データベースシステム　97－1
（１９９４．　３．　１８）

インスタンスの抽象度の異種性を考慮したスキーマ統合

鈴木源吾　山室雅司　磯部成二

NTT　情報通信網研究所

データベース設計における複数のユーザビューの統合や，既存データベースを相互利用可能にするために既存DBのスキーマからグローバルスキーマの定義を行うためには，複数のスキーマを組み合わせ一つのスキーマを構成する必要がある．それはスキーマ統合と呼ばれ，これまで方法論を含む様々な検討がされている．スキーマ統合の方法論においては，複数のスキーマ間の異種性の分析手法と解消法が大きな課題である．既存の検討では，意味的な異種性として，クラスの抽象度の異種性に焦点が当てられている．これはgeneralizationによって解消できる．

本稿では，意味的異種性として，インスタンスの抽象度の差を考慮した方法論を提案する．これは，例えば通信網管理データベースのような，階層的・統計的データを含むデータベースの統合において有効である．インスタンスの抽象度の異種性として，現実世界の実体のグルーピングと統計データのグルーピングの２つの場合を考慮している．これらの意味的異種性を解消するために，既存のスキーマ統合の方法論を拡張し，この異種性が記述可能なビューの記述言語を提案する．

Schema Integration Considering Heterogeneity of Abstraction of Instances

Gengo SUZUKI　Masashi YAMAMURO　Seiji ISOBE

NTT Network Information Systems Laboratories

We have to integrate several schemas to one unified schema when integrating user views in database design process or when defining global schema using schemas of existing databases so that those databases are interoperable. Such tasks are called "schema integration", and so far many researches including methodologies have been done about this topic. Methodologies for analyzing inter-schema heterogeneity and resolution techniques are two main problems of schema integration. So far heterogeneities of classification are focused as semantic heterogeneity. This type of heterogeneity can be resolved by generalization.

In this paper, we propose a schema integration methodology which considers heterogeneity of abstraction of instances. This method is valid when integrating databases such as telecommunication management databases, which store hierarchical and statistical data. We take into account two types of heterogeneity of abstraction of instances, grouping of real world entities and grouping of statistical data. We discuss the methodology of schema integration considering these semantic heterogeneity and propose a view definition language which has capabilities to describe this heterogeneity.

1．はじめに

スキーマ統合は，データベース設計においては，複数の独立に設計されたユーザビューの統合の目的で，連合データベース・マルチデータベースにおいては，既存のデータベースのスキーマから複数のデータベースを同時に見たいユーザが利用するスキーマ（グローバルスキーマ）を定義する目的で研究されてきた．スキーマ統合の研究課題としては，

・スキーマ間差異の整理と分類[4]

・手順・方法論

・支援ツール[5]

などがある．包括的なサーベイとしては，Batiniらによる[3]がある．

スキーマ統合の重要な課題として，スキーマの意味的な異種性の分析・解消がある．意味的異種性は，複数の設計者が同じ実世界の対象に対して，異なった抽象化（モデル化）を行うために生じる．これまでの研究では，おもにクラスの抽象化の差異（例：「生徒」クラスと「情報科学科生徒」クラス）に焦点があてられていた[3][6]．その解消の戦略はgeneralizationの概念の利用である．しかし，例えば，通信網管理データベースのような，階層的（例：ケーブルの中に複数の電話回線を含む）・統計的（例：東京・大阪間で１００callの通話があった）な性質を持つスキーマの統合においては，インスタンス（データ管理上の単位）の抽象度の差異が問題になる．例えば，一方のスキーマにおいては，電話回線を一本づつ単独にインスタンスとして管理しているが，もう一方のスキーマにおいては，両端のエリアで電話回線をグルーピングしたものを一つのインスタンスとして管理している場合がそれに当てはまる．

そこで，本稿では，インスタンスの抽象度の異種性を考慮したスキーマ統合の方法論を提案する．グルーピングの異種性である点では共通であるが，意味が異なる２種類の異種性（実体のグルーピング・統計的グルーピング）のパターンを考慮する．そして，Dayalらによって提案された，連合DBのグローバルスキーマを定義するためのビュー定義言語[7]を，この異種性の解消を記述できる形に拡張する．

第２章においてスキーマ間の差異の分類と，インスタンスの抽象度の異種性について説明し，議論の前提として利用するデータモデル（IFOモデル）に触れる．第３章でスキーマ統合の方法論について議論する．第４章では，ビュー定義言語の拡張についてのべ，第５章では通信網管理データベースを対象とした，スキーマ統合の適用例を述べる．第６章で要約する．

2．意味的異種性

2．1　スキーマ間差異（inter-schema discrepancy）

異なった利用要求に基づく複数のユーザービューを，それぞれ別の設計者によって設計した場合や，（不幸にも）複数のデータベースが独立にスキーマ設計された場合，同じ現実世界に対して，異なったモデル化が行われることがある．そこで生じる矛盾はスキーマ間の差異・異種性（inter-schema discrepancy, heterogeneity, conflict）と呼ばれる．比較的最近の，Spaccapietraらによる整理[4]によるとスキーマ間の差異は以下のように分類されている．

