多重視点の問題領域におけるオブジェクトの組織化過程

中谷多哉子 † 玉井哲雄†

複数のアプリケーションプログラムがオブジェクトを共有する問題領域をモデル化した場合，個々のアプリケーションで観察されるオブジェクトの属性や振る舞いが異なることがある．

本稿では，複数の視点から観察される問題領域を多重視点の問題領域と呼び，多重視点の問題領域をモデル化する過程を組織化過程と呼ぶ．組織化過程は，個々の視点に基づいてモデルを構築した後，それぞれのモデルを統合して全体のモデルを構築する過程を経る．統合時には，各モデルが衝突を起こしている部分を発見し解消する．このとき，オブジェクトを定義し，衝突を発見するためのツールとしてオブジェクト辞書を用いる．利用者ごとのモデルを構築する際には既存の方法論を活用できるので，本稿では，統合時に問題となる衝突の発見と解消について議論する．

The Integrating Process of Objects in Multiple Viewpoint Domain

Takako NAKATANI † and Tetsuo TAMAI†

When many application programs share an object, the object behaves in different ways in a viewpoint of each application user.

We call a domain which is seen from many viewpoints, a multiple viewpoint domain, and we regard the modeling process of the multiple viewpoint domain as an integrating process. The integrating process proceeds as follows: first, constructing models based on each user viewpoint of applications and then integrating those models into the whole domain model. While the model has been integrated, conflicts between the models must be resolved. We define an object dictionary for specifying objects and use it to extract conflicts between objects.

1. はじめに

オブジェクト指向システムの規模が大きくなるにしたがって，分析の重要性も増している．大規模システムでは，アプリケーションプログラムが，データベースなどに格納されたオブジェクトを共有することが多い．アプリケーションプログラムから共有されるオブジェクトとは，現実世界において，複数の利用者から観察される対象である．ここで，複数の利用者から観察される問題領域を多重視点の問題領域と呼ぶ．

多重視点の問題領域を対象とするシステムでは，分析を行う担当者が一人であるとは限らない．分析者が異なると用語の統一の問題が発生することもある．また，Harrison らが指摘しているように，個々のアプリケーションシステムの視点から分析したオブジェクトの振る舞いが，視点ごとに異なって見えるという問題も発生する[4]．

† 東京大学大学院総合文化研究科広域科学専攻
Graduate School of Arts and Sciences,
University of Tokyo
e-mail: {tina, tamai}@graco.c.u-tokyo.ac.jp

OMT 法や Coad/Yourdon 法をはじめとした多くの方法論[2),3),9),17)]は，分析の視点についてはあまり議論していない．Jacobson の Objectory[7)]にはアクターという分析の視点が存在するが，モデル構築に直接影響を与えるのはアクターではなく，アクターごとに定義したユースケースである．

データベースの分野では，スキーマを定義する際に，モデルを視点ごとに定義した後，競合状態を解消しながら統合するという手法が研究されてきた[1),20)]．我々は，多重視点の問題領域におけるモデル構築のための作業過程をオブジェクトの組織化過程と呼ぶ．多重視点の問題領域のオブジェクトのために，状態モデルの統合を行う方法が文献 5) で議論されている．各視点で観察された状態モデルは，現実世界のオブジェクトの状態モデルの一部しか観察されないため，オブジェクトモデルを定義する際には，すべての状態を満足するような状態モデルを構築する必要がある．

多重視点の問題領域に関するこれらの議論は，いずれも比較対象のオブジェクトが同一の事物や事象を指していると識別されることが前提となっている．統合時に衝突を解消する際には比較対象の選出も問題とな

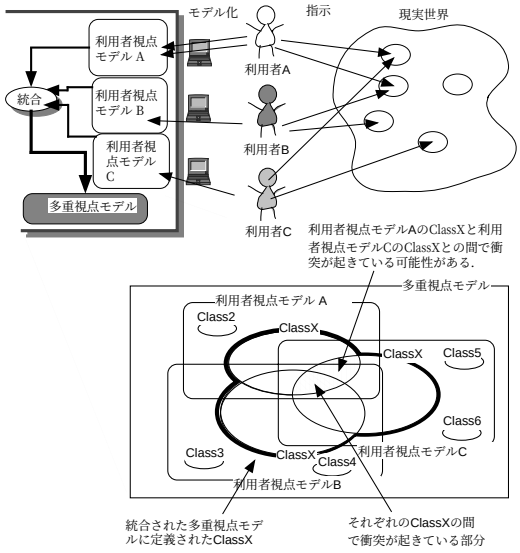

図 1 モデル間の関係
Fig. 1 Relationship between models

る．オブジェクトは，その属性，振る舞い，関連といった性質によって，形式的に同一であることを確認できる場合もある．しかし，多重視点の問題領域では，形式的に同一であっても現実世界で意味している事物が異なることもある．また，同じ事物をモデル化しても形式的に同一となるとは限らない．だから，形式的な表現だけでは統合のための同一オブジェクトの抽出は困難である．そこで，多重視点の問題領域では，オブジェクトが指し示している現実世界の事物や事象に対して，定義の集合を与えなければならない．これを指示という[12]．

