情報学基礎 6－2
（1987. 9. 21）

ソフトウェアモデリングにおける役割付与の構造について

小林要・木村高久・織田充

富士通㈱国際情報社会科学研究所

ソフトウェアを，その応用目的側の構造のモデル（対象モデル）と，その計算機側の構造のモデル（実現モデル）とに分離し，さらに，それらの対応関係を構造化して扱うことをここでは，ソフトウェアモデリングと呼ぶ．本研究はそのための方法を提案している．

対象モデルおよび実現モデルは，それぞれの世界における，実体や関連を記述するものと考える．実体が各々の関連において果たす役割を，明示的に表現するために，モデル記述の基盤として用いる基盤言語の上に，さらに“役割パターン”を導入する．モデルは役割パターンの組み合わせによって表現される．

対象モデルと実現モデルとの対応関係は，対象モデル側に登場する役割を，実現モデル側の実体や関連に付与する，という“役割付与”の関係と考える．役割パターンに登場する役割部分を結合して得られた構造表現を用いる役割付与の構造を提案している．

On the Structure of Role Assignment in Software Modeling

Kaname Kobayashi, Takahisa Kimura, and Mitsuru Oda

International Institute for Advanced Study of Social Information Science,
FUJITSU LIMITED
MIYAMOTO 140, NUMAZU-SHI, SHIZUOKA, 410 —03 JAPAN

‘ Software Modeling ’ is the idea which aims at decomposing software into the object model which models application's world, and the implementation model that models implementation of software. This research proposes one of the methods for the software modeling.

Object model and the implementation model are considered to represent entities and relationships among each world. For the purpose of identifying roles of the entities played in each relationships, the ‘role patterns’ are introduced. Each model can be defined as a combination of role patterns upon each base language.

The structure of correspondences among the object model and the implementation model is proposed as the ‘role assignment structure’ which assigns roles in the object model to those in the implementation model.

1. はじめに

1.1 ソフトウェア開発における作業

ソフトウェアを開発したり保守を行う作業には，多種多様な情報や知識を必要とする．こうした知的作業の，効率や質を向上させるには，作業に必要な情報や知識が，的確に作業に反映するための工夫が必要である．

作業に必要な様々な情報や知識が，相互に複雑に関連している場合には，それらが相互に独立になるように整理されている場合よりも，作業の効率や質が落ちる危険性が高い．プログラムのモジュール化や構造化の技術は，プログラム自体を，互いに独立な単位に分割することによって，こうした危険性を回避しようとする技術であると見なすことができる．

1.2 対象世界と実現世界

モジュール化などによって分割されたプログラムの中にも，本来は独立である情報や知識が，まだ未分離なまま混在していると考えられる．

例えば，“３人でバスに乗り，バス代は一人当たり 200円であった．全部で何円払えばよいか”という問題に対して，「バス代金＝３＊ 200を計算し，印字装置を改行させ，“バス代金は”と印字し，バス代金の数字を印字し，“円です”と印字する」に相当するプログラムを考えたとしよう．この場合，プログラムには，少なくとも２つの独立な世界が同時に関与している．

第１に，何人かの人間がバスに乗って料金を支払う，という世界が関与している．プログラムが対象とする問題領域の世界（対象世界）がある．

第２に，整数の乗算を行ったり，印字装置を改行して，文字列や数字を印字させるなど，計算機の動作系列を実現する世界が関与している．計算機による動作を実現させるための世界（実現世界）がある．

バスに乗って料金を支払う世界と，計算機の印字動作や整数演算を実現する世界とでは，それぞれに登場する実体が異なる．また，バス代の料金体系の変化が，計算機の印字装置や整数演算装置の定義を変えるわけではないし，その逆もない．対象世界と実現世界とは相互に独立な世界である．

プログラム実体には，対象世界と実現世界の，本来独立した世界の情報が，未分離のまま関与していると考えられる．ソフトウェアモデリング〔１〕（図１）は，対象世界と実現世界のそれぞれのモデル，すなわち対象モデル，実現モデルを記述し，これらを独立に扱えるようにすることによって，ソフトウェアに関する情報を，さらに独立なものへ分解することを目的としている．