（１）意味的矛盾（semantic conflict）

UoD（Universe of Discorse：現実世界）の実体を表現する２つの要素が，等式ではなく集合比較演算によって関係づけられている場合を，意味的矛盾と呼んでいる．例えば，”生徒”クラスと，”情報処理学科の生徒”クラスの関係がこれにあたる．

（２）記述的矛盾（descriptive conflict）

関連するUoDの実体を，異なる性質の集まりで表現している場合を記述的矛盾と呼んでいる．命名矛盾や単位の矛盾などを含む．

（３）データモデル矛盾（data model conflict）

２つのスキーマが異なるデータモデル（例：関係モデルとオブジェクト指向モデル）で記述されている場合をデータモデル矛盾と呼ぶ．

（４）構造的矛盾（structual conflict）

２つの関連したUoDが異なったデータ構造で表現されている場合を構造的矛盾と呼ぶ．例えば，データモデルとしてERモデルを用いた場合，ある概念を属性で表現しているか実体型として表現しているかといった矛盾がこの場合にあたる．

この整理は直交した概念に分類されており，全体的な整理として有効であるが，（１）の意味的矛盾に関しては不十分な点があることを指摘したい．それを次節で説明する．本稿では（２）～（４）の矛盾については新たな議論はしない．

2．2　インスタンスの抽象度の異種性

２．１の整理を含め，既存の検討においては，意味的矛盾は，インスタンスの集合：クラス（ERモデルではentity-typeに相当）の抽象化の差異として考えられている[3][6]．それは，domainの包含関係を分析し，generalizationの概念を用いることによって解消することができる．しかし，以下の特徴を持つデータベース（例えば，通信網管理データベース）をスキーマ統合の対象とする場合，それだけでは不十分である．

・階層性

データベースが管理している実世界が階層性を持っている．例えば，ケーブルの中に複数の電話回線を含む，などがある．

・統計性

データベースの持つ情報が統計的情報である．例えば，東京・大阪間で一定時間に１００callの通話がある，などの情報である．

このような性質を持つスキーマを統合する場合，インスタンスの抽象化レベル（現実世界の何を一つのイン

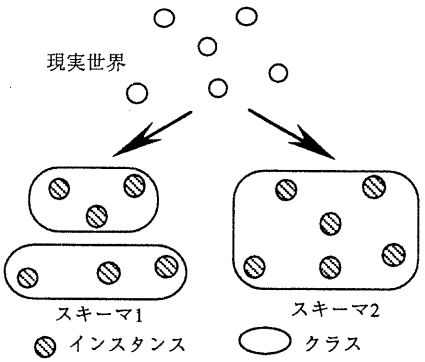


図1　クラス化の差異

スタンスと見るか，という単位）の差異がより頻繁に発生する．それは，複数の概念が互いに集合とその要素の関係になっている場合が多いためである．よってこの異種性もスキーマ統合の方法論の中で整理されていることが望ましい．ここでは，２種類のインスタンス化の異種性について述べる．これらはグルーピングによって解消できる点で同様のセマンティクスを持っているが，応用上や，異種性解消後のクラスの持つ属性が異なるために，区別を行っている．

またこれらは，それぞれ意味データモデル[8]や統計データモデル[9]の分野で新しい型を生成するための概念として捕えられていたが，スキーマ統合の方法論の中で考慮された例は，筆者の調査した範囲ではない．

（１）実体の集合としてのグルーピング

一方のスキーマのインスタンスが現実世界の個々の実体を表わし，もう一方のスキーマのインスタンスがその実体の集合に対応する場合である．例えば，一方のスキーマにおいては，電話回線を一本づつ単独に管理しているが，もう一方のスキーマにおいては電話回線をその両端のエリアでグルーピングして管理している場合がそれに当てはまる（図３）．この場合，グルーピングされたクラスの性質は，グループ内のインスタンスの数や，性質の平均や最大値などを含む．

（２）統計的データの並列化と直列化

一方のスキーマの統計データの１レコードが，もう一方のスキーマの複数の統計データをまとめたレコードとなっている場合．例としては，時系列情報があげられる（図４）．一方のスキーマでは，１時間ごとにレコードが構成されているが，もう一方では１日をレコードの単位とし，１時間ごとのデータは１つのレコード内に２４個の属性として保持する．前者から後者への変換を並列化と呼び，その逆を直列化と呼ぶ．

