ソフトウェア工学 90－15
（１９９３．２．５）

Scenario/Role/Objectモデルに基づくソフトウェア構築法

片山　佳則，小林　要
（株）富士通研究所　国際情報社会科学研究所

ソフトウェアの再利用性の観点から，オブジェクトの部品使用としてのインターフェースを拡張し，オブジェクトの共有や機能の柔軟な改良／拡張機構を持った新しいオブジェクトモデル（Scenario/Role/Objectモデル)を提案している．このモデルの概要と，このモデルに基づいたオブジェクト指向プログラミング言語の実現について述べる．S-R-Oモデルは，現在のオブジェクト指向モデルにおけるオブジェクトの再使用性に関する問題を解消するために提案したものである．このモデルは，情報の的確な分離技術を考慮して，ソフトウェアを理解容易で操作容易なものとすることを目的としている．

A Software Construction Method Based on Scenario/Role/Object Model

Yoshinori Katayama　Kaname Kobayashi

FUJITSU LABORATORIES LTD.
140 Miyamoto, Numazu-Shi, Shizuoka, 410-03 Japan

We have been developing a software construction model (S-R-O Model) for reusing objects with role assignment mechanisms. S-R-O model has powerful assignment mechanisms that provide excellent support for design and implementation, including role definition for existing object.

In this paper, we present a quick summary of the S-R-O model, and show some detail on how reuse of software is accomplished in S-R-O model by role assignment mechanisms. We draw a preliminary comparison between the functionality's provided by S-R-O model for reusing software, through the role assignment.

1．はじめに

我々は，ソフトウェアの再利用性の観点から，オブジェクトの部品使用としてのインターフェースを拡張することを始まりとして，オブジェクトの共有や機能の柔軟な改良／拡張を実現するための新しいオブジェクトモデル (Scenario/Role/Object モデル)を提案している[1]．このモデルは，プログラムとして実装されるクラスオブジェクトおよび実行環境で動作するインスタンスオブジェクトの両方を再利用のターゲットとしている．

既存のオブジェクト指向プログラミングがベースにしているオブジェクトモデルは，これまでのプログラミング言語が基にしているモデルの中でも高い評価を得ている[2),3)]．しかし，ソフトウェアの再利用の立場からみると，プログラムの記述面／実行面における広範性において，幾つかの問題点が既存のモデルにも存在している．

本稿では，ソフトウェアの再利用の観点から捕えた既存オブジェクト指向モデルの問題点を簡単にまとめ，これらの問題点に対処するための新しいオブジェクトモデルであるS-R-Oモデルのポイントをまとめる．さらに，S-R-Oモデルを実現したオブジェクト指向プログラミング言語におけるソフトウェアの再利用およびオブジェクトの共有の事例を示す．

2．既存オブジェクト指向モデルの問題点

ここでは，ソフトウェアの再利用性の観点から，大きく３つの問題[1]（記述情報と理解性，関係構造とその機能，実行オブジェクトの共有）を取り上げる．

（1）記述情報と理解性

ソフトウエアの再利用は，記述を変更しないで利用することが基本である．このため既存のオブジェクトモデルでは与えられたクラスオブジェクト群に対して，新たにサブクラス(差分)を追加することでプログラム開発が進められる[4]．これは，クラスが持つ階層構造や各オブジェクトの記述をそのまま受け入れ，すべてのオブジェクトを取り込みながら開発を進めなければならないことでもある．つまり，応用に依存したオブジェクトと応用に依存しない要素的なオブジェクトとの区分が明確でなく，プログラムの改良や再利用においては，常に全オブジェクトの絡み合い方を理解している必要がある．これは，情報の分離方法[5)]への対処の不十分さに原因がある．

また，表現内容に関して，各応用領域におけるオブジェクトの協調関係，全体の筋書き(ストーリー)，処理順序等の関係を容易に把握できない．これらの関係を扱うためには，応用領域のために開発する個々のオブジェクトと同様に，これらを管理するオブジェクトを開発者自身が個別に作成しなければならない．さらに，複合関係等を用いてプログラミングを進める場合には，複合部品として利用できるように既にあるオブジェクトを改良するための中間的なクラスオブジェクトを予め用意しなければならない．これらの中間的なクラスは，対象としている応用領域に依存したものであり，クラス階層をより煩雑にすることになる．