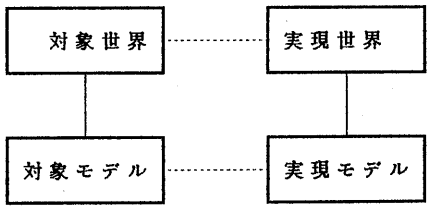

図１：ソフトウェアモデリング図式

1.3 ソフトウェアモデリングの効果

ソフトウェアを対象モデルと実現モデルとに分離することができれば，対象世界における様々な因果関係と，実現世界における諸因果関係とを分離独立して扱うことができる．すなわち，それぞれにおける推論が効果的に行えるための環境を整えることができる．

このことは，ソフトウェアの開発や保守における知的作業の単位を，関連する情報や知識の独立性に添って，さらに分離できることを意味する．また，作業に要する情報や知識がさらに局所化されることによって，作業内容や確かめるべき内容も局所化され，作業効率や作業の質の向上に寄与すると考えられる．

また，各世界において，成立する関係が，定義されてから変更されるまでの寿命を考慮すると，世界の分離は，プログラムの寿命の分離にも役立つ．一般に，対象世界における因果関係の寿命と，実現世界における因果関係の寿命は異なるが，プログラムの中に両者が混入している場合には，両者

のうちの寿命の短い方が，そのプログラムの寿命を決めてしまう．しかし，両者を分離しておくと，両者は，それぞれの寿命に従う自然な形になり，不必要な更新作業を減少させることができる．

更新作業量の減少や，長寿化の効果は特に，再利用や保守の場面で有効である．ソフトウェアコストの多くが保守コストであることや，生産性の向上の鍵が再利用にあることなどを考慮すれば，対象世界と実現世界との分離は，ソフトウェア開発にとって，極めて重要な課題である．

1.4 本研究の目標

ソフトウェアに関する対象世界と，実現世界を分離させるためには，どのようなことをしなければならないであろうか．対象モデルや実現モデルといっても，何をどのように表現し，どのように扱うことができればよいのかは必ずしも明確でない．

本研究では，対象モデルや実現モデルをどのように表現し，かつ，どのようにそれらを扱うべきかについての，一つの方法を提案する．

本研究では，対象世界と実現世界とのつながり方に注目する．対象世界と実現世界とは本来独立な世界であるとする場合，それらを結びつける根拠や手段が問題になる．我々は，実現世界における各々の個体や個体間の関係は，対象世界における何らかの個体や個体間の関係の役割を演じている，と考えた．すなわち，計算機の資源や動作の各々は，問題領域の対象世界における何らかの役を演じていると見なす．

しかし，何かが何かの役を演じる，という関係は必ずしも，対象モデルにおける個体と，実現モデルにおける個体との，すなおな対応では表せない．同一個体であっても，複数の役を演じる場合もあるし，同一の役を複数の個体が演じる場合もある．

次章では，役割が識別されるための工夫を施したモデル記述言語を導入し，3章では対象モデルから実現モデルへの，役割付与の構造について述べる．4章は本研究の考え方を実際に適用するための構想について述べる．

2. モデル記述言語

2.1 モデル記述言語への要請

対象モデル，および実現モデルを記述する目的は，以下の3つであると考える．

(a) 対象世界と実現世界とを分離すること，

(b) それぞれの世界の特徴を端的に反映すること，

(c) それを理解する者にとって，新たに獲得しなくてはならない情報や知識が最小化されている事（知識負担最小）．

(1) 基盤言語からの部分の抽出

(a) の要請に関して，対象世界と実現世界とを分離するには，対象モデルを記述するための言語と，実現モデルを記述するための言語とを，明確に区別する必要がある．少なくとも，それぞれの表現が，各々の世界における対象を確実に指示するものでなければならない．