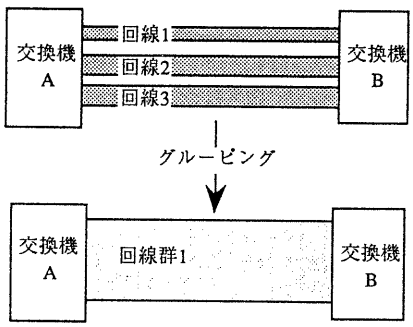


図3　実体の集合としてのグルーピング

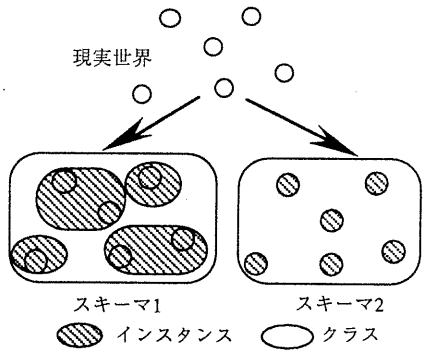


図2　インスタンス化の差異

２．３　データモデル（IFO）

スキーマの統合の記述に利用するデータモデルとして，グルーピングを陽に記述できるモデルを使う必要がある．本稿ではIFOモデル[8]を利用する．その理由は，

・グルーピングを構成子として持つこと

・意味データモデルの概念と用語の整理も兼ねた比較的新しいモデルであること．

・ビューの記述言語としてDayal, Hwang[7]の検討を利用したいこと．

[7]では関数型モデルを用いた言語を提案している．IFOが関数型の自然な拡張であるため，[7]の言語もグルーピングの概念を追加するだけで自然に拡張できる．

IFOモデルの概要を説明する．

２．３．１　基本要素

（１）object type

データベースアプリケーションに生ずるobjectの構造をモデル化している．これまでやや漠然と使っていたクラスに対応する．printable（□で表す．predefined typeの

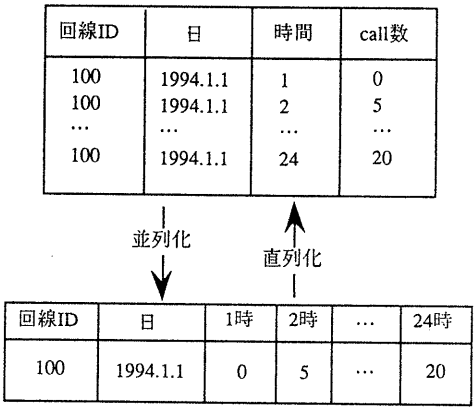


図4　統計的グルーピング（並列化と直列化）

objectsに相当），abstract（◇で表す．内部構造の見えないobjectsに相当），free（○で表す．IS-A階層により生成されるタイプ）の３種類がある．

（２）fragment

複数のobject type間の関数関係を表現する．関数型データモデルではfunctionに相当する．ERモデルでは属性または関連に相当する．object type間の方向付（定義域→値域）の矢印で表現する．Tを定義域とするfragment全体をfrag(T)で，fの値域をrange(f)で表す．本稿では，定義域中のobjectを一意に決めるfragmentの集合をキーと呼ぶことにする．

２．３．２　型の構成子（type constructor）

（１）グルーピング

あるobject typeのobjectの集合をobjectとするobject typeを定義する．記号＊で表す．

（２）generalization，specialization

generalizationは，複数のobject typeの非交和和集合のobject typeを生成するのに用い，type間の塗りつぶされた矢印（→）で表す．specializationは，object typeのある役割を持った部分集合のobject typeを定義するのに用い，幅広い矢印（⇒）で表す．

（３）直積

複数のタイプの直積の形をしたobjectのtypeを定義できる．記号×で表す．

３．スキーマ統合方法論

スキーマ統合は大きく４つの手順（統合準備・スキーマ比較・スキーマ調整・マージと再構成）で構成されることが明らかになっている[3]．本稿では手順自体の拡張はせず，インスタンス抽象化の異種性を考慮した場合の各手順における必要な分析を明らかにする．スキーマ統合はDB概念設計，連合データベースのグローバルスキーマ設計両方に利用できるが，ここでは主に後者を想定して議論する．分析の考え方自身は，概念設計の場合もほぼ同様であるが，若干の変更が必要な場合もある．統合対象とするDBをローカルDBと呼び，そのスキーマをローカルスキーマと呼ぶ．

全体の戦略としては，インスタンスレベルの抽象度の違いをスキーマ調整時に解消することによって，クラスレベルの差異が比較可能な形にする（そのような状態をcompatibleであると呼ぶ．）．マージ以降は既存の方法論を使用することができる．

３．１　統合準備（preintegration）

全体的方針（統合する対象・順番の決定）を決定するフェーズであるので，既存の方法論をそのまま利用することができる．

３．２　スキーマ比較（schema comparison）

スキーマ間の関係を明らかにするフェーズである．２つのフェーズに分解する．

（１）発見

スキーマ間で，集合（set-role-type，それを含むスキーマをset-role スキーマと呼ぶ．）とその要素（element-role-type，それを含むスキーマをelement-role スキーマと呼ぶ．）にあたる概念を発見する．発見の支援のために，概念の名前の関係から候補をあげることもできる（「X集合」←→「X」のように，他方の名前を含む関係）．しかし，そうでない場合は対象分野の専門家が分析することにより発見する．

