ソフトウェア工学 96－6
（1994．1．20）

型に基づく複合オブジェクトの振舞いの解析

宮本 衛市† 何 克清‡

†北海道大学工学部情報工学科 ‡武漢大学ソフトウェア工学国家重点研究実験室

ソフトウェア開発にとって、その上流工程である要求仕様の策定および基本設計などはきわめて重要であるとともに、最大の難所となっており、これに対してプロトタイピングが有力な手法の１つと考えられている。われわれは、オブジェクト指向に基づくプロトタイピングのオペレーショナルモデルを提案してきた。このモデルは状態遷移を基本的な枠組とし、状態の詳細化と機能の詳細化を推し進めていくものである。しかし、機能の詳細化に伴い並行オブジェクトが発生し、それらはメッセージを介して結合しているのであるが、個々のオブジェクトの振舞いと全体の振舞いの整合が甚だわかりにくいものとなる問題がある。本稿では、各オブジェクトの振舞いを正規表現を基本としたパス式で表し、これをオブジェクトの振舞いの観点から見た型と考え、この型を介してオブジェクト間の振舞いの整合を解析することを述べる。

An Analysis of Behavior of Composite Objects Based on Type

Eiichi Miyamoto† Keqing He‡

†Division of Information Engineering, Faculty of Engineering, Hokkaido University
Kita 13 Nishi 8, Kita-Ku, Sapporo 060, Japan
‡State Key Laboratory of Computer Software Engineering, Wuhan University
Wuhan 430072, People's Republic of China

For software development, it is very important and is considered to be the most difficult problem to meditate the software specification and its fundamental design. Prototyping technique has proved to be quite powerful for the preliminary phases of software development. We have proposed an operational model for prototyping based on the object oriented model. This model is basically a state transition model, and intends to analyse a software system in both refinement of states and functions. But, as the software becomes large, it is very difficult to recognize the consistency between the behavior of separate objects and of the total system. In this report, we propose a type expressed in regular form for objects viewed from their behavior, and the analysis of the behavioral consistency among objects.

1 はじめに

大規模で、複雑なソフトウェアを高品質に、かつ頑健に構築するためには、ソフトウェアプロセスでも、特に仕様策定や概念設計などの上流工程でのつめが大事であり、そのためにはプロトタイピングの手法がきわめて有効と考えられている。プロトタイピングでは、ソフトウェアの仕様をその機能面から分析し、機能仕様を作成し、それをシミュレートすることで、ソフトウェアに対する潜在的ニーズを目の前で確認することができる。そこで、プロトタイピング時にソフトウェアの機能仕様を記述するためのモデルが必要であり、さらにそモデルに基づいて記述した仕様で、ソフトウェアの振舞いのシミュレーションができなければならない。

これまでにも、システムの状態遷移に基づいたプロトタイピングモデルがいろいろと提案されてきた。Harel の Statecharts モデル [1] は、基本的には有限状態機械モデルを拡張したもので、状態遷移図を用い、構造のあるシステムの状態を把握するために、並列動作の記述が可能な記述モデルを与え、状態の詳細化に伴う状態数の増加を押え、大規模システムの挙動解析に強力な手段を提供している。Coleman らは Statecharts を応用したオブジェクトの設計法を提案した [2]。そこでは、システムを構成するオブジェクトを状態遷移に基づいて設計しようとしており、各オブジェクトをオブジェクト向けに Statecharts を拡張した Objectcharts で記述する。酒井はオブジェクト指向によるシステムの概念設計のため、オブジェクトの振舞いに基づくコントラクトの概念を定義した [3]。これにより、オブジェクト固有の振舞いの表現を基礎とし、オブジェクト間での、一貫性制約や振舞い協調の記述に基づいて設計を進めていく方法を提案している。これらに対し、、われわれはオブジェクトの各状態をオブジェクト化し、プロトタイピングのオペレーショナルモデルとして十分な柔軟性と記述力のあるモデル OPM(Object-oriented Prototyping Model) を提案した [4]。これは機能オブジェクトの状態の段階的詳細化を横糸とし、各機能オブジェクトをより基本的な機能オブジェクトに分解していく過程を縦糸とし、この両者が同値関係を保ちつつ 2 次元的にソフトウェアシステムを展開していくために基本となるモデルである。