（2）関係構造とその機能

オブジェクトの利用は，既存のオブジェクトモデルでは，クラスオブジェクトのレベルで決められる．部分的な変更は，サブクラスの作成によって継承を制限する形で実現される．この継承によらない利用としては，複合表現と呼ばれる内部変数へのオブジェクトのバインドやクラス階層とは異なるpart-of的関係などの導入が進められ，部品的な関係構造による利用を実現していた[6),7)]．これらの継承／非継承関係は，対象としている応用領域に依存している．従って，オブジェクトが適応される応用領域に基づいて継承／非継承が柔軟に対応できなければならない．既存のモデルでは既に作成された関係構造による継承／非継承が強要される．このようなモデルでは，異なる応用領域からのオブジェクトの広範な利用には，利用する側の順応性や柔軟性が要求される．

（3）実行オブジェクトの共有

既存のオブジェクトモデルでは，実行環境はクラス階層を用いて，クラスレベルで実現する．異なる実行環境は，異なるクラスとして実現しなければならない．従って，クラスから実体化された一つのインスタンスオブジェクトに複数の実行環境を与えることや動的な機能改良は不可能である．つまり，一つのインスタンスオブジェクトが実行環境を意識して柔軟な対応をとることはできないため，実質的に実行オブジェクトの共有は困難である．各オブジェクトが状況に応じた複数の実行環境を柔軟に処理できることが重要になる[8)]．多重継承[9)]により，クラスの記述レベルで複数のメ

ソッドを持たせても，実行環境を区別するためには，すべての実行環境を把握し，処理内容の整合性をクラスレベルで意識する必要がある．デリゲーション[10]は，実行環境等のの複雑化を解消する一つの方法として捕えられるが共有化のモデルとしては十分とはいえない．

3．S-R-Oモデルの概要

我々は，前節の問題点を踏まえて，ソフトウェアの利用性を考慮した新しいオブジェクトモデル＝Scenario-Role-Object(S-R-O)モデルを提案している[1]．このS-R-Oモデルは，Scenarioユニット，Roleユニット，Objectユニットによって応用プログラムを実現するものである．

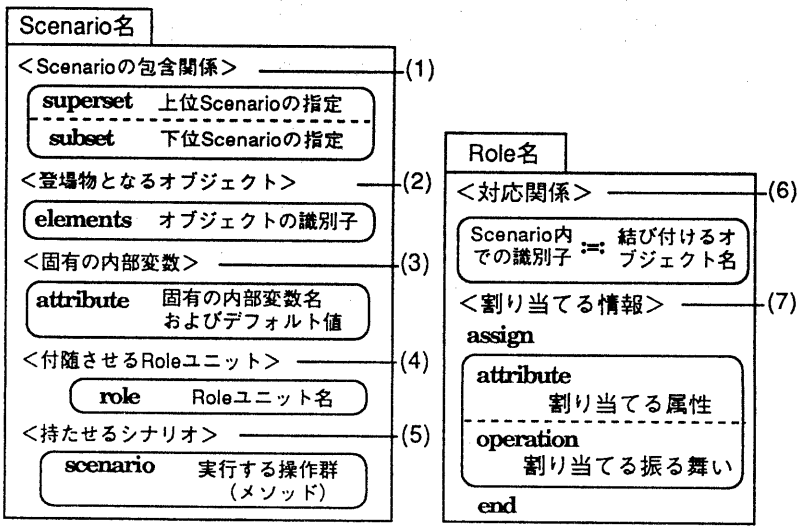
3.1 S-R-Oモデル

S-R-Oモデルでは，応用領域のプログラム開発は，シナリオ(筋書き)とそこに現われる登場物を記述することから始められる．つまり，応用領域の分析はシナリオと登場物を明確にすることである．これらを記述するのがScenarioユニットである．そして，各登場物に割り当てるオブジェクトの情報をRoleユニットとして記述する．これらの記述でプログラム開発が進められる．