（２）統合可能分析

発見されたset-role-typeとelement-role-typeが，互いにcompatibleな形に変換できる関係であるか，何らかのスキーマの変形が必要であるか，などの相互関係を分析する．統合されたスキーマを仮想的に見せるためには，setとelementの対応付けに必要な情報がローカルスキーマに存在することが必要である．もし，存在しない場合，新たなデータ入力が必要となる場合もある．このフェーズでは，以下のどの場合に該当するかを判断する（図５）．どの場合に該当するかによって，次のスキーマ調整フェーズでスキーマの変形が必要になる．set-role-typeをT1，element-role-typeをT2とする．

[CASE１：compatible化可能]

T1とT2はそれぞれ現実世界の要素と集合を表わしている．set-role-typeとelement-role-typeの集合と要素の対応の意味（人間が現実世界で認識する意味）を表わす写像をFで表わすとする．

$\exists f1 \subset frag(T1), f2 \subset frag(T2)$ s.t.
f1 is key of T1 , range(f1) = range(f2) and
f1(F(e)) = f2(e) for all e $\in$T2

の場合をT1とT2がcompatible化可能であると呼ぶ．これはつまりset-role-typeのキー(f1)と同じtypeの値をとるelement-role-typeのfragment(f2)があり，集合要素の意味を表現している場合である（図５）．この時，f1とf2を同等であると呼ぶ．

この時，T2/f2（T2をf2の値が等しいという同値関係で類別した同値類の集合）とT1が統合可能（generalization-compatible）である．T1のf1の値とT2/f2の代表元の値を比較することによって，T2/f2とT1が統合される．

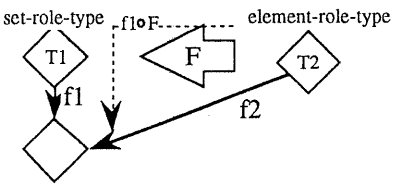
[CASE２：set化可能]

$\exists f1 \subset frag(T1), f2 \subset frag(T2)$,
$\exists g : range(f2) \rightarrow range(f1)$ 全単射（人間が認識） s.t.
f1 is key of T1 , range(f1) $\neq$ range(f2) and
f1(F(e)) = g(f2(e)) for all e $\in$T2

の場合をset化可能，compatible化不可能であると呼ぶ．これは，element-role-typeをf2によって類別するとset-role- typeのobjectと一対一に対応するが，f1とf2のとる値は別のtypeの場合である（図５）．T2/f2とT1のobjectは，抽象度が同じ現実世界のものを表すが，同一であることを判定するfragmentが存在しない状態である．

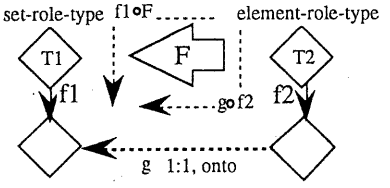
[CASE3：set化不可能]

CASE1・CASE2のようなf1, f2が存在しない場合をset化不可能（ゆえにcompatible化不可能）であると呼ぶ．そのままでは，T1と１対１に対応するobjectを，T2から生



f1(F(e)) = f2(e) for all e ∈T2

（１）compatible化可能



f1(F(e)) = g(f2(e)) for all e ∈T2

（２）set化可能

図5　統合可能分析

成することができない状態である．

3．3　スキーマ調整

(schema conforming)

3．2でなされた分析ごとに，スキーマの変形を行う．element-role スキーマに，同じ抽象度のtypeを追加を行うcompatible化と，set-role- typeに，必要なfragmentを定義するfragment定義の２つを行う．

3．3．1　compatible化

（１）compatible化可能な場合

element-role スキーマに，set-role- typeとcompatibleなtypeを追加する．

（２）set化可能compatible化不可能の場合

T1とT2/f2をgeneralization-compatibleにするためには，element-role スキーマまたはset-role スキーマのスキーマ変更（fragmentの追加）が必要である．変更の方法には，以下の３つが考えられる．

a)導出fragmentの定義

導出方法としては，既存の複数のfragmentの結合（RDBにおける他のテーブルとのjoinに相当）と算術演算などの操作の既存のfragmentへの適用の２つの方法がある．後者の例としてはfragment年月日から年のみを導出するなどの場合がある．

b)export範囲の拡大

セキュリティなどの理由によって，ローカルスキーマの一部のみを，グローバルユーザに公開する場合がある（公開されたスキーマをexport schemaなどと呼ぶ[1]）．この場合，その範囲を見直すことによって，compatible化可能にできないか検討する．

c)関係を表す情報の追加

T2/f2とT1の１対１関係を表す情報を入力する．それは，連合DBMSが管理する場合（Multibaseでいう補助スキーマ[10]）の場合と，ローカルDBのスキーマ変更による場合がありえる．

もし，必要なfragmentの追加が可能となったなら，compatibleなtypeを生成する．もし，追加が不可能であった場合は，統合は不可能であると判定する．

（３）set化不可能な場合

（２）と同じ方法で，set化可能・compatible化可能のために必要なfragmentを追加する．しかし，この場合は，ローカルスキーマの変更なしですませることは難しい．追加できれば，（２）と同様にcompatibleなtypeを生成するか，統合不可能と判断をする．