OPM では、オブジェクトの詳細化あるいは抽象化の枠組は用意されているが、並行オブジェクト間でのメッセージ交換は、各オブジェクトの動作の記述に委ねられている。しかし、並行オブジェクトの数が増えるにつれてメッセージの数も急増するとともに錯綜し、オブジェクト群全体としての振舞いが大変にわかりにくくなってくることが避けられない。これに対し、CCS[5] ではプロセスの内部には触れず、メッセージに焦点を当てて並行プロセスに対する理論的基礎を与え、原田は CCS に基づく型グラフを定義してこれをオブジェクトの型とみなし、オブジェクト間での型のマッチングを調べる方略を与えている [6]。しかし、われわれはプロトタイピングの立場から、オブジェクトを状態遷移の観点から分析しており、状態に着目したオブジェクトの振舞いを与える指標が欲しい。そこで、われわれはオブジェクトの状態遷移を基本的には正規表現で表したパス式を考え、これでオブジェクトの振舞いの観点から見て型と考えることにする。そしてこの型を通してオブジェクト間の振舞いの協調を分析し、オブジェクト全体の振舞いを見通しよいものにすることを考えている。

以下、第 2 章では、拡張したプロトタイピングモデルを定義し、このモデルで在庫管理システムの例を記述する。第 3 章では、オブジェクトの振舞いを記述するために状態遷移を正規表現で表したパス式を導入し、これに基づきオブジェクト間の振舞いの整合を判断することを述べる。第 4 章では本稿で提案したモデルの問題点を議論し、今後の研究課題を挙げる。

2 OPM によるソフトウェア記述

本章では、プロトタイピングモデル OPM の紹介を行ない、これにより共通問題である在庫管理

システムの記述を試み、並列オブジェクト群の全体としての振舞いがきわめてわかりにくいことを示す。

2.1 プロトタイピングモデル OPM

われわれは、先にオブジェクト指向モデルに基づいたオペレーショナルモデルを提案した。われわれがソフトウェア機能仕様を記述するためにオブジェクト指向モデルを採用したのは、このモデルが抽象化能力に優れているばかりでなく、階層構造などの構造化にも柔軟に対応することができるからである。それ故、オブジェクト指向モデルによれば、ソフトウェア機能仕様の抽象化、およびそれに基づく部品化にも対応することができるばかりでなく、トップダウンにソフトウェアの機能を機能要素に分割していくプロトタイピング・プロセスにも対応することができると考えたからである。

プロトタイピングでは、ソフトウェアの要求仕様を機能に基づいてシミュレートすることを目的としている。この機能を明解に分析するために、機能オブジェクトをオートマトンに見立てて状態遷移を作成し、各状態を機能的に扱うために、状態オブジェクトで表現しようとするのがわれわれの意図である。しかし、ソフトウェアが大規模になるとその機能も複雑になり、状態遷移も複雑化し、その結果ソフトウェアの機能の見通しが悪くなるおそれが出てくる。これに対し、状態の詳細化と機能の詳細化で対処する。前者は、ソフトウェアに対する見方が詳しくなるにつれて、それまでに分類されていた状態オブジェクトをさらに詳細化した状態オブジェクト群へ展開することであり、後者はソフトウェア機能をいくつかの要素機能オブジェクトへ展開して実現することを意図している。

次に機能オブジェクトの定義式を示す。これは先に提案したモデルを拡張したものである。その主たる改良点は、受け取ったメッセージに対するオブジェクトの働きとして、条件付実行の記述ができるようにしたことである。なお、式の冒頭に'an`を付けたオブジェクトは一時的オブジェクトを表すものとし、付けないものは永続的オブジェクトを表すものとする。

機能オブジェクト::=
[an] object *object_name*:
  stateset{*state_name.*}$^+$:
  in {*state_name*(*event_name*).}$^+$:
  out {*state_name*(*event_name*).}$^+$:
    {state *state_name*:
      {*receving_event_name*[(*parameters*)]
        /*object_action*:
      }$^+$
    }$^+$
  end

object_action::=
  [all *iteration_name*:]
    '{' { '['*condition*']'*object_task*:}$^+$'}'
    |*object_task*
object_task::=
  sending_events'|'*next_state_name*
sending_events::=
  ({*sending_event.*}$^+$)|*sending_event*
sending_event::=
  sending_event_name[(*parameters*)]
   < *receiving_object_name* >

ただし、イタリック体はメタ変数を、{}$^+$は1回以上、{}*は0回以上の繰返しを、[ ]は省略可能を意味するメタ記号を表し、式を簡単にするために最後に現れる区切り記号は省略できるものとする。また、{.}.[.],| を終端記号として用いるときには、引用符でくくるものとする。