一般に，対象世界の事象や個体を指示するために用いられる言語は，それぞれの対象世界毎にあると考えられる．例えば，銀行世界では，元金，利息，利率などの表現が指示する対象は明確である．それらの分野毎の表現言語は，何らかの基盤となる言語の上でのある特定の部分になっていると考えられる．

例えば，元金や利息などの表現は，日本語という基盤の上にある．さらに，銀行用語として，日本語の中のある特定の部分を構成していると考えられる．このように基盤となっている言語のことを，ここでは，基盤言語と呼ぶことにする．モデル記述言語は，基盤言語の部分を取り出したものとなればよい．

(2) 実体—関連関係の認識と役割の抽出

(b) の要請は，モデル記述が，単なる記述なら何でも良いというのでなく，何らかの構造を表現するものであることを要請している．すなわち，モデル記述から，容易に対象および対象間の関係が導き出されることが望ましい．

ここでは，Chen〔2〕の実体—関連モデルを参考に，実体に関するスキーマと，関連に関するスキーマとに対応するべき表現に注目する．対象モデルや実現モデルの記述に用いられる表現のクラスが，実体—関

連モデルのスキーマ表現に類似のものを考える．

実体―関連モデルでは，属性と役割との区別があり，スキーマ表現のレベルでは，スキーマ相互間の関係として，属性よりは役割を用いて表現することに着目する．

モデル記述言語としては，実体―関連モデルでいう役割の部分が抜き出せることが要請されることになる．

(3) モデル表現のクラスの最小化

(c)の要請である知識負担の最小化のためには，モデル記述に用いた表現のクラスがモデル理解者にとって必要十分なものを具体的に指定できる必要がある．

このためには，表現のクラスを，個々の表現のインスタンスとは区別して扱う必要がある．すなわち，個々のモデル表現とは別に，その表現に用いる表現のクラスを指定する手段が明確で，その指定された内容も，モデル理解に必要十分であればよい．

2.2 役割パターンの導入

前節で述べた3つのモデル記述言語への要請に対し，これらを満たすための工夫として，ここでは，以下で示されるような，役割パターンを導入する．

(1) 基盤言語におけるパターン

簡単のために，基盤言語Bはすでに与えられているとし，Bを基盤言語における表現の集合であるとする．Bの要素である各々の表現は，さらに簡単のために，文字列であるとしよう．

基盤言語Bの上のパターンpを考える．pはn個の引数の関数で，Bの上で定義されるものとする．今，簡単のために，pを次のような形式で表現しよう．

p (x1, x2, . . . , xn) =
" k1 〔x1〕 k2 〔x2〕 ... 〔xn〕 kn+1 " ..①

ここで，kiは基盤言語Bの要素であり，引数xjはパターン変数である．〔 〕はパターン変数と，基盤言語表現kiの並びとを区別するメタ記号である．引数個数nは非負整数で有限とする．

(2) 実体表現

引数個数が0のパターンは，基盤言語Bの要素になる．モデルを表現する場合には認識されるべき実体に相当する表現が必要になるが，ここでは，そのような表現を，引数個数が0のパターンとして準備する．これを特に実体表現と呼ぶ．

(3) 役割パターン

pはB上の関数であることから，パターン変数x1,x2,...,xnに，B上の表現がそれぞれ代入されると，全体としてもまたB上の表現になる．

モデルを表現する場合に，(1)で準備した実体表現によって指示される実体が，相互に関連する場合の表現が必要になる．このため，実体―関連モデルでいう関連関係のスキーマに対応するパターンを選択する．

パターンの選択基準は，モデル構築における関連の記述に必要十分なものであることであり，具体的な選定は，モデル作成者が，対象とする世界をどのように認識するかにかかってくる．

ただし，x1,x2,...,xnにB上の表現が代入された場合，pが関連関係を指示するためには，x1,x2,...,xnに代入された各々の表現で指示される対象には，pの指示する関連関係の中における，それぞれの役割が付与されねばならない．