3．3．2　fragment定義

実体の集合としてのグルーピングの場合，要素の個数や値の平均といったfragmentを新たに定義できる．統計的な並列化の場合は，element-role-typeでの値がset-role-typeのfragmentになる．例えば，２．２（２）の例の場合，element-role-typeで値であった「１時」「２時」が，「１時」「２時」　というfragmentになる．

3．4　マージと再構成

(merge and restructuring)

これまでで，objectレベルでの抽象度がそろったために，このフェーズでは，既存の方法論の変更は必要ない．object-typeのdomainの包含関係の分析結果に基づき，スキーマを重ね合せてグローバルスキーマを作る．

4．ビュー定義言語

ここで，与えられたスキーマからビューを定義するための，ビュー定義言語について述べる．この言語で記述されたビュー定義によって，

・ユーザビューの問い合わせをグローバルスキーマへの問い合わせに変換する処理に利用（ビュー）

・グローバルスキーマへの問い合わせをローカルスキーマへの問い合わせに変換する処理に利用（連合データベースの問い合わせ処理）

・スキーマ相互間の意味の対応の管理（設計）

の３点を行うことが目的である．本稿では全く新たな言語を提案するのではなく，関数型データモデルに基づくDayal, Hwang[7]のビュー定義言語をグルーピングを記述できる形に拡張する．５章で適用例を説明するために，Dayal, Hwang[7]とほぼと同じ仕様に関しても説明する（（１）～（３）．ただし，詳しい仕様は省略する．）．（４），（５）が本稿独自の拡張である．

（１）inclusion

include <object variable> **as** <object type>
　(<list of fragment>)
where <qualification>

与えられたスキーマの<object type>から，ビュー定義に用いる範囲を，<qualification>の範囲のobjectと，<list of fragment>のfragmentの組に限定する．

（２）仮想object typeと仮想fragmentの定義

define objecttype <object type>
(<list of assignments>)
where <quantification>

与えられた<object type>から，<qualification>によって限定されるobjectを持つ<object type>を定義する．<list of assignments>でその上のfragmentを定義する．assignmentの例を示す（従業員eのWork-inするdeptのNames）．

DeptNames := {n **in string where**
n = Name(d) **and** d **isin** Works-in(e)}

define fragment <fragment name>
for <object variable> **in** <object type>
(<assignment>)
where <qualification>

<object type>上のfragmentを<assignment>により，とる値のtypeを指定し，<qualification>により値を指定する．

（３）supertypeとその上のfragmentの定義

define supertype <object type-0> **by**
<object type-1> **isa_o** <object type-0>
．．．
<object type-k> **isa_o** <object type-0>
id: <list of single-valued fragment names>
for <object variable> **in** <object type-0>
<list of superfragment declarations>
<list of conditional assignments>
end

複数のobject typeのobjectを含むような，ビューのobject type（＝supertype）を定義する（ビューのobject typeと元のobject typeはIFOでいうspecializationの関係である．**isa_o**の**o**はobjectの**o**である．）．

supertype上のfragmentの定義としては，superfragmentと，条件分けによるfragmentの２種類が定義できる（上記定義中のsuperfragment declarationとconditional assignmentsにそれぞれ対応）．superfragmentとは，値として与えられたfragmentの値の和をとるfragmentである（**isa_f**で表す）．条件分けによるfragmentとは，ビューのobjectが与えられたobject typeのどの部分に含まれるかによって値が決まるfragmentである．後者の構文は以下の通りである．

<fragment name> := **case**
<condition-1> => <term-1> **where** <qualification>
．．．
<condition-N> => <term-N> **where** <qualification>
endcase

（４）グルービングタイプとfragmentの定義

本稿独自の拡張である．与えられたスキーマのobject typeからグルービングによって，新しいobject type（グルービングタイプと呼ぶ）を生成する．

define groupingtype <object type name> **by**
<object type name-1>
id:<list of single-valued fragment names>
for <object varible> **in** <object type>
<list of assignment>

<object type name-1>で指定されたobject typeのobjectの集合をobjectとする<object type name>という名前を持ったobject typeを生成する．２．２で述べた実体のグルービングの場合と，統計的並列化の場合は同じsyntaxで記述される．supertypeの場合はid句を，objectの同一視に使うidを指定するのに用いたが，グルービングタイプの場合は，id句において，グルービングタイプのobjectを一意に特定できるfragment（の組）を指定する．つまり，id句のfragmentの値（の組）を単位にグルービングを行う．これは，３．２のf2に対応する．

for句中で，グルービングタイプのfragmentを定義する．assignmentには，グルービングに伴う各種のbuilt-inのfragmentを利用できる．以下に説明する．

統計的並列化の場合を記述するために，グルービングされたobjectとその要素のobjectの関連を表す関数として，**member**を追加する．a = **member**(b)とは，object a が object b の要素になっていることを表す．