2.2 共通問題の記述

ここでは、共通問題である在庫管理システム [7] を取り上げ、これを OPM で記述してみる。まず、次は在庫管理システム全体としての機能仕様を記述したもので、簡単のためサービス開始のイベントで、常時サービス中の状態にいるものとしている。

object 在庫管理システム:

```
  stateset サービス中;
  in サービス中 (サービス開始);
    state サービス中:
    コンテナ搬入 (コンテナ)/
     all 納入銘柄:
      {[不足数量入荷]
          遅れ発送通知<顧客> | サービス中;
       [不足数量未入荷]
          未入荷通知<顧客> | サービス中;}
    出庫依頼 (銘柄, 数量)/
      {[依頼数量在庫あり]
          即時発送通知<顧客> | サービス中;
       [依頼数量在庫なし]
          未在庫通知<顧客> | サービス中;}
end
```

次に、この在庫管理システムを受付、倉庫、不足品リストおよびコンテナの4つの機能オブジェクトに詳細化したときの、各オブジェクトの記述を次に示す[8]。ただし、受付、倉庫および不足品リストは永続的オブジェクトとし、コンテナは一時的オブジェクトとして扱っている。

```
object 受付:
  stateset 受付中, 入荷調査, 入荷銘柄調査,
           銘柄在庫調査, 銘柄数量調査;
  in 受付中 (サービス開始);
  state 受付中:
    コンテナ搬入 (コンテナ)/
      (格納指示 (コンテナ)<倉庫>,
       入荷調査開始<不足品リスト>,
       コンテナの指定<コンテナ>)| 受付中;
    不足品リストあり/
      次の不足品<不足品リスト> | 入荷調査;
    出庫依頼 (銘柄指示, 数量)/
      在庫調査開始<倉庫> | 受付中;
    コンテナリストあり/
      次のコンテナ<倉庫> | 銘柄在庫調査;
    搬出要求 (コンテナ)/
      搬出指示 (コンテナ)<倉庫>;
  state 入荷調査:
    不足品あり (銘柄)/
      在庫ありや<コンテナ> | 入荷銘柄調査;
    不足品なし/
      入荷調査終了<コンテナ> | 受付中;
    搬出要求 (コンテナ)/
      搬出指示 (コンテナ)<倉庫>;
  state 入荷銘柄調査:
    入荷あり (銘柄, 在庫数)/
      {[不足数を越える在庫数がある]
        (不足数量を予約
          (銘柄, 不足数)<コンテナ>,
          予約品を出庫 (銘柄)<倉庫>,
          不足品リストから削除し次の不足品
          (銘柄)<不足品リスト>
        | 入荷調査;
       [不足数を下まわる在庫数がある]
        (在庫数を予約し在庫リストから削除
         (銘柄)<コンテナ>, 次の不足品
         <不足品リスト>)| 入荷調査;
       [不足数と在庫数が等しい]
        (在庫数を予約し在庫リストから削除
          (銘柄)<コンテナ>,
          予約品を出庫 (銘柄)<倉庫>,
          不足品リストから削除し次の不足品
          (銘柄)<不足品リスト>)
        | 入荷調査;}
    入荷なし/
      次の不足品<不足品リスト>
        | 入荷調査;
  state 銘柄在庫調査:
    コンテナあり (コンテナ)/
      在庫ありや (銘柄)<コンテナ>
        | 銘柄数量調査;
    コンテナなし/
      不足品追加 (銘柄)<不足品リスト>
        | 受付中;
    搬出要求 (コンテナ)/
      搬出指示 (コンテナ)<倉庫>;
  state 銘柄数量調査:
    在庫あり (銘柄)/
      {[依頼数を越える在庫数がある]
        (依頼数を予約 (依頼数)<コンテナ>,
```

```
        在庫調査終了し予約品を出庫
        (銘柄)<倉庫>)| 受付中;
      [依頼数を下まわる在庫数がある]
        (在庫数を予約し在庫リストから削除
        (銘柄)<コンテナ>, 次のコンテナ
        <倉庫>)| 銘柄在庫調査;
      [依頼数と在庫数が等しい]
        (在庫数を予約し在庫リストから削除
        (銘柄)<コンテナ>,
        在庫調査終了し予約品を出庫
        (銘柄)<倉庫>)| 受付中;}
    在庫なし/
      次のコンテナ<倉庫> | 銘柄在庫調査;
end

object 不足品リスト;
  stateset 不足品受付中, 不足品調査中;
  in 不足品受付中 (サービス開始);
  state 不足品受付中:
    入荷調査開始/
      {[リストが空でない] 不足品リストあり
        <受付> | 不足品調査中;}
    不足品追加 (銘柄)/
      {[不足品リストに銘柄追加]
        | 不足品受付中;}
  state 不足品調査中:
    次の不足品/
      {[次の不足品なし]
        不足品なし<受付> | 不足品受付中,
       [次の不足品あり]
        不足品あり<受付> | 不足品調査中;
    不足品リストから削除し次の不足品 (銘柄)/
      {[銘柄を削除後次の不足品なし]
        不足品なし<受付> | 不足品受付中;
       [銘柄を削除後次の不足品あり]
        リストから先頭を削除]
        不足品あり<受付中> | 不足品調査中;}
end

object 倉庫;
  stateset 格納受付中, 在庫調査中;
  in 格納受付中 (サービス開始);
  state 格納受付中:
    格納指示 (コンテナ)/
      {[コンテナリストに追加]| 格納受付中;}
    搬出指示 (コンテナ)/
      {[コンテナリストから削除]| 格納受付中;}
    在庫調査開始/
      {[リストが空でない] コンテナリストあり
        <受付>| 在庫調査中;}
    予約棚への移管 (銘柄, 数量)/
      {[予約リストに追加]| 格納受付中;}
  state 在庫調査中:
    次のコンテナ/
      {[次のコンテナなし]
        コンテナなし<受付中> | 格納受付中;
       [次のコンテナあり]
        (コンテナあり<受付>,
         コンテナの指定<コンテナ>)
          | 在庫調査中;}
    在庫調査終了し予約品を出庫 (銘柄)/
      発送通知 (銘柄)<顧客>| 格納受付中;
    搬出指示 (コンテナ)/
      {[コンテナリストから削除]| 在庫調査中;}
    予約棚への移管 (銘柄, 数量)/
      {[予約リストに追加]| 在庫調査中;}
end

an object コンテナ;
  stateset 探索要求受付中, 指定銘柄在庫中,
           消滅;
  in 探索要求受付中 (コンテナの指定);
  out 消滅;
  state 探索要求受付中:
    在庫ありや (銘柄)/
      {[指定銘柄在庫あり] 在庫あり
        (銘柄)<受付> | 指定銘柄在庫中;
       [指定銘柄在庫なし] 在庫なし
        <受付中>| 探索要求受付中;}
  state 指定銘柄在庫中:
    要求数量予約 (銘柄数量)/
      予約棚へ移管 (銘柄, 数量)
```

```
        | 探索要求受付中:
      在庫数を予約し在庫リストから削除(銘柄)/
        {[銘柄削除後リストが空]
          (予約棚の移管(銘柄,在庫数)<倉庫>.
           搬出要求<受付>)| 削除;
          [銘柄削除] 予約棚の移管
          (銘柄在庫数)<倉庫>
          | 探索要求受付中;}
end
```