多重視点における組織化過程は，従来のオブジェクト指向分析方法論の前後に 2 つの作業を追加して構成されている．まず，方法論を適用する前に，モデル化されたオブジェクトと現実世界を繋げる指示の定義を行う．そして，方法論を用いて視点に対応するオブジェクトモデルを定義した後，モデル間の矛盾を発見し，解消してモデルを統合する．

本稿は，2. で，多重視点の問題領域にあるオブジェクトの組織化過程を説明し，3. で組織化の例を示す．また，4. では組織化の手法に関する議論を行う．最後に，5. で，多重視点に関する他の研究を紹介する．

2. 組織化過程

2.1 概　　要

利用者の各視点で定義するモデルを利用者視点モデルと呼び，それらを統合したモデルを多重視点モデルと呼ぶ．現実世界，利用者視点モデル，多重視点モデルの間には**図 1** に示す関係がある．

現実世界と利用者視点モデルの間に存在する記述が指示である．指示に基づいて利用者視点モデルを構築した後，利用者視点モデル間の矛盾を発見して解消する．矛盾を解消したモデルを統合すると，多重視点モデルを定義できる．これが組織化過程である．

図 1 では，利用者 A，利用者 B，利用者 C の利用者視点モデルが定義した *ClassX* の振る舞いに矛盾が起きている．各利用者の *ClassX* を統合したクラスは，図 1 中の，太線で囲んだ 3 つの円の結びで表現される．統合したクラスでは，各利用者の *ClassX* の間で起きていた矛盾が解消されている必要がある．

2.2 指　　示

指示とは，次の形式を持つ．

- 認識規則 ≈ 指示された用語
- 例：x はソフトウェアを開発する人 ≈ 開発者 (x)

指示には，問題領域の中で認識される現象と，認識するための規則を記述する[12]．指示によって，その問題領域の分析者であれば，オブジェクト名に対応する問題領域のオブジェクトを限定できるようになる．

2.3 衝突の種類

利用者視点モデルを統合する際，発見すべきモデル間の衝突として，次の 3 つを考慮する．

- 名前の衝突
- 継承の衝突
- 構造の衝突

2.3.1 名前の衝突

異なる利用者視点モデルで，異なる現実世界の事物や事象を指示していながら，同じ名前が使われている，あるいは逆に，同じ現実世界の事物や事象を指示していながら異なる名前が使われているといった，モデル間の矛盾を名前の衝突という．

名前の衝突は，クラス名の他に，属性名やサービス名にも起こりうる．クラス名の衝突は，クラスが指示している現実世界の事物や事象を比較して発見する．属性は，本来オブジェクトのカプセル内に隠蔽されている情報であり，オブジェクトの責任を実現するために定義されるデータと解釈することもできる．そこで，属性の情報については，サービスに関する衝突が解消されてモデルが統合された後に，個々のクラス内で検討することもできる．よって，属性名の衝突は組織化過程では検討しない．サービス名の衝突は，責任駆動型設計手法[19]に基づく契約の情報が衝突を検討する時の比較対象となる．

OMT 法[17]の表記を用いて，**図 2** に名前の衝突の状態を示した．図で (a) および (b) はそれぞれ異なる視

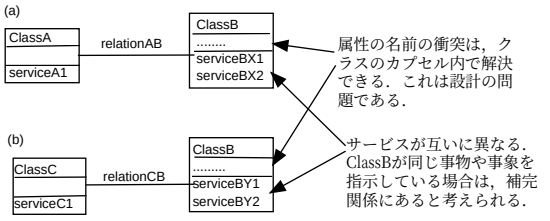

図 2 名前の衝突
Fig. 2 Name Conflict

点で定義されたモデルである．両者には *ClassB* が定義されているが，*ClassB* に定義されている振る舞いは互いに異なっている．

(a) の *ClassB* と (b) の *ClassB* との間に考えられる関係として，両者は同一のオブジェクトを指示しており，したがって，両者を合成したクラスが多重視点モデルの *ClassB* となる場合，両者の間に継承関係や兄弟関係が存在する類似クラスである場合，名前が同じだったのは偶然であり，意味的に別のクラスである場合などが考えられる．