グループに対して算術関数を適用することによるfragmentの定義が可能である．実体のグルービングの場合に適用される．算術関数としては，**count**，**sum**，**avg**（平均）などがあり，通常用いられる意味を持つ．例えば，a = **count**(b)は，bの要素の個数がaとなるという意味である．

assignment中で，値が一つに決まらない場合，そのfragmentは集合値関数となる．

（５）直列化タイプの定義

直列化タイプの定義は仮想object typeの定義を拡張することによって，サポートできる．変更点は次の２点である．

・conditional assignment

super typeの定義に用いたconditional assignmentと同じ構文でfragmentの定義が可能．

・***take_any_value***

元の属性名中に含まれた分類のための値（例えば，「１時のcall数」というfragment名に含まれる1という値）は，全てfragment（例えば「時間」）の値に変換される．直列化タイプのobjectは，その新しいfragmentに関して必ず全ての値をとる（例えば，1時から24時）ことになる．この場合を「***take_any_value*** **in** <object type>」で表す．

例を示す．

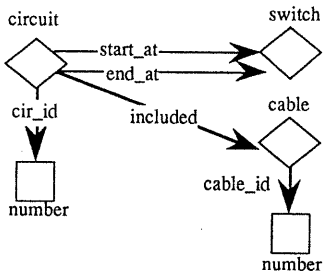
```
range of dc is day_calls
define object type hour_calls
  for hc in hour_calls
    cir_group := cir_group(dc)
    time      := take_any_value in time_type
    calls     := case
      time(hc) = 1 => calls_at_1(dc)
      . . .
      time(hc) = 24 => calls_at_24(dc)
      endcase
  end
```

5．適用例

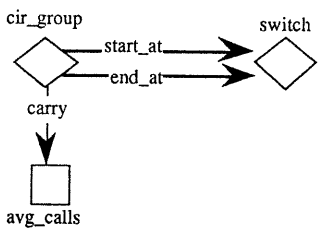
通信網管理データベースを例にとり説明する．

5．1　実体のグルーピングの例

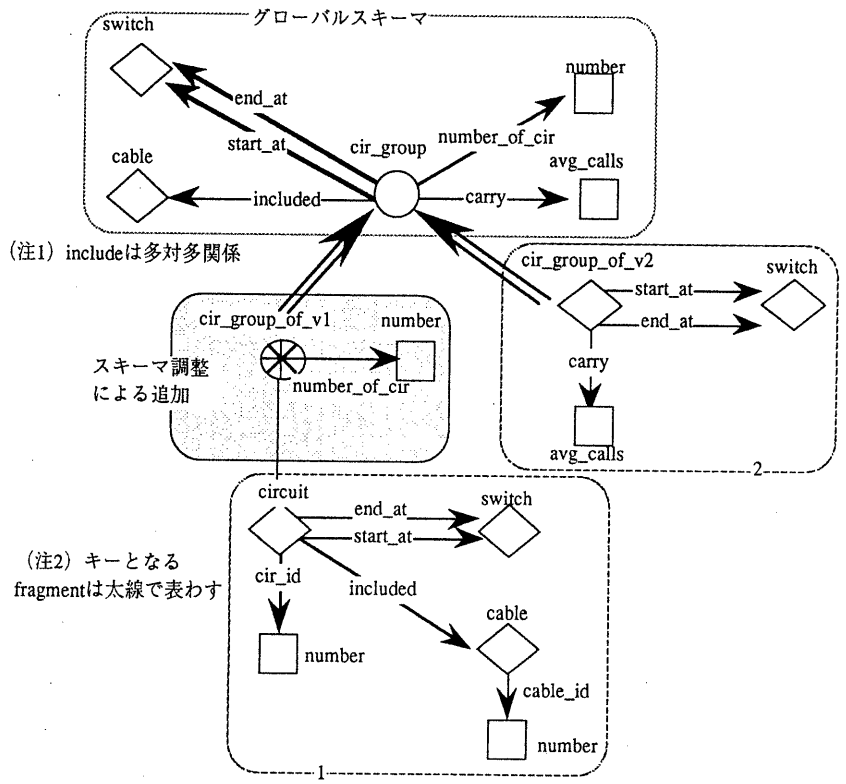
通信網構成管理スキーマと，通信量管理スキーマを統合する（図6）．通信網構成管理スキーマでは，circuit（電話回線）・switch（交換機）・cable（通信ケーブル）といったobject typeを持っている．そしてcircuitとswitchには，始点となるという関連（start_at）と，終点となるという関連（end_at）が定義されている．circuit



