

Scenario-Role-Objectモデルにおける分析／設計／実装

片山 佳則

(株) 富士通研究所 情報社会科学研究所

ソフトウェア開発のさまざまなフェーズに対して、オブジェクト指向方法論の導入が検討されている。しかし、それらの方法論に対して、いくつかの問題点も指摘されている。我々は、既に、オブジェクト指向プログラミングに関わる問題を克服するモデルとしてScenario-Role-Objectモデルを提案している。ここでは、これまでの方法論に対する問題を踏まえて、このプログラム開発のためのS-R-Oモデルをソフトウェア開発プロセスの上流工程である要求分析から進める方法に展開し、ソフトウェア開発プロセスのモデルとしての拡張を検討し、分析から設計／プログラミングまで一連の流れを持たせる方法とその特徴を述べ、考察する。

Analysis/Design/Implement on Scenario-Role-Object Model

Yoshinori Katayama

FUJITSU LABORATORIES LTD.

Institute for Social Information Science (ISIS)

9-3, Nakase 1-chome, Mihama-ku, Chiba-shi, Chiba, 261 Japan

In the last years, the Object-Oriented methodology has been adopted at the level of programming and design. However, this methodology does not lead a perfectly reasonable mechanism. We have already proposed Scenario-Role-Object Model that worked out a number of problem on Object-Oriented Programming.

This paper is focused on the requirement analysis phase of the software development process. We apply Scenario-Role-Object Model to the requirement analysis phase. Through this utilization, we have realized the effectiveness of the Scenario-Role-Object Model approach in comparison with the software development process.

1. はじめに

オブジェクト指向方法論は、OOA、OOD、OOP/OOI等の分類で、ソフトウェア開発のさまざまなフェーズに取り込まれている。各フェーズは、その応用領域の理解に重点を置いたもの、実装を重視しているものなどその目的に違いがある。しかし、すべてオブジェクトモデルを中心として解析を進める点が共通している。一般に、このようなオブジェクト指向方法論の導入は、モデルの共通化により開発プロセスの各フェーズに一貫性を持たせることができると主張されている[1]。しかし、評価基準の不明確さも含めて、オブジェクトの決め方やフェーズに渡るモデルの推移性等、オブジェクト指向方法論導入に関わる問題が指摘されている。これらの問題を部分的に対処するために、“シナリオ”を重要視したいくつかの新しいモデルの提案[3,4]も進められている。

我々は、既にオブジェクト指向プログラミングに関わる問題を克服するモデルとして、S-R-Oモデルを提案している[2]。本稿では、特に要求や仕様の変化に柔軟に対応できることを目的として、このS-R-Oモデルをプログラミングのモデルからソフトウェア開発プロセスのモデルに拡張することを検討し、分析から設計／プログラミングまで一連の流れを持たせる方法とその特徴を述べ、考察を加える。

2. オブジェクト指向プログラミングの問題点

オブジェクト指向概念は、理論的基盤というより実際の見地から生まれているため、有用性を数多く含んでいる[5]が、一方で実際のソフトウェア開発プロセスに対応させると幾つかの問題を捕えることができる。ここでは、はじめにプログラミング作業の結果であるプログラムの変更容易性の観点から、プログラミングにおける問題点をまとめる。

2-1 実装側と応用側の機能の混在

オブジェクトは階層構造により機能の継承を受ける。これは、オブジェクト指向概念の一つの大きな特徴でありその有効性は、さまざまな場面で論じられている。ここでの問題は、継承メカニズムによってオブジェクトが持っている本来の機能（実装機能）だけでなくそのときの応用に依存した機能（応用機能）も同時に同じメカニズムで継承されることである。

一般にオブジェクトに持たせる機能をメソッドとして実現する場合には、単純化や汎用化を常に意識する必要がある。これらの概念を基に生産性の向上が計れる。これらを前提とした上で、オブジェクトの機能として実装側と応用側の分離を明確にしなければならない。これらが守られない限り、ad hocで変更し難いオブジェクトが多数実現される。さらに、このような状況において、オブジェクトの機能継承メカニズムでは、実装機能と応用機能をその区別なく継承させる。ここで、複数の応用機能が誤動作な