衝突の種類と解消方法は，(a) と (b) に定義された *ClassB* が現実世界の何を指示しているかによって異なる．(a) と (b) とで，両者が同一のオブジェクトを指示していると判断できたら，次は各サービスに名前の衝突が起きていないかを検査する．たとえば，2 種類の *ClassB* に，同じサービスを意味していながら異なる名前が使われているといったサービス名の衝突が発生している可能性もある．また，2 種類の *ClassB* が補完関係になっている場合も考えられる．この場合は，名前の衝突というよりは，構造の衝突と呼んだ方がよい．構造の衝突については後で詳細に触れる．

2 種類の *ClassB* が異なるオブジェクトを指示している場合は，クラス名に名前の衝突が発生しているので，指示内容を適切に表現できる名前を定義し，変更する．

2.3.2 継承の衝突

複数の利用者視点モデルで定義された継承が異なる場合を継承の衝突という．

クラスの継承構造もクラスの情報隠蔽の対象となる．継承の衝突を解消するには，各利用者視点モデルに定義された具象クラスを集めて分類し，クラスの抽象化を行って継承構造の整理を進める．継承の衝突を解消する際の目標は，各利用者視点モデルで定義された具象クラスの仕様を過不足なく定義することである．したがって，新たな抽象クラスを定義した後，もう一度具象化を行ってクラスの定義を調整しなければならない．

2.3.3 構造の衝突

異なる視点で定義された 2 つのクラスにおいて，それらが指示している事物や事象が一致するにもかかわらず，クラス間の関係が一致しない，振る舞いが一致しないといった問題が発生した場合，これらのクラスの間に構造の衝突が起きているという．

構造の衝突を発見した際は，補完関係にあるか，一方の構造を他方の構造で置換可能かを見極める．オブジェクトは，他のオブジェクトが保持する関連によって構成される外部構造と，自分自身が保持する内部構造とを持っている．内部構造は他のオブジェクトから隠蔽される構造であるから，外部構造とは分けて考えなければならない．そこで，OMT 法の表記に，外部構造と内部構造，それぞれの構造を区別する表記を追加した．追加した表記では，外部構造，すなわち他のオブジェクトから直接参照されるオブジェクト間の関連を太い線で表わし，内部構造として定義される関連を細い線で表わす．ここでは，太い線で表わした関連を実関連を呼び，細い線で表わした関連を仮関連と呼ぶ．実関連と仮関連とを分析時から区別することによって，分析の工程でクラスの情報隠蔽を考慮することが可能となる．

図 3 に，実関連と仮関連の例を示した．*ClassA* を観察している視点 X には，*ClassA* から直接 *ClassB* が見えているが，*ClassB* から *ClassD* へ至る関連や *ClassD* から *ClassE* へ至る関連は *ClassB* の内部構造である．だから，これらの関連は仮関連となる．*ClassA* の視点 X にとって重要なのは，*ClassB* が *getD:aD* や *getE(aD):aE* のサービスを提供することであるから，仮関連を実関連として定義するか否かは，*ClassB* がどのようにしてこれらのサービスを実現するかという問題である．これは，分析の問題ではなく設計時に検討する問題となる．

2.4 オブジェクト辞書

2.4.1 概 要

モデルの衝突を発見するためには，図式で表現されたモデルの情報の他に，現実世界とを結び付ける指示，オブジェクトの振る舞いを記述するための形式的な記述をモデルに与える必要がある．オブジェクト辞書は，これらの情報を記述するためのツールである．

多重視点の問題領域は視点ごとにモデル化される内容が異なるので，各分析者は個々の視点に基づいたオブジェクトの指示，サービスの契約内容を定義したオブジェクト辞書を作る．ここで，利用者視点モデルごとに定義するオブジェクト辞書を暫定版オブジェクト辞書と呼び，多重視点モデルに対応する正式なオブ