図1にScenarioユニットとRoleユニットの記述形式を示す．図1(a)のようにScenarioユニットは，包含関係(図1(1))，登場物となるオブジェクトの識別子(図1(2))，固有の内部状態変数(図1(3))，付随させるRoleユニット(図1(4))，実行するシナリオ(図1(5))からなっている．

このScenarioユニットの記述では，シナリオ(図1(5))と，そのシナリオに現われる登場物の指定(図1(2)の識別子の記述)が特徴的である．シナリオの状態変数の記述はこれまでのオブジェクトが持っていたものと変わらない．実行するシナリオも，これまでのメソッドの記述に対応している．しかし，Scenarioユニットでは登場物の指定を行なうため，記述するシナリオは，特に処理順序を明確にすることを基本として各登場物へのメッセージが記述される．このScenarioユニットのシナリオを実行するために，必要とする登場物とそれにあてはめるオブジェクトの結びつきを実現するのがRoleユニットである．

図1(b)のようにRoleユニットは，付随しているScenarioユニットで登場物として示されたオブジェクトの識別子と実際に結び付けるオブジェクトとの対応関係(図1(6))，結び付けるオブジェクトにそのScenarioユニットのシナリオ実行のためだけに割り当てる情報を記述する(図1(7))．このように，シナリオに登場させるオブジェクトは，予め与えられるObjectユニットから適切なオブジェクトを選び出し，それに対する固有の改良をRoleユニット部分で行なったうえで連結される．Roleユニットに記述された割り当て情報は，そのScenarioユニットのシナリオを実行する時のみ，選んだオブジェクトに動的に付け加えられる具体的な内部状態変数とメソッドの集まりである．



(a) Scenarioユニットの記述形式　(b) Roleユニットの記述形式

図1　ScenarioユニットとRoleユニットの記述形式

Objectユニットは，応用領域のプログラム開発に対してシステムが装備しているオブジェクトである．これらのオブジェクトは，直接改良されることはなく，必要な改良はすべてScenarioユニットに付随させるRoleユニットによって，各Scenarioユニットごとに独立に行なわれる．従って，同一のオブジェクトに対してScenarioユニットごとに自由な改良／変更を行なえる．

また，各Scenarioユ

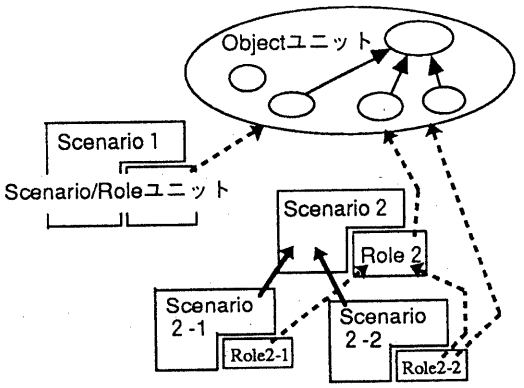
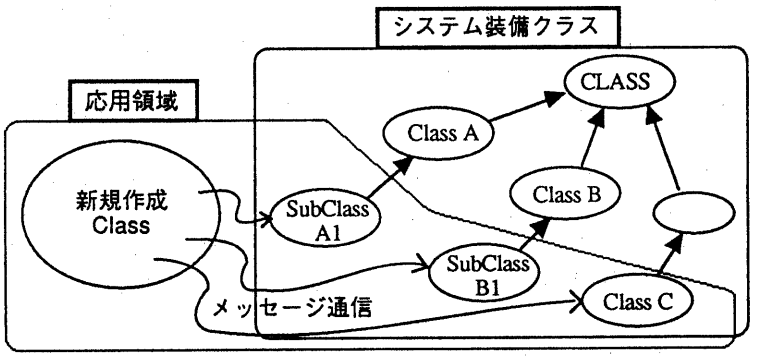