（1）通信網構成管理スキーマ



（2）通信量管理スキーマ



（3）統合されたスキーマ

図6　例1：実体の集合としてのグルーピング

```
range of e1 is VIEW1.circuit, range of e2 is VIEW1.switch, range of e3 is VIEW1.cable
range of e4 is VIEW2.cir_group, range of e5 is VIEW1.switch, range of e6 is VIEW1.avg_calls

include e1 as circuit (cir_id, included, start_at, end_at)
include e2 as switch, include e3 as cable(cable_id)
include e4 as cir_group_of_v2(start_at, end_at, switch, carry)
include e5 as switch, include e6 as avg_calls

/* scheme conformation */
  /* group化したtypeの追加 */
define groupingtype cir_group_of_v1 by
        circuit
        id : end_at, start_at
  for cg in cir_group_of_v1
    (number_of_cir := count(cg))
  end
                                        (注) udはundefinedを表わす.
                                             データベース中に値がない.
/* merge */
define supertype cir_group by
        cir_group_of_v1 isa_o cir_group, cir_group_of_v2 isa_o cir_group
        id : start_at, end_at
  for cg in cir_group
    count : = case
      cg isin cir_group_of_v1 => count(cg)
      cg isin cir_group_of_v2  - cir_group_of_v1 => ud
      endcase
    carry : = case
      cg isin cir_group_of_v2 => carry(cg)
      cg isin cir_group_of_v1  - cir_group_of_v2 => ud
      endcase
    included : = case
      cg isin cir_group_of_v1 => included _v1(cg)
      cg isin cir_group_of_v2  - cir_group_of_v1 => ud
      endcase
  end
```

図7　例1のビュー定義

はcableに含まれるという関連（included）を持っている．通信量管理スキーマにおいては，回線グループ（cir_group）と，その回線グループに発生した呼（avg_calls）を管理しているとする．start_atとend_atはcir_groupのキーであるとする．

（1）スキーマ比較

回線グループは，回線と互いに集合と要素の関係にあるので，circuitとcir_groupが分析の候補となる（発見）．

circuitがcir_groupのキーであるstart_atとend_atと同等のfragmentを持つので，circuitはcompatible化可能である（分析）．

（2）スキーマ調整

circuitから，cir_groupにcompatibleなtypeであるcir_group_of_v1を生成する．

（3）マージ

cir_group_of_v1とcir_group_of_v2はキーであるstart_atとend_atの値の組を比較することにより，統合することができる．

5．2　統計的グルーピング（並列化）の例

日別load管理スキーマと時間別call管理スキーマを統合する．日別load管理スキーマにおいては，各回線グループ（cir_group）に対して，日別（注．一つのobjectが一日を表すという意味で日別と呼んでいる）の通話量が1時から24時まで，24個のfragment（load_at_1等）として表現されている（day_calls）．時間別call管理スキーマにおいては，各回線グループに対して，日と時間別にコール数（call）が定義されている（hour_calls）．日と時間は一つのobject type（date_time）で表現されている．

（1）スキーマ比較

day_loadとhour_callsを比較すればよいが，hour_callsが直列的，day_loadsが並列的スキーマである．ここでは，並列的スキーマに変形することに決定する（これは，アプリケーションからの要求条件などで決まる）．fragmentを分析すると，day_loadのkeyであるdayは，hour_callにはない．しかし，day_timeの一部である日と1対1対応がとれることが容易にわかるので，そのままではset化可能ですらないが変形によりcompatible化できることがわかる．

（2）スキーマ調整

compatibleにするために，fragment：dayを追加する．day_timeからdayを取り出すためには，部分文字列の取り出し操作などを用いればよい．追加されたこのdayをkeyにして，hour_callsを並列化する．

（3）マージと再構成

day_loadとday_callsをキーdayを比較することにより，統合することができる．

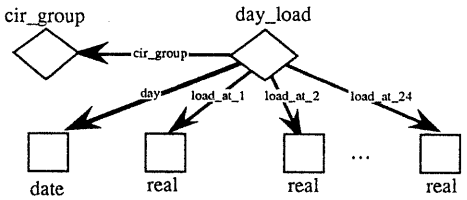
6．まとめ

インスタンスの抽象度の差を考慮したスキーマ統合の方法論を提案し，特に統合可能分析を明らかにした．また，その異種性を解消するグルーピングの記述が可能なビュー定義言語を提案した．グルーピングを陽に記述できるデータモデルとしてIFOを用いているが，定義言語は関数型モデルによる言語に対して，最小限度の拡張によって構成できた．通信網管理データベースを例にとって，方法論と定義言語の記述能力の有効性を示した．

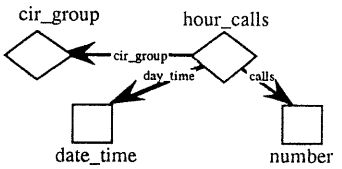
Dayal, Hwang[7]では，ビュー定義を用いて，連合データベースの問い合わせ処理を行っている．本稿では，問い合わせ処理アルゴリズムの拡張にはふれなかったが，本拡張はグルーピングの意味論を完全に記述しているので，論理的には同様の問い合わせ処理が可能なはずである．アルゴリズム設計・実システムでの確認は今後の課題である．