すなわち，pが関連関係を表現するパターンであることから，パターン変数x1,x2,...,xnのpの中における各々の役割r1,r2,...,rnが対応して存在するものとする．

B上の任意のパターンから，役割が認識されたパターンを選定して，役割パターンとする．

(4) 関連関係における役割

上記の役割riは，モデル作成者が，対象とする実体が果たす役割として認識したものでなければならないが，それを表現する役割パターンpにおいては，pの中における各々の引数の場所を識別するものに相当する．

役割riは引数xiと類似ではあるが，ここでは，引数そのものとは区別する．引数のパターン変数はパターン内に閉じた自由変数であり，各々をパターンの外からは特定できないと考えるからである．

また役割riは他の役割パターンにおける引数の場所とも区別する．このため，役割riは役割パターンpと，そのpの中における引数の場所sとの組，

＜ｐ，ｓ＞　　..②

として定義される．Chen〔２〕の場合には役割はｓのみを識別するものであったが，ここでは，それを拡張し，ｐとｓの組としている．

その理由は，３章で述べるモデル間の役割付与構造に必要だからである．また，引数の場所を識別する，ということが，変数の識別と違う点は，それが基盤言語表現のレベルで表現できないメタレベルの情報であることも意味している．

このようなメタレベルの情報を準備する理由は，３章で行うモデル間の対応構造がメタレベルで行われるためである．

2.3 viewの導入

今，役割パターンｐの引数x1,x2,...,xnの各々に対して，実体表現e1,e2,...,enが代入されたとする．このときの結果は，

p (e1,e2,...,en) =

“k1e1k2e2...enkn+1 ”　　..③

となる．これもまた基盤言語Ｂの要素となることは，パターンｐがＢ上の関数であることから分かる．

ところで，役割ri＝＜p,i ＞であることから，パターン変数x1,x2,...,xnに対して実体表現e1,e2,...,enを代入する，という関係は，役割riと実体表現eiとの組の集合によっても表現できる．

今，役割riと実体表現eiの組を，

ri／ei　　..④

のように表現すると，③の表現には，

view＝ {r1／e1, r2／e2, ...,rn／en}

..⑤

なる表現が対応する．ここで，｛　｝は集合の意味であり，要素の順序に依存しないことも意味する．

この時，⑤の表現は，実体表現eiが役割riを果たしている，という，モデル作成者の認識をも表現していると解釈できる．その解釈は，モデル作成者が準備した役割パターンで識別された関連関係の枠組みにおいて認識される一つの視点の表現でもある．

その意味から，⑤の表現で得られる集合のインスタンスを，ここでは視点のインスタンスと考え，上記のように識別子viewを用いて⑤の集合のインスタンスを区別した．

ここでは，⑤のことを単にviewと呼ぶことにする．⑤は役割と実体表現との組の集合を表しているが，⑤の表現が③の表現に対応するには条件がある．その条件は，任意のviewには，唯一の役割パターンが対応していなければならないことである．⑤の例では，ｐがそれに対応する．

このようにして定められるviewは，Ｂ上の関連関係表現のインスタンスに対応することから，拡張された実体表現でもある，と見なすことができる．つまり，何らかの関連関係のインスタンスで表現される抽象的な実体が認識された，と考える．個々の関連関係の事象も拡張された実体と見なすのである．

このことにより，本来の実体表現を役割と組み合わせるばかりでなく，そうして得られた個々のviewをも，役割riと組み合わせることができると考える．このことは，Ｂ上の表現から見れば，いれ子状の表現形式を許すことに対応する．