図2　S-R-Oモデルの概要

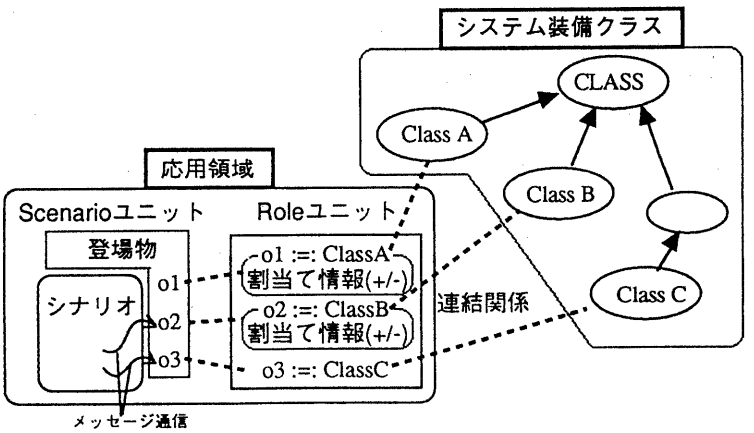
ニットのシナリオには，複数の別のScenarioユニット(サブScenarioユニット)の処理手順を記述することができる．これらのサブScenarioユニットの処理順序を明確にすることがそのサブScenarioユニット間の協調関係を決めることになる．このサブScenarioユニット間で共有させたいオブジェクトを上位のScenarioユニットの登場物として管理することで，サブScenarioユニット間の協調作業を実現できる．オブジェクトの共有は，Roleユニットで記述するオブジェクトの結びつき先として，包含関係の上位のScenarioユニットで記述されている登場物を指定することで実現できる．これにより動作時に上位のScenarioユニットの登場物にバインドされているオブジェクト自身を複数のサブScenarioユニットが共有する．さらにこの共有の際にも，各サブScenarioユニットの登場物として必要な固有の割り当て情報が，そのサブScenarioユニットに付随するRoleユニットに記述できる．これによって，同じオブジェクトを共有しながら，その共有オブジェクトに動的な機能付加／変更が行なえる．

各Scenarioユニットと，それに付随させるRoleユニットを一つのモジュールとして独立なオブジェクト集合を構成し，応用プログラムが実現される．図2にS-R-Oモデルの概要を示す．

図2のScenario 2におけるサブScenarioユニット



（1）既存のオブジェクト指向モデルによるプログラミング



（2）S-R-Oモデルによるプログラミング

図3　これまでのモデルとS-R-Oモデルの違い

の各RoleユニットからScenario 2のRoleユニットへの対応付けは，それらのサブScenarioユニット間のオブジェクトの共有を示している．

3.2 S-R-Oモデルとこれまでのモデルとの違い

ここでは，2節の問題点の解消をS-R-Oモデルのポイントとしてまとめ，S-R-Oモデルとこれまでのオブジェクト指向モデルとの違いを示す．

記述情報と理解性については，S-R-Oモデルでは，Scenario/RoleユニットとObjectユニットを完全に分離して開発を進めることによって解消できる．これまでのオブジェクト指向モデルでは，図3(1)のようにシステム装備クラスと各応用領域に依存して作成されたクラスとが明確に分離されていない．図の中のサブクラス(SubClass A1, B1)は，応用領域に依存して改良を加えたクラスである．これらが他のクラスと区別無くクラス階層に追加されてしまう．このため開発を進めるに従ってクラス階層は煩雑になってしまう．しかし，S-R-Oモデルでは，図3(2)のようにRoleユニットが各Scenarioユニットに付随して必要な変更分の記述を行なうため，システムとして装備されるObjectユニットは直接変更されない．従って，一般的な要素オブジェクトと各応用領域に依存した応用オブジェクトとは，明確に区別される．Roleユニットの割当て情報が，これまでのモデルでのサブクラス(SubClass A1, B1)の記述に対応しており，これらが応用領域側として管理される．これによって，オブジェクトの構造が煩雑になるのを防ぐだけでなく，プログラムの改良や拡張が，Scenarioユニットの変更やRoleユニットの変更として，必要な部分で効果的かつ容易に行なえるようになる．