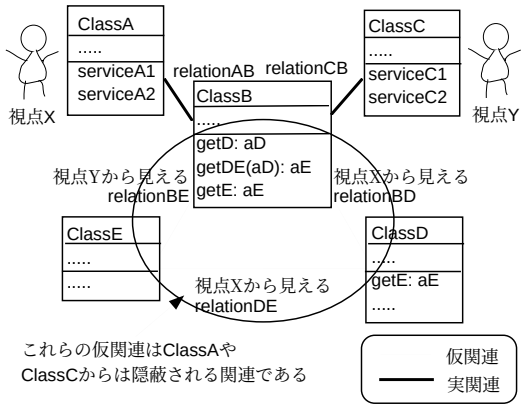

図 3 構造の衝突
Fig. 3 Structure conflict

ジェクト辞書を正式版オブジェクト辞書と呼ぶ．正式版オブジェクト辞書は，暫定版オブジェクト辞書に起きていた衝突解消後に合成して作られる．

2.4.2 辞書の構成

オブジェクト辞書を構成する項目を次に示した．

『オブジェクト名』
- 所有者名: オブジェクトを定義した視点を持つ利用者の名前
- 指示: 現実世界に対応する事物を指示して，その定義範囲を限定的に説明する文.
  認識規則 ≈ 指示されたオブジェクト名
- 継承するオブジェクト名:
- ＜責任＞
  定義しているオブジェクトが他のオブジェクトへ提供するサービスで，以下サービス数繰り返し.
  サービス名 (引き数の型の並び)
  - 事前条件:
  - 事後条件:
  - 不変表明:
  - 戻り値の型:

オブジェクトの情報は，所有者名に対応する利用者視点モデルごとに定義される．サービスの事前条件，事後条件，不変表明はオブジェクト間で締結される契約に基づく表明であり[10]，オブジェクトの責任を明示する情報である．クラスの継承構造を設計で検討する場合にも，これらの情報を活用できる[8]．

OBA[16]のオブジェクトモデル化カードには，ここで示した情報の他に属性／論理的性質の項目が付加されている．しかし，オブジェクトの属性はカプセルの中に隠蔽されるものなので，オブジェクト辞書では定義しなくともよい．

継承の衝突を発見するときは，継承するオブジェクト名の項目を比較するだけでなく，この項目で示されたオブジェクト名を辞書で参照し，指示項目を比較する．これは，スーパークラスに名前の衝突が起きている可能性もあるからである．

以上の内容が記述された暫定版オブジェクト辞書を用いて，利用者視点モデル間の名前の衝突，継承の衝突，構造の衝突を発見する．このとき，比較すべきオブジェクトを指示項目によって対応づけ，対応付けられたオブジェクトについて各項目を比較する．

衝突を発見して解消した暫定版オブジェクト辞書は利用者視点モデルを修正するために使われる．正式版オブジェクト辞書は，衝突を解消した暫定版オブジェクト辞書に登録されているクラスごとに，対応するクラスの情報の集合を合成して定義される辞書である．

2.5 組織化過程

ここまで説明してきたオブジェクト辞書を用いる組織化過程を次に整理した．

(1) アプリケーションの利用者を定義する．この利用者の視点が分析の視点となる．
(2) 分析者は担当する視点から利用者視点モデルを構築するとともに，各オブジェクトについて，暫定版オブジェクト辞書を定義する．オブジェクト辞書には，オブジェクトの指示と責任を記述する．
(3) 暫定版オブジェクト辞書で，名前の衝突，継承の衝突，構造の衝突を発見し，解消して辞書を修正する．
(4) 修正した暫定版オブジェクト辞書の内容を利用者視点モデルに反映する．
(5) 修正した暫定版オブジェクト辞書と利用者視点モデルを参照しながら，対応するクラスを合成して正式版オブジェクト辞書を定義し，多重視点モデルを構築する．

設計では，多重視点モデルに定義されている各クラスに視点を移動して，仮関連の整理や属性，継承といった内部構造を定義する．

3. 事　　例

3.1 システム概要

我々が提案する組織化過程を検証するために，ソフトウェア開発管理領域のモデル化を行う．システムの利用者として，プロジェクト管理者，開発者，構成管理者，品質監査者を想定する[6]．