3 パス式による型の導入

本章では、オブジェクトの状態遷移をパス式で与え、これをオブジェクトの振舞いの枠を規定しているものとして、オブジェクトの型と考えることができることを述べ、オブジェクト間での型の整合の解析方法について述べる。

3.1 パス式によるオブジェクトの振舞いの表現

前章で在庫管理システムの例を示したが、これではオブジェクトの個々のメッセージに対する働きはわかっても、このようなメッセージをやりとりしてオブジェクトが意図している振舞いを理解することは難しく、ましてやオブジェクト群としての意図した振舞いを把握することはいっそう困難である。そこで、オブジェクトの振舞いを直観的に理解するために、通常、状態遷移図を用いるが、ここではそれに対応して正規表現で表すと次のようになる。

$path$(在庫管理システム) = サービス中*

$path$(受付) = (受付中・((入荷調査
・入荷銘柄調査)* + (銘柄在庫調査
・銘柄数量調査・(+受付中))*))*

$path$(倉庫) = 格納受付中*
+(格納受付中・在庫調査中*)*

$path$(不足品リスト) = 不足品受付中*
+(不足品受付中・不足品調査中*)*

$path$(an コンテナ) = (探索要求受付中
・指定銘柄在庫中・(+消滅))*

ここで、・はある状態からある状態への一義的な遷移を表し、+は受け取ったメッセージあるいは内部条件により、いくつかの状態遷移があることを表し、*は一つの状態あるいはいくつかの状態間にまたがって繰り返しがあることを表している。ただし、状態遷移図は単なる状態遷移のみを表しているので、パス式はいわば状態遷移図を、逐次、分岐および繰返しの表現要素で、構造的に記述し直したものということができる。なお上式で、受付の式中の(+受付中)やコンテナの式中の(+消滅)は、意味的には正規表現の範囲を逸脱している。これらは繰り返しからの脱出を意図したものであるが、便宜上、相手のない分岐として正規表現風に記述している。