また，表現内容に関しては，S-R-Oモデルでは，Scenarioユニットが全体の筋書き，各サブScenarioユニット間で共有する登場物および協調関係を扱うことから，各応用領域ごとのオブジェクトの関連の把握が容易になる．

関係構造とその機能については，S-R-Oモデルでは，Roleユニットに記述される割り当てメソッドの起動が，それが付随しているScenarioユニットだけに規定され，他のScenarioユニットからは起動できない．これによって，非継承メソッドを実現する．再利用のために必要な順応性や柔軟性のテクニックは，シナリオの関係と各Roleユニットの対応付けおよび割り当て情報として明確化される．Roleユニットの記述の継承は，常にそのScenarioユニットの包含関係の下に行なわれ，互いのScenarioユニットの登場物に同じ識別子を持ち，実行時にバインドさせるオブジェクトを共有することで実現される．

動作実体の共有は，S-R-Oモデルでは，Scenarioユニットの包含関係の実現と，各Scenarioユニットの登場物へのオブジェクトの対応関係の記述で実現できる．包含関係の上位のScenarioユニットの登場物をサブScenarioユニットのRoleユニットからの対応先として指定することで，サブScenarioユニットでのオブジェクトの共有を実現する．つまり包含関係の上位のScenarioユニットの登場物はサブScenarioユニット間の共有オブジェクトとなりえる．さらに，各Roleユニットにおいて固有の情報を付加できるため，実行時にこの登場物にバインドされる動作実体としてのインスタンスが，シナリオごとに機能を動的に変更しながら必要な処理を進められる．これにより同じインスタンスが複数の実行環境を柔軟に処理できる．

オブジェクトを共有しないで独立に利用する場合は，各Roleユニットで登場物に必要なオブジェクトを直接対応付ければよい．

4．S-R-Oモデルのプログラミング実装

ここでは，S-R-Oモデルの特徴を理解するための簡単な実装例として，棒グラフを実現している各要素オブジェクトを用いて，棒グラフの各棒の連続的な上下運動により，ウエーブを実現するプログラムの開発を取り上げる．ウェーブ動作の概観を図4に示す．

ウエーブ動作のプログラムであるScenarioユニットとRoleユニットをそれぞれ図5，6に示す．Scenarioユニット(wave-10)の登場物としてframe, w_window,w1,w2,w3, w4, w5, w6, w7, w8, w9, w10

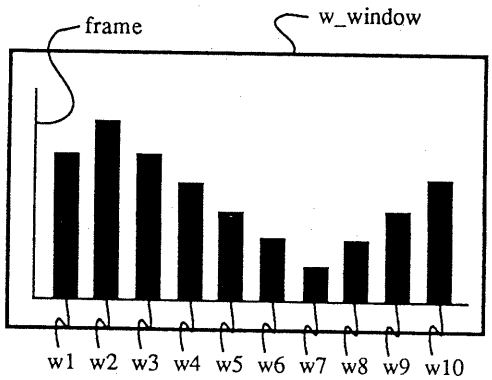


図4　wave_10動作の概観

Scenarioユニット

```
defscenario wave_10 ::
  elements      w_window, frame,
     w1,w2,w3,w4,w5,w6,w7,w8,w9,w10 ;          (1)
  attribute     element(10),
     datalist([1,2,3,4,5,4,3,2,1,2,3,4,5]) ;
  role     wave_role ;
  scenario
     set() : @(element,E),
       (w_window <- set_create(E) ),
       (frame <- init(E,w_window) ),
       (^ element_set([w1,w2,w3, ... ,w10],0) ), !;

     doit() :
       @(datalist,Data),
       (Data=[] ; Data=[X | Res], #(datalist,Res),
         (w1 <- propagate_set(X,w_window) ),
         (^ doit) ), !;
     . . .
```