3.2 利用者視点モデルの定義

図 4 と**図 5** にプロジェクト管理者と開発者の視点で定義した利用者視点モデルの一部を示した．

3.3 名前の衝突の発見と解消

名前の衝突には，同じ名前で異なるオブジェクトを

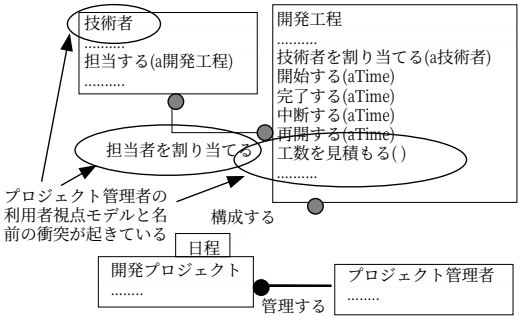

図 4 プロジェクト管理者から見える領域
Fig. 4 Domain model seen from Project Manager

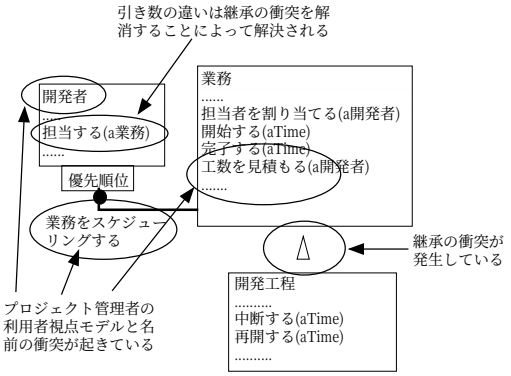

図 5 開発者から見える領域
Fig. 5 Domain model seen from Developer

表わしている場合と，同じオブジェクトに異なる名前がつけられている場合とがある．図 4 は，プロジェクト管理者がプロジェクトの開発工程をスケジューリングする部分を抜き出したモデルであり，図 5 は，開発者が，担当する業務をスケジューリングする領域について記述したモデルである．

暫定版オブジェクト辞書を以下に示す．

『開発者』
- 所有者: 開発者
- 指示:
  x はソフトウェアを開発する人 ≈ 開発者 (x)
  .....

『技術者』
- 所有者: プロジェクト管理者
- 指示:
  x はソフトウェアの開発工程を担当し，実施する人 ≈ 技術者 (x)
  .....

プロジェクト管理者の暫定版オブジェクト辞書に定義された指示の意味から，図 4 の**技術者**と図 5 の**開発者**は現実世界の同じ事物を意味していることがわかる．これは，同じ意味を表わしているオブジェクトに異なる名前がつけられた名前の衝突が起きていた例である．

次に示すのは，異なる意味に同じ名前がつけられた例である．

『業務』
- 所有者: 開発者
- 指示: x は，人によって遂行される業務 ≈ 業務 (x)
- ＜責任＞
  工数を見積もる (x:開発者)
  - 事前条件:
  - 事後条件:
  - 不変表明:
  - 戻り値の型: z : 工数

  .....

『開発工程』
- 所有者: プロジェクト管理者
- 指示:
  x はソフトウェア開発のある区切られた工程 ≈ 開発工程 (x)
  .....
- ＜責任＞
  工数を見積もる ( )
  - 事前条件:
  - 事後条件:
  - 不変表明:
  - 戻り値の型: z : 工数

開発工程のスケジューリングを行うとき，プロジェクト管理者は，企業の標準の見積り技術を使って開発工程の工数を見積る．また，開発者は，自分の担当業務をスケジューリングするために，自分自身の技術力で必要となる工数を知る必要がある．そこで，上記の開発者が定義した辞書の業務オブジェクトの責任項目には，開発者に依存した工数の見積りを行うことが定義される．したがって，それぞれの視点で，2 つのクラスに**工数を見積る**という同じ名前のサービスが定義されることとなった．継承の衝突を解消すると，**業務**と**開発工程**の間には継承関係があるので，名前の衝突を検討する必要があるし，シグネチャも合わないので，継承構造上，問題が生じる[8]．開発者の視点で定義された**工数を見積る**には，引き数が定義されているが，プロジェクト管理者の視点で定義された**工数を見積る**には引き数がない．この違いは，前者が，指定された開発者の能力に依存して工数を見積もるのに対して，後者では，標準見積もり技術に則って工数を見積もることに由来する．これらのサービスが意味する内容は異なっているから，名前の衝突が起きていると判断できる．

図 6 に，名前の衝突を解消して統合した多重視点モデルを示す．

3.3.1 継承の衝突

図 5 では，開発工程は業務のサブクラスなので，プ

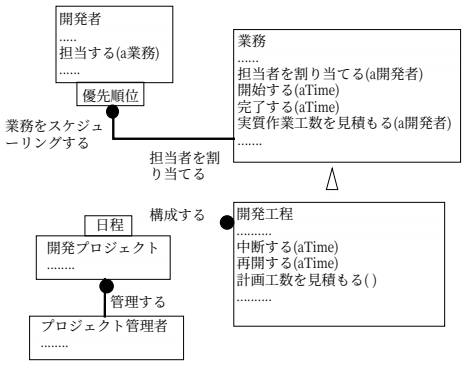

図 6 衝突を解消した結果
Fig. 6 Resolved conflict

ロジェクト管理者の開発工程との間で継承の衝突が発生する．継承の衝突を解消する際，プロジェクト管理者のモデルに定義された開発工程と技術者の間にある仮関連は，その意味から，開発者から見える開発者と業務との間の実関連に対応するので，仮関連を実関連で置き換えられる．