例えば，

p1 (x1,x2) = “k1〔x1〕k2〔x2〕k3”,

p2 (x1,x2) = “k4〔x1〕k5〔x2〕k6”,

p3＝“e1”, p4＝“e2”, p5＝“e3”..⑥

などのパターンを準備して，さらに，役割

r1＝＜p1,1＞, r2＝＜p1,2＞,

r3＝＜p2,1＞, r4＝＜p2,2＞　　..⑦

とする時，以下のような視点view1,view2を考えたとする．

view1 = {r1／e1, r2／e2},

view2 = {r3／view1,r4／e3}　　..⑧

この場合，view1 に対応するＢ上の表現をview1', 同じくview2 に対応する表現をview2'とすると，view1',view2' はそれぞれ，

view1' = “k1e1k2e2k3”,

view2' = “k4k1e1k2e2k3k5e3k6”　　..⑨

のようになる．

2.4 モデルの記述と表現のクラス

前節で述べたようなviewの集合によってモデルを構築することを考える．すなわちモデルＭはviewの集合であり，モデルＭの任意の要素viewは，各々，唯一の役割パターンを持つ．

モデルＭに用いられた役割パターンの集

合をP，実体表現に用いられたパターンの集合をE，とすると，対象モデルや実現モデルは，それぞれ，M，P，Eの3つの組によって構成されることになる．

PとEが準備されると，PとEだけを用いて構成されるviewのクラスが定まる．したがって，PとEを定めると，基盤言語上での表現のクラスも定められることになるので，モデル理解者にとっては，PとEから構成される世界に局所化されることを意味し，知識負担を最小化することができると考えられる．

また，Pの各々の要素は関連関係の表現に対応し，Eの各々の要素は実体の表現に対応することから，実体—関連モデルのスキーマについての表現にも対応するモデルMが得られ，対象とする世界の実体や関連を的確に把握するのに効果がある．

さらに，モデルMの表現は，B上の部分に対応することから，個々の表現が指示する対象が，基盤言語Bの意味において，指示されることになる．

これらの特徴は，2.1 節で述べたモデル記述言語への要請をすべからく満足し，ソフトウェアモデリングのためのモデル記述言語として適切であると考えられる．

3. 対象モデルと実現モデルとの対応

前章において準備した内容は，対象モデル，実現モデルの両者に共通する，モデル記述言語の枠組みに関するものであった．ここでは，さらに，複数のモデル間の対応関係について述べる．

3.1 役割パスの導入

前章において定めたviewの構造について再度考察する．⑧の例を取ろう．⑧におけるview2 の中には，view1 が含まれている．ここで，view1 をさらにview2 の中でも表現することを考える．

そのまま展開すると，view2 は，

{r3／ {r1／e1, r2／e2} , r4／e3} ..⑩

となる．ここでは，表記上，⑩に相当するものとして，以下の表現を導入する．

view2 =

{r3／r1／e1, r3／r2／e2, r4／e3} ..⑪

⑪の要素は，⑩の集合の入れ子表現を，ちょうど多項式展開のように，展開した要素に対応する．⑪のように展開した表記法を用いて，⑩に置き換えて考えてみる．⑪は，⑩の表記上の便法でしかない．

この場合，⑪は（r3／r1），（r3／r2），（r4）という3つの役割を持つ役割パターンであると考えても，不都合が生じない．その場合のパターンは，

p6 (x1,x2,x3) =

“ k4k1 〔x1〕 k2 〔x2〕 k3k5 〔x3〕 k6 ” ..⑫

で，役割（r3／r1），（r3／r2），（r4）は，それぞれ，

（r3／r1）＝＜p6,1＞，

（r3／r2）＝＜p6,2＞，

（r4）＝＜p6,3＞ ..⑬

である．

ところで，view2 は，そのパターンに添って図2で示すような，構文木相当の木構造を持つと考えることもできる．

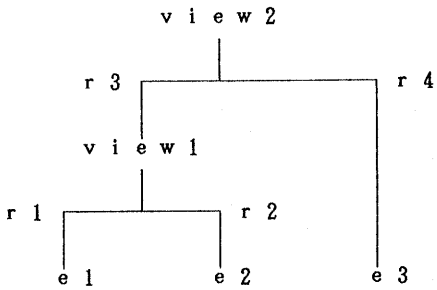

図2：viewの木構造図

図2を参考にすると，（r3／r1），（r3／r2），（r4）は，木構造における上から下へのパスに対応していることが分かる．この意味で，（r3／r1），（r3／r2），（r4）を，役割パスと呼ぶ．