参考文献

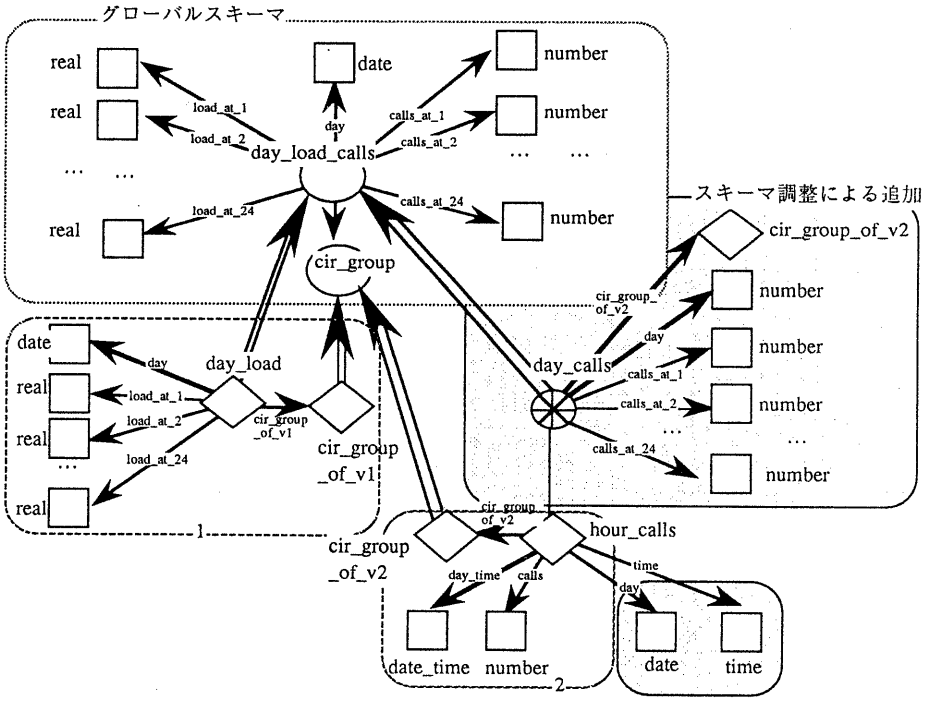
[1] Sheth, A.P. and Larson, J.A. "Federated Database Systems



（1）日別load管理スキーマ



（2）時間別call管理スキーマ



（3）統合化されたスキーマ（日単位）

図8　統計的グルーピング（並列化）

```
range of e1 is VIEW1.day_load, range of e2 is VIEW2.hour_calls

include e1 as day_load (load_at_1, load_at_2, . . . load_at_24)
include e2 as hour_calls (day_time, calls )

/* scheme conformation */
  /* fragmentの追加 */
define fragment day
      for e in hour_calls
        (day := 日の部分を取り出す(day_time(e)))
define fragment time
      for e in hour_calls
        (time := 時間の部分を取り出す(day_time(e)))

  /* group化したtypeの追加 */
define groupingtype day_calls by
        hour_calls
        id : cir_group_of_v2, day
  for dc in day_calls
    (calls_at_1 :
      = {c in calls where c = calls(hc) and hc isin hour_calls and hc = member(dc) and time(hc) = 1})
    … (24時間分繰り返し)
  end

/* merge */
define supertype cir_group by
        cir_group_of_v1 isa_o cir_group, cir_group_of_v2 isa_o cir_group
        id : start_at, end_at

define supertype day_load_calls by
        day_load isa_o day_load_calls, day_calls isa_o day_load_calls
        id : cir_group, day
  for dlc in day_load_calls
    load_at_1 : = case
      dlc isin day_load => load_at_1(dlc)
      dlc isin day_calls - day_load => ud
      endcase
    … (24時間分繰り返し)
    call_at_1 : = case
      dlc isin day_calls => call_at_1(dlc)
      dlc isin day_load - day_calls => ud
      endcase
    … (24時間分繰り返し)
  end
```

図9 例2のビュー定義

for Managing Distributed, Heterogeneous, and Autonomous Databases" *ACM Computing Surveys*, Vol.22, No.3, 1990

[2] Litwin, W. and Abdellatif, A. "Multidatabase interoperability" *IEEE Comput. Mag.*, Vol.19, No.12, 1986.

[3] Batini,C., Lenzerini, M., and Navathe, S.B. "A Comparative Analysis of Methodologies for Database Schema Integration" *ACM Computing Surveys,* 1986-4.

[4] Spaccapietra,S. and Parent,C. "Conflicts and Correspondance Assertions in Interoperable Databases", *SIGMOD Record*, Vol.20, No.4, 1993

[5] Sheth,A.P., Larson,J.A., Cornelio,A. and Navathe,B. "A Tool for Integrating Conceptual Schemas and User Views" In Proc. of *IEEE 4th International Conferencre on Data Engennering,* 1988.

[6] Larson,J.A., Navathe,S.B. and Ermasri,R. "Theory of Attribute Equivalence in Databases with Application to Schema Integration" *IEEE Transsaction on Software Engeneering,* 1989-4

[7] Dayal,U. and Hwang, H "View Definition and Generalization for Database Integration in a Multidatabase System" *IEEE Transsaction on Software Engeneering,* 1984-6.

[8] Abiteboul,S. and Hull,R. "IFO: A Formal Semantic Database Model" *ACM Transaction on Database Systems,* 1987.

[9] 佐藤 "統計データベースの設計と開発" オーム社, 1988.

[10] Landers, T. and Rosenberg, R. L. "An overview of Multibase" *in Distributed Databases*, North-Holloand, 1982.