図 6 では，継承の衝突も解消されている．

3.3.2 構造の衝突

構造の衝突は，構成管理者と品質監査者の暫定版オブジェクト辞書の中から，**文書**と**記録**の例を示す．

次に示すのは，構成管理者と品質監査者の文書に関する暫定版オブジェクト辞書の内容である．

『文書』
- 所有者: 構成管理者
- 指示:
  x は，版管理されている成果物 $\approx$ 文書 (x)
  .....
- ＜責任＞
  参照文書を得る
  - 事前条件:
  - 事後条件: 戻り値の型を Z としたとき，$\mathbf{Z} = \{z|$ 参照する $(self, z)\}$
    ここで $self$ は，辞書で定義されたオブジェクトを指す
  - 不変表明:
  - 戻り値の型: $\mathbf{Z}$
    .....

『文書』
- 所有者: 品質監査者
- 指示:
  x は，版管理されている開発工程の成果物 $\approx$ 文書 (x)
  .....
- ＜責任＞
  参照成果物を得る
  - 事前条件:
    $!\exists y \cdot$ 生成する $(y, self)$
  - 事後条件:
    戻り値の型を $\mathbf{Z}$ としたとき，
    $\mathbf{Z} = \{z|$ 入力する $(y, z) \wedge$ 生成する $(y, self)\}$
  - 不変表明:
  - 戻り値の型: $\mathbf{Z}$

構成管理者の責任は，プロジェクトの記録や文書，ソフトウェア部品などの成果物を適切に配布し，保管し，廃棄することである．さらに，成果物がどのような成果物を参照して生成されたのかといった文書間の関係も管理する．一方，品質監査者は，構成管理が適切に行われていることを監査すると共に，プロジェクトの開発プロセスがその時点の最新の成果物を参照して実施されていたかを監査しなければならない．

構成管理者の文書オブジェクトに定義された**参照文書を得る**と，品質監査者の文書オブジェクトに定義された**参照成果物を得る**は，意味的に等しいので，次の関係が成り立つといってよい．

参照する $(x, z) =!\exists y \cdot ($入力する $(y, z) \wedge$ 生成する $(y, x))$

これは，品質監査者のモデルを用いれば構成管理者のモデルでも文書オブジェクトから目的の参照文書まで到達可能であることを表わしている．したがって，構成管理者の文書間の参照関連は，品質監査者のモデルで置換できる[20]．

4. 議　　論

4.1 手法の評価

組織化の過程は，大きく次の 3 つの過程に分かれている．すなわち，オブジェクトが指示する現実世界の事物や事象，およびサービスの契約を暫定版オブジェクト辞書へ定義し，利用者視点モデルを構築する過程，定義された暫定版オブジェクト辞書と利用者視点モデルをもとに，衝突を発見し解消する過程，衝突が解消された暫定版オブジェクト辞書を統合して正式版オブジェクト辞書を作成し，多重視点モデルを構築する過程である．組織化過程を段階的に進めることによって，オブジェクト抽出と衝突の発見／解消の作業を分離でき，それぞれの作業を簡素化することができた．また，利用者視点モデルは，多重視点モデルに比べて規模が小さく，レビューに参加する利用者の視点から観察される範囲だけがモデル化されているため，レビューによる検証に適している．以上から，多重視点の問題領域では，一度に全体のモデルを構築するよりも，利用者の視点ごとにモデルを構築した方が適切なモデルを効率的に定義でき，検証もしやすくなるといえる．