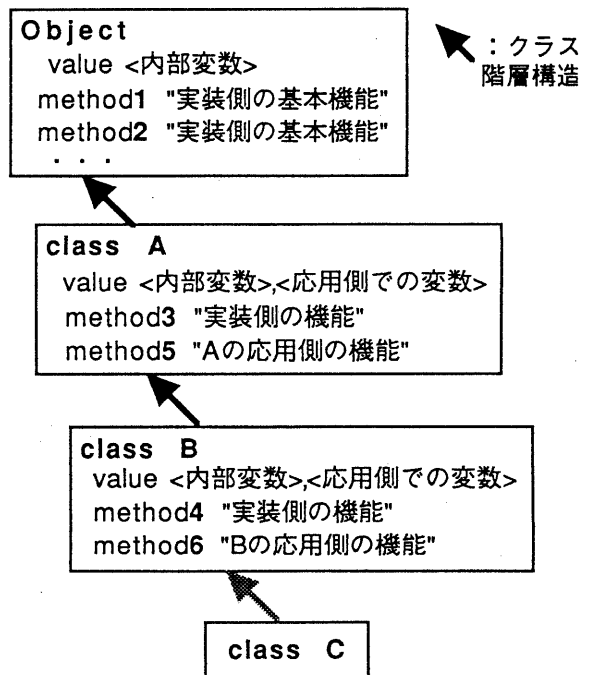


図1 オブジェクトの機能継承

くその機能を達成させるには専門的なプログラミングのテクニックが不可欠になる。さらに対象外の応用機能の継承が、現在対象としている応用領域側のオブジェクトの役割を不明確にすることになる。

例えば、図1のクラス階層で、class Bのサブクラスとしてclass Cを実現することで、実装機能(method1~4)を引き継がせることができるが、同時にclass AやBそれぞれの異なる応用機能(method5, 6)も引き継いでしまう。この例では、各メソッドに実装側/応用側の区別を付けているが、実際の記述では判定できない。従って、class Cの対象とする応用に依存した機能を的確に把握できない。

2-2 オブジェクトの参照関係の不透明さ

オブジェクトは、メッセージ送信によって他のオブジェクトと連携して処理を進める。さらに、そのメッセージ送信先も機能継承以外の場合は動的に決めることができる。従って、応用機能における各振る舞いの参照環境や参照場面を容易に把握できない。応用領域側に関連するオブジェクトの分けがができない。このため、応用機能にかかわる記述内容の変更は、他の応用領域に連鎖的な反動と混乱を引き起こす場合がある。

2-3 欲しい機能の実現テクニック

実装機能と応用機能の区別に関係なく、オブジェクトとして実現したい機能が、既に継承関係内などに拡散している場合、それらの機能を抜き出してオブジェクトの機能を実現することは困難である。

例えば、図1のオブジェクト構造に対して、class Cとしてmethod1~3とmethod6の機能だけを持ったオブジェクトを実現するには、メソッドごとのdelegationメカニズムを用いるなどの必要以上の手間がかかる。

実装機能の変更はそれが要素的なものであることから、追加されることがあるが、直接記述内容を変更されることはない。従って、機能に変更が起こるのは、ほとんどが応用機能に関してである。このことから、応用側と実装側の機能の実現に同じメカニズムを用いることは、プログラマへの負担になることがわかる。

変更への対応性というポイントでは、この他にも設計者やプログラマのテクニックで処理される課題は、トランザクション単位の設定等、いくつか考えられる。ここで挙げた問題はそれぞれ応用機能ごとにプログラマがその知識と経験によりad hocな方法で対処できる。しかしこれでは、開発プロセス全体を見通した場合、変更などの対応は各プログラマの専門知識や経験に強く依存する。これを避けるためには、応用機能ごとの状況の変化を、プログラマのテクニックで対処するのではなく、モデル上の操作として扱えるようにすることが必要になる。

特に、ここに挙げた問題を回避するには、第一にどこからでもメッセージを受けることで処理を進めるオブジェクトとしての記述内容の再検討が必要である。つまり、実装機能としての共有的なオープンな記述と、応用機能としての依存的なクローズな記述を、オブジェクトとしてでなく応用領域側の分析とともに区別して扱えるメカニズムが必要である。

S-R-Oモデルは、このような改善を実現するモデルとして位置付けられ、ここに挙げた問題を解決できるモデルである。

3. S-R-Oモデルの特徴

オブジェクト指向プログラミングが持つメカニズムを維持し、その実現におけるいくつかの問題に対処し、応用領域側のモデル化能力とソフトウェアの生産性の向上に大きく貢献するために提案したものがS-R-Oモデルに基づくプログラミングである[2]。S-R-Oモデルは、これまでのオブジェクトの記述を、応用機能の記述を