すなわち，任意のviewを，役割パスと実体表現の対の集合である，と見なすことができる．ただし，その逆は言えない．任意の，役割パスと実体表現の対の集合が，常に，何らかのviewを構成するわけではない．任意のviewが与えられた時にのみ，それに対応するものとして，役割パスと実体表現の対の集合として表現できるだけである．

3.2 役割パスと役割パスの対応

対象モデルをM1，M1に用いた役割パターンの集合をP1，同じくM1に用いた実体表現の集合をE1とする．同様に，実現モデルをM2とし，M2の役割パターン集合をP2，実体表現集合をE2とする．

対象モデルM1において，モデル作成者が理解している個々の役割は，用いた役割パターンとその引数の場所によって確認できる．また，モデルM1は，役割パスと実体表現との組の集合としても理解できることから，モデルM1に登場する実体表現が果たす役割を，役割パスの集合として把握することができる．

すなわち，モデル作成者が対象モデルM1において定めている役割は，すべてM1における役割パスに反映してると考えられる．同様に，実現モデルM2における実体表現はすべからく，M2における役割パスによってその，実現世界において果たすべき役割が識別される．

このことは，対象モデルM1における役割パスを，実現モデルM2における役割パスに対応づけることによって，対象モデルの中の役割を，実現モデルの中の実体表現に付与することができることを示している．

役割付与の要素は，したがって，対象モデル側の役割パスから，実現モデル側の役割パスへの有向性のある対（有向対）の集合として定められる．

集合とする理由は，役割パスの一つ一つは，パスとしては独立であるが，役割パターンが，実は，パス同士の必然的な組み合わせを要求するからである．

例えば，図２の例でいうと，役割パスは３本で，

{r3／r1，r3／r2,r4 }　　..⑭

となる．パス同士の対応を示すには，これら３本についての有向対が示されねばならず，有向対の集合となる．

今，実現モデル側のパターンと役割とを次のようなものと考えたとする．

p7 (x1,x2) = “ if 〔x1〕 then 〔x2〕 ; ”,
p8 (x1,x2) = “ 〔x1〕 < 〔x2〕 ”,
p9 = “ C ”, p10 = “ D ”, p11 = “ return ”
　　..⑮

r5 = <p7,1>, r6 = <p7,2>,
r7 = <p8,1>, r8 = <p8,2>　　..⑯

さらに，実現モデルM2は，次のような集合であるとする．

M2 = {view3, view4} ,
view3 = {r5／view4, r6 ／return} ,
view4 = {r7／C,r8／D }　　..⑰

view3 の基盤言語上の表現view3'は，

view3' = “ if C<D then return ; ” ..⑱

となる．また，view3 の役割パスによる表現は，以下のようになる．

view3 =
{r5／r7／C,r5／r8／D,r6／return} ..⑲

今，view2 の（r3／r1）を（r5／r7）に割り当て，（r3／r2）を（r5／r8）に割り当て，（r4）を（r6）に割り当ててみようこの時の表現として，役割パスの有向対を⟶印で結び，以下のように表現する．

A = {r3／r1⟶r5／r7,
　　r3／r2⟶r5／r8, r4⟶r6}　　..⑳

これをモデル間の役割付与と考える．この例では，対象モデルM1における実体表現e1（で指示される実体）の果たしている役割（r3／r1）を，実現モデルM2で（r5／r7）の役割を果たしているC （で指示される実体）が担うことになる．

結果的に対象モデル側のe2の役は実現モデル側のD が演じ，e3の役をreturnが演じていることが分かる．このような対応関係は，役割付与の情報Ａを辿ることにより，容易に結びつけることができる．

ここで注意すべきは，結果的に実体表現から実体表現への役割付与が得られているが，役割付与Ａの行っている対応づけの構造は，けっして実体表現から実体表現への対応ではなく，役割パスから役割パスへの対応である点である．