4.2 分析における情報隠蔽

組織化過程では，モデルと，モデルが指示している

現実世界の事物とを対応づけるために，OBA[16]で扱われるオブジェクトのモデル化カードに指示の項目を追加し，クラスの情報隠蔽を守るために属性の項目を削除した．多重視点モデルでは，名前の衝突がクラス名，属性名，サービス名，関連名で起こる可能性がある．だから，衝突の検討対象を絞り込まないと，多くの繁雑な検査を行う必要が生ずる[13]．分析段階でオブジェクトの情報隠蔽を考慮するために，オブジェクトを，他のオブジェクトから参照される外部構造とオブジェクトが保持する内部構造へ分割した．オブジェクトを外部構造と内部構造へ分けることによって，最初の衝突の検討をオブジェクトの外部構造についてだけ進め，衝突を解消することができる．内部構造については，オブジェクトの責任を果たすための構造が仮の構造として残されているので，仮関連に関する衝突の発見と解消の作業は，オブジェクトのカプセル内の作業へ転換することが可能となる．オブジェクトの属性は，カプセル内の構造を定義する過程で必要となるデータであるから，属性の項目は正式版オブジェクト辞書へ定義するだけでよい．

4.3 モデルの取り扱い

組織化過程が一巡した時点で，多重視点モデルが参照している正式版オブジェクト辞書と利用者視点モデルの暫定版オブジェクト辞書の対応が保証される．だから，レビューの対象は，どのような人がレビューに参加するかによって変えてもよい．

多重視点モデル構築後のモデルの更新も同様に，利用者視点モデルと多重視点モデルのどちらに対して行っても問題はない．ただし，変更されたモデルの情報を他のモデルへ反映させる再組織化は必要である．最初の組織化で，正式版オブジェクト辞書が利用者視点モデルと多重視点モデルの仲立ちをしたように，再組織化では，どちらのモデルが変更されたとしても，正式版オブジェクト辞書がモデル間の仲立ちとなる．

5. 関 連 研 究

多重視点を考慮したモデル化の研究として，Harrison らの研究がある．Harrison らは，主観的な視点から観察される形態としてサブジェクトを提案した．サブジェクトが本稿で議論した視点に該当する．Harrison らの機構に従うと予め多重視点モデルを構築しておく必要がない．また，共有オブジェクトを参照するモデルが定義された段階で，既存のオブジェクトの構造と新たに定義されたオブジェクトの間の対応関係を管理するオブジェクト識別子と Composite Rule を定義する．Composite Rule を定義することで，共有オブジェクトの状態に不整合が起きないようにでき，各アプリケーションモデルのオブジェクトは継承構造が異なっていても問題とならなくなる[4]．Composite Rule がモデルとは分離した広域的な領域に定義されているため，個々のモデルの変更は柔軟に行えるが，モデルの変更による整合性の保守に問題があると思われる．我々の多重視点モデルは，視点の拡大や追加について，Harrison らの方法ほど柔軟に対応できないが，2 種類のモデルとモデル間の対応をつけるオブジェクト辞書が存在するので，利用者視点モデル間の整合性を保守する機構を提供した．

Minsky らによる Law-Governed Architecture(LGA) は，各視点ごとに代理オブジェクトが定義され，個々の代理オブジェクトが基本オブジェクトへメッセージの伝達を行う機構を持つ．この機構では，複数の代理オブジェクトが基本オブジェクトを共有するので，基本オブジェクトが共通サービスを代理オブジェクトへ提供し，代理オブジェクトはその他の付帯的なサービスをクライアントへ提供するといった役割分担がある．しかし，クライアントからは基本オブジェクト，代理オブジェクトの区別を意識することなくメッセージを送ることができる[11]．我々が提案した組織化過程は，利用者視点モデルと多重視点モデルを提供できるので，LGA を適用する際にも有効である．

Nuseibeh らの多重視点に関する研究では，多重視点で定義されるオブジェクト間の矛盾を発見する方法として，オブジェクト間のメッセージの送受信に着目する方法が提案されている[14),15]．サービスの入力引き数となるオブジェクトは必ず生成されていなければならないといったモデルの過不足を統合時に発見する手段として有効である．組織化過程には，サービスに限らず，モデルの妥当性を評価する過程が必要である．これは今後の課題としたい．

プログラミング言語を多重視点へ適用させる研究は，Shilling らによる CV++の開発がある．これは，クラスに多相的なインタフェースを定義できるように C++を拡張した言語である．クラスは，同じ名前のインタフェースを視点ごとに持ち，メッセージを受信したとき，メッセージの送り手に依存した実装を実行できる．この機構は，インタフェース層，手続き対応層，および手続き層の 3 層によって構成されている[18]．多重視点を支援するプログラミング言語を使用する場合も，利用者視点モデルや多重視点モデルを構築する必要がある．CV++のような多相的なインタフェースをクラスに定義できるのであれば，サービス名の衝突のうち，同音異義語については名前を変更する必要がな