受け持つRoleユニットと実装機能の記述を受け持つObjectユニットに分けて取り扱う。この2つを結び付けたものを具体的な登場物として、応用領域側での要求を実現させる記述としてScenarioユニットがある。このScenario, Role, Objectの3つの要素の連携によって、これまでのオブジェクトの記述に対してオープンな記述とクローズな記述を応用機能とともに分析し、区別して記述することを実現し、2節で取り上げた問題に対する解決を計っている。

S-R-Oモデルの特徴は、RoleユニットとObjectユニットとの結合メカニズムにある。Roleの記述を選ばれた核となるObjectに張り付けることでその応用領域側での要求機能に対する具体的な動作主体を作り上げる。これらの動作主体に対してその規則性や関連性を明確にし、特に実行順序を明示的に管理するものがScenarioである。ScenarioとRoleが応用領域側に依存し、Objectのみが実装領域側に依存する。このようにして応用機能と実装機能を明確に分ける。応用機能の記述に関してはScenarioにより、動作の関連状態や範囲を的確に把握できる。つまり、核とするObjectに対しての利用目的が明確化され、応用領域側の必要とする動作主体がはっきりする。また、実装機能として予め使用できるものの制限を明確に規定させることもできる。

メソッドの記述面では、これまでは応用機能を、常に汎用性を考慮した上で、オブジェクトのメソッドとしての実現が必要とされていた。この汎用性に関する前提を、RoleユニットとObjectユニットとの分離記述によってScenarioとしての限られたドメイン内での考慮に狭められる。

ここで挙げたようにS-R-Oモデルは、応用機能と実装機能を区別させることで、プログラムのテクニックとして必要とされていた各応用機能のためのad hocな問題回避作業からの開放を実現している。

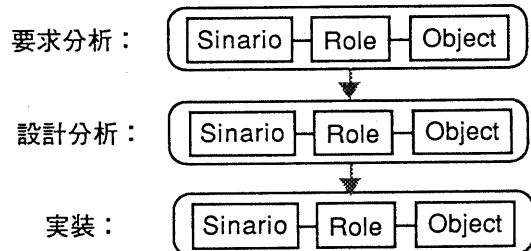
4. S-R-Oモデルとソフトウェア開発プロセス

S-R-Oモデルは、応用機能の記述をScenarioやRoleとして実装機能の記述から切り分けている。この概念は、プログラミングレベルでの特徴が得られるという位置付けだけにとどまらず、要求分析や設計分析などソフトウェア開発プロセスの上流工程においても、その利点を役立てることができる。そこで、本稿ではソフトウェア開発プロセスへのS-R-Oモデルの適用方法を分析し、開発プロセス全般にわたるモデルとしての有用性を考察する。

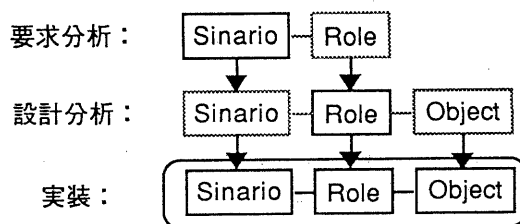
ソフトウェア開発プロセスへのS-R-Oモデルの適用の方法として、次の2種類(図2)を検討している。

(1) S-R-Oモデルを各プロセスごとに横に割り当てる(図2の(1))。

ソフトウェア開発プロセスの各プロセスにそれぞれS-R-Oの関係を実現する。これによって、各プロセスの基本機能がObjectユニットとしてまとめられ、プロセスごとの応用依存の記



(1) S-R-Oモデルを横に割り当てる



(2) S-R-Oモデルを縦に割り当てる

図2 S-R-Oモデルの適用

述がScenarioユニット／Roleユニットとして個別にまとめられる。その結果、各プロセス内では、変更への対応や部分機能の再利用を独立に実現できる。但し、この場合、このままではプロセス間の移行性に関して積極的な連結を実現することは困難であり、新たに強力な支援メカニズムが必要になる。

(2) S-R-Oモデルを各プロセスに渡って縦に割り当てる(図2の(2))。

ソフトウェア開発プロセスの全体を通して一つのS-R-Oの関係を割り当てる。開発プロセスの上流工程である要求分析は、Scenarioユニットを中心として応用機能の理解に基づく実行手順(特に外部仕様)とそれぞれの実行に必要な動作主体の機能仕様の抽出に当てられる。設計プロセスでは、動作主体としての機能仕様から、Roleユニットとしての仕様とObjectユニットとしての仕様の分析が行われる。実装段階では、応用機能としてのScenarioユニットおよびそれに依存した機能を記述するRoleユニットの実現が進められ、同時に既にあるもの以外に必要なObjectユニットの整備を行う。