このことから，対象モデル側での一つの実体（表現）が，実現モデル側での複数の実体（表現）にも対応できるし，複数の実体（表現）が，実現モデルでの同一実体（表現）に対応させることもできる．

4. ソフトウェア開発への応用方法

この章では，これらの方法を，ソフトウェア開発や保守の作業に，どのように用いると効果的であるのか，について述べる．

(1) 役割パターンと実体表現の集合の準備

一般に，対象とする分野や，専門分野によって，実体や，実体間の関連を把握するのに用いるパターン，役割認識などは，それぞれの分野毎に特徴的なものがあろう．それらの特徴的なパターンは，分野毎の歴史や分化背景により，つみあげられてきた財産にもなっていると考えられる．

プログラム言語の場合には，部品ライブラリィなどが財産として存在するので，部品ライブラリィの各々の部品のインターフェースが役割パターンと結びつけると効果的である．

このようにして，役割パターンの集合をあらかじめ，ある程度用意することができる．ただし，すべからく準備できるとも思えないので，徐々にパターンを登録するように考えるのが良い．

ここで大切なことは，パターンなら何でも良いのではなく，パターン変数部分に対応する実体（表現）が果たす‘役割’が，その分野において認識しうるものでなければならない点である．

(2) 役割付与情報の蓄積

対象側でのある役割パスが，実現側でのある役割パスに対応づけられる，とする役割付与の対応構造は，何度も述べるように実体と実体との対応の理屈から生じるのではなく，あくまでも，役割と役割の対応の理屈から来るものである．

では役割とは何か，が問題になるが，それは，それぞれの世界において，モデル作成者が，実体と実体との関連における役割として認識された概念であって，本研究の範囲を越えた問題である．

しかし，ひとたびモデル作成者や理解者が何らかの役割概念を捉えたとすると，ここで扱ったように，役割パターンとその中における場所の組によって識別され，役割概念を形式的に扱えるようになる．

また，役割付与の内容は，本来独立な世界同士を結びつけており，かなり経験的である．ソフトウェア開発における設計判断などに近い．設計判断の蓄積として活用すると効果的である．

(3) 実現モデルの生成への応用

対象世界に関連する P1,E1，および，実現世界に関連する P2,E2を準備しておき，さらに，役割付与情報 Aを蓄積しておく．P1,E1 のみを用いた対象モデルM1を記述する．それは自動的に基盤言語B1の上での表現となる．

役割付与情報の蓄積内容を用いながら，M1における役割パスを，M2 における役割パスに置き換えていく．詳細は文献〔3〕に譲りここでは割愛するが，役割パスの対応構造において，実現側の実体表現を補充するものを導入し，実現モデルが得られる．

5. おわりに

本研究では，対象モデルにおける役割パスを，実現モデルにおける役割パスに対応づける方法を述べた．この方法により，従来のモジュール分割では得られなかった情報分離を，任意のソフトウェアについて，行うことが可能となる．

対象モデルと実現モデルの分離と対応づけ，を目標とするソフトウェアモデリングの考え方を実現する，一つの有力な方法であると考えられる．

ここでは，簡単のために文字列パターンを用いて述べたが，役割パスの表現が可能なものであれば文字列パターンに限らず，役割パターンを定義できることも主張しておきたい．

謝辞

富士通㈱国際情報社会科学研究所の，北川会長，榎本所長からは，本研究に関する適切なアドバイスを頂いており，ここに深甚なる謝意を表します．

参考文献

〔1〕大須賀節雄他：ソフトウェアモデリング・夏期シンポジウム報告書，国際情報社会科学研究所，昭和60年９月．

〔2〕Chen,P.P. :The Entity-Relationship Model～Toward a Unified View of Data, ACM Trans.on Database Systems, Vol.1,No.1,March 1976,pp.9-36.

〔3〕Kobayashi,K.,Kimura,T.,and M.Oda :Role Assignment in Software Modeling, IIAS-SIS Research Report No.75, FUJITSU LIMITED,May 1987.