くなる．

組織化過程では，設計工程以降のモデルの取扱については触れていない．これは，関連研究で説明したように，多重視点の問題領域を実現する方法はいくつもあり，どのモデルを実装するかは実装環境に依存して決まるからである．

6. ま と め

本稿では，多重視点の問題領域を分析するとき，システムの利用者を明らかにし，利用者ごとのモデルを構築した後に各モデルを統合する組織化過程を提案した．段階的に組織化を進めることによって，オブジェクト抽出と衝突の発見／解消の作業を分離できるようになった．また，利用者視点モデルは，多重視点モデルに比べてレビューしやすい規模となるという利点がある．組織化過程で使用するオブジェクト辞書は，そこに登録される各項目が衝突の発見に有効であることを示した．

参 考 文 献

1) Batini, C. and Lenzerini, M.: A Methodology for Data Schema Integration in the Entity Relationship Model, *IEEE Transactions on Software Engineering*, Vol.SE-10, No.6, pp.650–664 (1994).
2) Booch, G.: *Object-Oriented Analysis and Design with Applications*, Benjamin Cummings, second edition (1994) (山城明宏監訳, Booch 法：オブジェクト指向分析と設計, 星雲社 (1995)).
3) Coad, P. and Yourdon, E.: *Object-Oriented Analysis*, Prentice Hall, second edition (1991) (羽生田栄一監訳, オブジェクト指向分析 (OOA), トッパン (1993)).
4) Harrison, W. and Ossher, H.: Subject-Oriented Programming (A Critique of Pure Objects), *Proceedings of OOPSLA*, ACM, pp. 411–428 (1993).
5) 本位田真一, 山城明宏: オブジェクト指向システム開発, 日経 BP 出版センター (1993).
6) ISO-9000-3, *Quality Management and Quality Assurance Standards*, ISO (1991).
7) Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G.: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley (1992) (西岡利博他訳, オブジェクト指向ソフトウェア工学 OOSE, トッパン (1995)).
8) Liskov, B. and Wing, J. M.: Specifications and Their Use in Defining Subtypes, *Proceedings of OOPSLA*, ACM, pp. 16–28 (1993).
9) Shlaer, S. and Mellor, S.: *Object-Oriented Systems Analysis*, Prentice Hall (1988) (本位田真一他訳, オブジェクト指向システム分析, 近代科学社 (1990)).
10) Meyer, B.: *Object-Oriented Software Costruction*, Prentice Hall (1988) (二木厚吉監訳, オブジェクト指向入門, アスキー出版 (1990)).
11) Minsky, N. H. and Pal, P. P.: Providing Multiple Views for Objects by Means of Surrogates, Viewpoints 96: International Workshop on Multiple Perspectives in Software Development (SIGSOFT 96), ACM Press (1997) (http://www.cs.city.ac.uk/homes/gespan/vptoc.html).
12) Jackson, M.: *Software Requirements & Specifications*, ACM Press (1995).
13) Nakatani, T. and Tamai, T.: A Domain Method Integrating Multiple Views, *Proceedings of Changsha International CASE Symposium*, China, pp. 205–210 (1995).
14) Nuseibeh, B.: Towards a Framework for Managing Inconsistency Between Multiple Views, Viewpoints 96: International Workshop on Multiple Perspectives in Software Development (SIGSOFT 96), ACM Press (1997) (http://www.cs.city.ac.uk/homes/gespan/vptoc.html).
15) Nuseibeh, B., Kramer, J. and Finkelstein, A.: A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, *IEEE Transactions on Software Engineering*, Vol.20, No.10, pp. 760–773 (1994).
16) Rubin, K. and Goldberg, A., Object Behavior Analysis, *Communications of the ACM*, Vol.35, No.9, pp. 48–62 (1992).
17) Rumbaugh, J., et al.: *Object-Oriented Modeling and Design*, Prentice Hall (1991) (羽生田栄一監訳, オブジェクト指向方法論 OMT, トッパン (1992)).
18) Shilling, J. J. and Sweeney, P. S.: Three Steps to Views: Extending the Object-Oriented Paradigm, *Proceedings of OOPSLA*, ACM, pp. 353–361 (1989).
19) Wirfs-Brock, R. and Wilkerson, B.: Object-Oriented Design: A Responsibility-Driven Approach, *Proceedings of OOPSLA*, ACM, pp. 71–75 (1989).
20) Yao, S. B., Waddle, V. E. and Hausel, B. C.: View Modeling and Integration Using the Functional Data Model, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 6, pp. 544–553 (1982).