図5　Scenarioユニットの記述例

Roleユニット

```
defrole   wave_role ::
  w_window :=: gwindow ------------------------(1)
    assign
     operation
      set_create(E) : Wis14*E+35,  ------------(2)
        #(size,[50,10,300,W]), (^^ create)
    end
  frame :=: lines ------------------------------(3)
    assign
     attribute   pensize([2,2]), ... ;
     operation
      init(E,Window) : ...  ------------------(4)
        (^^ put(points,[(A,Left),...])), ...
    end
  w1 :=: fillbox -------------------------------(5)
    assign
      superset  wavebox
     end
   . . .
  w10 :=: fillbox ------------------------------(5)
    assign
      superset  wavebox
     end
wavebox :=: non
  assign
   attribute   right, ... ;                     (6)
   operation
     propagate_set(X,W) : @(point,P), ...
        (W <- add_pic(self) ),
        (R <- propagate_set(X1,W) ), ...
    end
```

図6　Roleユニットの記述例

があげられている(図5(1)). w_windowは，このウエーブシステムの動作を表示するためのウインドウオブジェクトである(図4). 従ってRoleユニット(wave_role)では，その識別子を基本的な要素オブジェクトであるgwindowオブジェクトに対応させ(図6(1))，wave_10シナリオのための固有のメソッド(set_create：棒グラフの要素数のデータを受け

```
      . . .
  w1 :=: filloval
    assign
      superset  wavebox                         (a)
     end
   . . .
  w10 :=: filloval
    assign
      superset  wavebox
     end
      . . .
```

(1) 楕円要素waveのための変更部

```
      . . .
wavebox :=: non
  assign
   attribute   right, ... ;
   operation
    propagate_set(X,W) : @(point,P), ...
       (W <- add_pic(self)),
       (R <- propagate_set(X1,W)), ...
    get_description(fillfg(C,MyD)) :  ----------(b)
       @(point,P), @(description,MyD),
       (P=0,C=white;P=1,C=cyan; ... )
  end
```

(2) カラーwaveのための変更部

図7　Roleユニットの変更

取ってウインドウのサイズを計算し生成するメソッド)を割り当てている(図6(2)). frameは棒グラフの座標軸を表示するオブジェクトである(図4). wave_roleにおいてlinesオブジェクトに対応させ(図6(3))，wave_10シナリオのための固有の属性(pensize等)とメソッド(init等)を割り当てている(図6(4)). w1からw10が棒グラフの各棒オブジェクトである(図4). wave_roleで，それぞれfillboxオブジェクトに対応させている(図6(5)). これによりfillboxオブジェクトを棒オブジェクトとして利用する．ここでは棒オブジェクトとして，wave_10シナリオが必要とする上下運動を実現させるための操作(propagate_set等)と右隣を知るための固有の属性(right)等の割り当ても行なっている．これらの割り当ては，w1からw10のすべてに共通であるため，waveboxとしてwave_role内でモジュール的に記述し，それぞれが参照することで必要な情報の割り当てを効率よく実現している(図6(6)). 図5のシナリオのようにScenarioユニットのシナリオには，その応用領域における各登場物の実行の起動や順序関係が主に記述される．ScenarioユニットやRoleユニットにおけるoperationやscenarioの詳細な記述方法については，文献7)を参照.

プログラムの基本的処理手順に影響しない改良は，Scenarioユニットを操作することなく，各Roleユニットで対応させるオブジェクトを変更するだけで実現できる．例えばウエーブでは，各棒に対

Scenarioユニット

```
defscenario two_way_wave ::
  subset      wave_right, wave_left ;                    (1)
  elements    w1, w2, w3, ..., w10, frame, w_window ;
  attribute   element(10),
    datalist([1,2,3,4,5,4,3,2,1,2,3,4,5,6,7,8,7,6]) ;
  role        two_wave_role ;
  scenario
   set() :
     @(element,E),
     (w_window <- set_create(E) ),
     (frame <- init(E,w_window) ),
     (^element_set([w1,w2,w3, ... ,w10],0) ), ...
   doit() :                                              (2)
     @(datalist,Data),
     (Data=[] ; Data=[X|Res], #(datalist,Res),
       (wave_right <- put(data,X) ),
       (wave_left <- put(data,X) ),
       (wave_right  <- doit),
       (wave_left <- doit), (^show([w1,w2,w3, ... ,w10])),
       (^ doit) ), !;
   . . .
```