この適用の場合には、プロセス間の移行情報がそれぞれ必要な仕様として分析されるため、プロセス間の積極的な連結を実現できる。

本稿では、変更への対処性に注目していることからプロセス間の関連に強力な後者の適用を

取り上げて以下で述べる。

ただし、これまでのウォーターフォールの各プロセスとしての厳密な切り分けを主に考えていない。

5. S-R-Oモデルと分析/設計

5-1 S-R-Oモデルと分析

Scenarioユニットは、応用領域側の言葉で実際にやりたいことを機能的な手順として記述する。従って、Scenario自体は抽象度の高いものから低いものまでさまざまなレベルが考えられる。対象とする応用機能としての要求を集めたものが抽象度の高いレベルのScenarioの記述である。このScenarioの精密化を進めることで、そのScenarioに現われてくる登場物の抽出を行う。

初期の抽象Scenarioから、Scenarioの洗練/精密化と登場物の集約を行っていくことがS-R-Oモデルにおける分析過程である。この分析の終了時点では、洗練されたScenarioと登場物が抽出され、各登場物に対する機能要求がScenarioから導き出され、登場物の仕様としてまとめられる(図3)。この登場物に対する機能要求が、そのScenario実現のために必要な動作単位の仕様記述に相当する。

精密化されたScenarioと登場物、および登場物の仕様が次の設計段階に移行される情報となる。

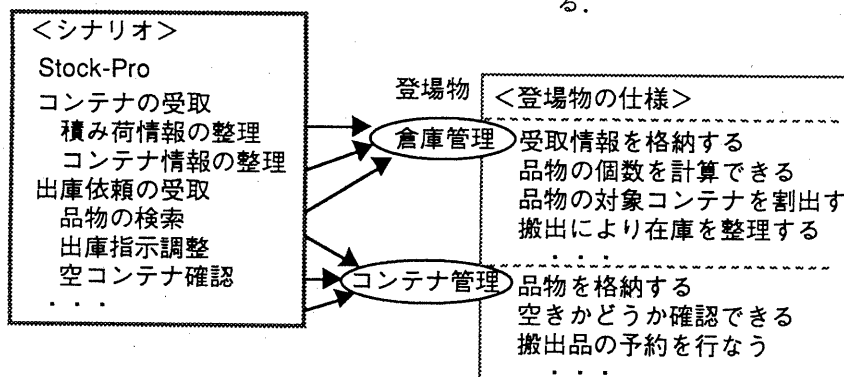


図3 シナリオと登場物および登場物の仕様

5-2 S-R-Oモデルと設計

前分析過程の動作単位に対して、Roleユニットで割り振る機能と基本的なObjectとの振り分けを実現し、応用領域側の範囲を確定した上での

機能設計を進めることがS-R-Oモデルでの設計となる。つまり、前過程の分析結果として得られる登場物の仕様を、RoleとObjectでどのように構成するかを決めることである。

Roleユニットのポイントは、対応しているScenarioユニットがスコープを決めていることであり、そのScenarioからのメッセージだけで利用される(シナリオ依存)。Objectユニットのポイントは各ScenarioやRoleと無関係に、どんな場合においてもメッセージで自由に使われるということ(共有ライブラリ)である。従って、同じObjectが異なるScenarioにおいてRoleユニットの定義が多重に行われて利用されることになる。

もう一つの重要なポイントとして、Roleユニット内で規定された機能は、Scenarioユニットを通じてObjectユニットを通じても継承されることはない。Roleで規定される機能は、対応しているScenarioでのみ動作する。それに対して、Objectはどこからのメッセージでも常に持っている機能を一意に実行するものである。

相対的に見ると、应用機能に依存して本来関数として表わしたかった機能的表現はすべてRoleユニットで規定し、应用機能に依存しない型などの情報がObjectユニットとして規定される。

6. S-R-Oモデルの開発プロセス

S-R-Oモデルでは、必要なプロセスを明らかにしたシナリオの認識と登場物の識別、プロセスと登場物の関係を明らかにして、应用領域の理解から最終的な実装までの一連の流れを確立する。

ここでは、このS-R-Oモデルの定式化に向けて、分析から設計／実装の基準を定めるために全体を5段階に分ける。S-R-Oモデルの開発プロセスでは次の5つの段階を経て最終的な実装を得ることができる。

.....