図8　両方向waveを実現するScenarioユニットの記述

応させたfillboxをfillovalに変更する(wave_roleにおける w1,w2,w3, ...,w10 の対応先のオブジェクトをfillovalに変更する[図7 (1)の(a)])だけで棒グラフのウエーブプログラムを楕円要素によるウエーブプログラムに変えることができる．さらに同じScenarioユニットに対して，Roleユニットで割り当てる情報内容を変更することによる応用プログラムの改良／拡張も行なえる．例えば，各棒の対応先オブジェクトは変更しないで，割り当てる情報だけを変更する(図7 (2)の(b)のようにwaveboxに割り当てる情報の一部を追加／変更する)ことで，棒グラフの棒の上下によるウエーブから色の変化の伝播表示のプログラムを容易に実現できる．

これらは，Scenarioユニットに記述されたシナリオ自体に変更を加えることなく付随するRoleユニットの変更のみで応用プログラムの改良や拡張を行なった例である．同様に，Scenarioユニットだけを変更して，実行手順や表示手順などのシナリオを個別に改良することもできる．これらが，S-R-Oモデルにおける要素的オブジェクトおよびScenarioユニットやRoleユニットの再使用／改良／拡張の典型的な例である．

また，S-R-Oモデルにおける実行オブジェクトの共有の例として，前述のウエーブの例に対して，左と右から同時にウエーブを行なうプログラムを容易に実現できることを示す．これには，右から左への波を描くScenarioユニットと左から右への波を描くScenarioユニットをサブScenarioユニットとして持つようなScenarioユニットを実現する(図8 (1)および図9 (a)と(b)の(1))．各サブScenarioユニットでは，右から左と左から右への波をそれぞれ独立に実行するものである．そしてこれらの各サブScenarioユニットで，ウエーブさせ

Scenarioユニット

```
defscenario  wave_right ::
 superset two_way_wave ;                                 (1)
 elements  w1, w2, w3,w4,w5,w6,w7,w8,w9, w10 ;           (2)
 attribute data(0);
 role      wave_right_role ;
 scenario
  left_set([X,Y|Res]) :
    (X <- p_right(Y) ),
    (^^ left_set([Y|Res]) );
   ...
  doit() :
    @(data,X),
    (w1 <- propagate_set(X)).
```

Roleユニット

```
defrole  wave_right_role ::
 w1 :=: *, fillbox :                                     (3)
  assign
   superset wavebox ;
  end
  . . .
 wavebox :=: non :
  assign
   attribute right() ;
   operation
     p_right(R) : #(right,R);
     propagate_set(X) :
         @(rpoint,P),(P=X ;  ... ;
  end
```

(a) 左から右の波を実現するScenarioユニット(wave_right)

Scenarioユニット

```
defscenario  wave_left ::
 superset two_way_wave ;                                 (1)
 elements  w1, w2, w3,w4,w5,w6,w7,w8,w9, w10 ;           (2)
 attribute data(0);
 role      wave_left_role ;
 scenario
  right_set([X,Y|Res]) :
    (X <- p_left(Y)),
    (^^ right_set([Y|Res]) );
   ...
  doit() :
    @(data,X),
    (w10 <- propagate_set(X)).
```

Roleユニット

```
defrole  wave_left_role ::
 w1 :=: *, fillbox :                                     (3)
  assign
   superset wavebox ;
  end
  . . .
 wavebox :=: non :
  assign
   attribute left() ;
   operation
     p_left(L) : #(left,L);
     propagate_set(X) :
         @(lpoint,P),(P=X ;  ... ;
  end
```

(b) 右から左の波を実現するScenarioユニット(wave_left)

図9　両方向waveを実現するためのサブScenarioユニットの記述

る棒オブジェクトを共有させることにより，同一ウインドウ上での協調的ウエーブを実現する．協調性を実現するための共有の指定は，各サブScenarioユニットが上位Scenarioユニットと同じ登場物(elements)の指示を行ない(図9 (a)と(b)の(2))，それらの登場物に同じオブジェクトをバインドさせることである．このプログラミング言語では，サブScenarioユニットに付随するRoleユニット内の対応関係記述の先頭に書かれている'*'がその役割を果たしている(図9 (a)と(b)の(3)参照)．この'*'によって上位Scenarioユニットの登場物のオブジェクト自身がバインドされ共有利用を実現する．