(0) 個々のシナリオの対象とする应用領域側を設定する。 [シナリオの分析]

(1) 必要とする機能とその関連に注目して应用領域側を掘り下げる。 [シナリオの詳細化]

(2) 機能をまとめながら登場物の候補を挙げる。 [登場物の仕様作成]

(3) 登場物を分析しそれらの定義と核にするオブジェクトを定める。 [ロールの設計]

(4) 動作可能なシステムとして全要素(シナリオ、ロール、オブジェクト)を実現する。 [実装]

.....

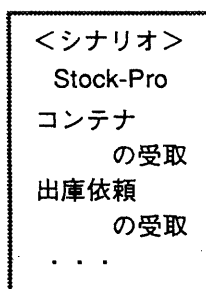
実際には、ソフトウェア開発プロセス全体で用いるモデルを統一しても、分析と設計のギャップを埋めることは容易ではない。ここでは、変更への対処を迅速に行わせるために、S-R-Oモデルの利用に対して分析段階と設計段階を完全に分離していない。そのため、純粋な分析ではなく設計を意識した分析といえる。

S-R-Oモデルの役割は、应用機能に対する実装側からの対応と実装機能に対する应用側からの対応を個別に扱えるようにし、それぞれの変更や改良をスムーズに行わせることである。これが再利用技術や標準化技術などに結び付く基本的かつ重要な役割と考えている。

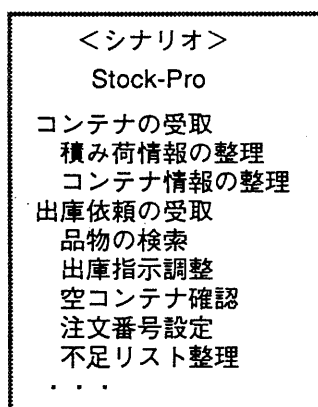
(0) 段階は、应用領域側の理解に基づく分析から生み出されるものであり、(1)と(2) 段階が設計を意識した分析の中核になる。従って、大きく(0)~(2) 段階が分析フェーズ、(3) 段階を設計フェーズ、(4) 段階をプログラミング(実装)フェーズとして分類できる。重要な点は、これらの各段階におけるフィードバックや初期の要求に関する変更への対処が、その内容に合わせて個別に処理できることである。

7. 事例

ここでは、S-R-Oモデルによる分析／設計／実装がどのように行われるかその概要を示す。事例として文献の共通問題の設計(在庫管理システムの記述)[6]を取り上げる。この問題は、



(a) 初期シナリオ



(b) シナリオの詳細と登場物

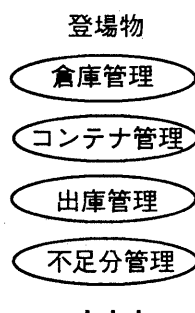


図4 在庫管理システムの分析過程

倉庫に搬入される酒瓶のコンテナに対して出庫の注文を受けて出庫指示や不足の処理を行うものである。

Step(0) シナリオの分析

この問題では、倉庫係りや受付係りのシナリオに分けたり、在庫管理システムのシナリオとして全体をまとめるなど、シナリオの対象をいろいろなレベルで捕えられる。ここでは、概要説明として後者での場合を行う。シナリオの対象を決めたことで、初期の抽象シナリオが必要な動作を主にした形で応用側の言葉で得られる(図4-a)。このステップでは、何が応用領域であるかをはっきりさせることが重要である。

Step(1) シナリオの詳細化

ここでは、前のステップで得られた抽象シナリオに対して、いくつかの主動作を核にしてその手順等の具体化を進める。同時に、シナリオに書かれた動作を見通して、必要な登場物を列挙する(図4-b)。ここで挙げられる登場物はロール仕様やロール設計の段階で精密化されてはじめてRoleユニットとなる。

Step(2) 登場物の仕様作成

シナリオ実現のために必要な機能を登場物に対応させ、動作単位としてまとめる。ここで、登場物としての動作内容を考慮した上で、ロー

ルとして必要な登場物の精密化(統合や分割を含む)を行う。この段階で図3の情報が得られる。

Step(3) ロールの設計

登場物の仕様としてまとめられたものを、どのようなオブジェクト(実装機能)を核として実現し、そのオブ

ジェクトに新たに割り当てる応用機能の分類と仕様化を行う。

Step(4) 実装

シナリオとロールをScenarioユニットとRoleユニットとして実現し、必要なObjectユニットを用意する。

各Stepでの精密化や仕様作成に関するフィードバックに対しては、ユニットごとの対応がとれる。また、応用領域における要求の変更は、実装機能自体の変更と応用機能としての変更を区別してそれぞれ個別に対処できる。さらに、応用機能の変更に関しても、処理機能の手順の変更や個々の動作単位が持つ応用機能の変更などを区別して的確に対処できる。

8. 事例に基づく考察

変更やフィードバックへの対処方法の違いに着目して、これまでのオブジェクト指向モデルとの比較をまとめる。

S-R-Oモデルによる分析から設計/実装では、各Stepで変更や改良要求が起こった場合でも迅速な対応ができることを目的としている。その本質は、必要とする振る舞いを応用機能と実装機能とに分け、RoleとObjectとして明確に切り分けることである。これを実現したS-R-O

表1 変更により変わる内容の違い

モデルの要素	変更によって変わる内容 (変えたい対象)
シナリオ	対象とするアプリケーション 応用領域、目的
ロール	シナリオで決められた 枠組みに対する内部記述
オブジェクト	基本要素機能、型

モデルは、結果的に本稿で述べたようにプログラミングのモデルとしてだけでなく、応用領域の分析段階から採用できるモデルとしてその効果を発揮できる。

S-R-Oモデルの各ユニットの変更が対応する変化内容を表1にまとめる。この表からも分かるように、ScenarioとRoleは、応用領域側に依存した必要な機能を実現するものであり、応用機能の変化要求にしたがって変更されるものである。対してObjectは、応用機能が変化してもその影響をまったく受けない。逆に、Objectだけを変更することで実装機能としての核となる機能の変化や改良に対処することができる。このような変更は、これまでのオブジェクト指向モデルでは、各オブジェクトの変更として処理を進められるが、実際には、応用機能と実装機能との区別が無いため、その変更がad hocであり、プログラマの技量に大きく依存していた。特に、実装における制限などを応用側で検討することはできない。その点S-R-Oモデルでは、実装面からのフィードバックをRoleでの動作単位としての対処あるいはScenarioでの対処など段階に応じたフィードバックを行える。

また、Scenarioの区別は、応用領域の違いにとどまらず、大規模なシステムにおけるサブ実現機能として捕えることも、開発担当範囲の区別として捕えることもできる。このような場合には、Objectは、予め互いに共通化しなければならない基本機能や与えられる環境(制限)とし

て設定することができる。

さらにこのモデルは、再利用を目指すものとしての位置付けもできる。しかし、プログラムやそこに至るまでの過程の再利用は、プログラマやアナリストの考え方や意欲に大きく依存する。したがって、それらの検証は容易なことではない。そこで、ここでは、このモデルのポイントを変化への対応性という観点で捕え、再利用コストは変更コストと対応させてその効果を追及している。変更のためのコスト削減は、現実の開発においても再利用という枠組みに直接結び付く重要なポイントである。

このモデルを採用する効果は、他にもさまざまな観点から評価することができる。たとえば、Scenarioによってその応用領域や場面が規定されるため、結果的にこれまでのようなObjectに対する名前付けのスコープが限定される。これは、協調的プログラム開発において必要以上の分析を行わないためにも重要である。また、Objectとの対応関係を明確にすることは、その応用領域に対して利用しているObjectをはっきりと限定することである。つまり、使用している基本部品を明確に規定できることになる。これは、利用性や理解性などとともに影響範囲や応用領域に対するトップダウン的のスコープにボトムアップ的のスコープを加えることになる。

参考文献

- [1] P.Coad & E.Yourdon : "Object-Oriented Analysis second edition" Yourdon Press, 1991
- [2] 片山, 小林 : "ソフトウェアとしての利用性を指向した新しいオブジェクトモデル(S-R-Oモデル)" Proc. Symposium of Information Science 1993, Jan 1993
- [3] K. S. Rubin & A. Goldberg : "Object Behavior Analysis" Communications of the ACM, Vol.35, No.9, 1992
- [4] I. Jacobson : "Object-Oriented Software Engineering" Addison-Wesley, Reading, Mass., 1992
- [5] G. Booch : "Object Oriented Design with Applications" The Benjamin/Cummings Publishing Company, Inc. 1991
- [6] 二村, 雨宮, 山崎, 淵 : "新しいプログラミングパラダイムによる共通問題の設計" 情報処理 Vol.26 No.5, May 1985