このようにして，上位Scenarioユニットの登場物(elements)に列挙されたすべての棒オブジェクトを，互いのサブScenarioユニット間で共有し，左右からの協調ウエーブを実現できる．S-R-Oモデルではこのようなオブジェクトの共有による協調的プログラムが容易に記述できる．包含関係の上位にあるScenarioユニットには，各サブScenarioユニットの協調動作方法がシナリオとして記述される(図8 (2))．

各サブScenarioユニットだけを個別に起動するときは，その登場物にバインドするオブジェクトとして，'*'の次のfillboxへの対応関係が適応され，固有の棒オブジェクトが生成される．

この協調ウエーブは，Scenarioユニットの包含関係によるオブジェクトの共有でなく，前のwave_10シナリオを要素的オブジェクトとして，Roleユニットの対応先オブジェクトにバインドさせ，同じシナリオを異なる登場物(右からと左からのウエーブ)として再使用することで実現することもできる．ただしこの場合は，wave_10シナリオ内の登場物(w1~w10)は独立で共有されない．

5．まとめ

S-R-Oモデルは，既存のオブジェクト指向プログラミングの持つ利点を生かしてオブジェクトの利用性を追究したものである．S-R-Oモデルでは，Roleユニットによってオブジェクトの広範な再利用を実現するだけでなく，インスタンスオブジェクトの共有利用を積極的に進めることができ，同じオブジェクトでも，動作場面によって様々な振る舞いを柔軟に持たせることができる．

また，Scenarioユニットによる全体の筋書きや協調関係の内容やRoleユニットによる応用側オブジェクトと要素側オブジェクトの分離などの実現によって，把握性を向上させること，およびScenarioユニット，Roleユニットそれぞれの変更によるプログラムの容易な改良／拡張が，ソフトウェアの利用性を向上させる．

さらには，S-R-Oモデルでは，プログラムの記述方法がScenarioユニットとRoleユニットによって規制され，そのもとにプログラム開発が進められる．これは，プログラムの標準化の役割も果たし，再使用性という立場からプログラミング技術を向上させるものでもある．

[参考文献]

1) 片山 佳則，小林 要："ソフトウェアとしての利用性を指向した新しいｵﾌﾞｼﾞｪｸﾄﾓﾃﾞﾙ(S-R-Oモデル)" 情報学シンポジュウム'93, Jan 1993

2) B. Meyer : "Object-oriented Software Construction" Prentice Hall, 1988

3) D. C. Rine & B. Bhargava : "Object-Oriented Computing" Computer Vol.25,No.10, 1992

4) A. Goldberg & D. Robson : "Smalltalk-80: The Language and Its Implementation" Addison- Wesley, 1983

5) 小林 要，木村 高久，織田 充："ソフトウェアの対象モデルと実現モデルの対応構造に関する一考察" 情報処理学会ソフトウェア工学研究会 62-3, 1988

6) R. K. Raj & H. M. Levy : "A Compositional Model for Software Reuse" Proc. European Conference on Object-Oriented Programming, ECOOP'89, July 1989

7) 片山 佳則："多視点に基づくオブジェクト指向表現システム" 情報処理学会ソフトウェア工学研究会 73-5, 1990

8) 本田 康晃，渡 滋，所 真理雄："適応化コンポジション" ｺﾝﾋﾟｭｰﾀ ｿﾌﾄｳｴｱ, Vol,9, No.2 ,1992

9) D. A. Moon : "Object-Oriented Programming with Flavors" Object-Oriented Programming, Systems Languages and Applications (OOPSLA) '86 Conference Proceedings, Published as SIGPLAN Notices Vol.21, No.11, 1986

10) H. Lieberman : "Using prototypical objects to implement shared behavior in Object Oriented Systems" Object-Oriented Programming, Systems Languages and Applications (OOPSLA) '86 Conference Proceedings, Published as SIGPLAN Notices Vol.21, No.11, 1986