3.2 型としてのパス式の意味

オブジェクトの状態の設定には任意性があり、オブジェクトの状態遷移、したがってパス式も一義的には決まらず、オブジェクトの振舞いを表すものとしてはオブジェクトに固有なものと考えることはできない。しかし、オブジェクトの状態遷移をパス式で表すことにより、オブジェクトの振舞いの構造を表明することが可能となる。このような性質を持つパス式に対し、われわれがオブジェクトの振舞いの型に対応させようという意図は、つぎのような視点からである。すなわち、オブジェクト間でメッセージの対応が取れて、協調した働きができるということは、そこにお互いの状態遷移に構造的な対応が取れているからであり、それはパス式でも対応が取れるはずであると考える。言い換えると、パス式上で対応できるような状態遷移をオブジェクトの振舞いとして設定できるとき、オブジェクト間での振舞いの対応が取れているものと考えようということである。そこで、パス式上での対応を判定する条件を以下に定義する。

1. まず、パス式で与えられる繰返し構造に着目し、パス式上で状態が置かれている位置での繰返しのネストの深さに対応した値を付与する。永続的なオブジェクトに対しては、この値を絶対的なものとし、一時的なオブジェクトに対しては、相対的な値とし、メッセージ授受が行なわれる位置でオフセットが加えられるものとする。

2. 各オブジェクトが送り出すメッセージごとに、メッセージを送り出す前の状態のパス値を p、送り出した後のパス値を q とし、それを受け取るオブジェクトの状態のパス値を u、そのオブジェクトが遷移する状態のパス値を v_i とする $(i=1,\ldots,n)$。ここで、メッセージを受け取ったオブジェクトのメッセージ発行後のパス値は、条件によりいくつかの働きに分岐して、異なるパス値の状態に遷移する場合があることに対応している。そこで、あるオブジェクト x が $(p \rightarrow q)$ の状態遷移に対応した働きにより、他のオブジェクト y に $(u \rightarrow v_1,\ldots,v_i)$ に対応する状態遷移を引き起こすことを、次のように表す。ただし、オブジェクト y は $v_1,\ldots,v_n$ のいずれかに遷移するものとする。

   $(p \rightarrow q) \rightarrow (u \rightarrow v_1,\ldots,v_i)$

3. オブジェクト x と y が永続的オブジェクトの場合には、$p=u$ の関係がなければならない。これはメッセージを受け取ったオブジェクトと、そのオブジェクトが次にメッセージを送るオブジェクトが構造的に同じネスト状態であることを要求している。ただし、オブジェクト x とは構造的な一致を取る必要のない、手続き呼出し的なオブジェクトは除く。この場合のオブジェクトは完結型の仕事の繰返しなので、$(1 \rightarrow 1)$ のネスト値が与えられているものとする。以下、このような場合は検討から除外する。

4. $p=q$ のとき、$u=v_i$ の場合と、$u \neq v_i$ の場合がある $(i=1,\ldots,n)$。前者はオブジェクト x が構造を変えないことにならって、オブジェクト y も構造を変えないことに対応し、後者はオブジェクト y に構造を変えるイニシアティブを取る場合があることを意味している。

5. $p \neq q$ のとき、$q=v_i$ であること $(i=1,\ldots,n)$。これはメッセージの発送元が構造を変えたので、メッセージを受け取ったオブジェクトも同様な変化をしなければならないことを意味している。

6. 一時的なオブジェクトは、永続的なオブジェクトの構造的なある断面に繰り込まれるものと考える。したがって、一時的なオブジェクトがメッセージの送り手あるいは受け手にいる場合には、3の条件からネスト値のオフセットを求め、それを加えて4あるいは5の判定を行なう。

7. 各オブジェクトに対し分岐ごとにサブパスを考え、あるオブジェクトのあるサブパスからのメッセージは、相手のオブジェクトの特定のサブパスにのみ送られていなければならない。

8. 以上の判定をすべてのオブジェクトのすべての状態のすべての場合のメッセージ送信に対して行ない、すべて満足した場合にパス式の意味でオブジェクト間に対応が取れていると判断する。

3.3 在庫管理システムのパス式に基づく振舞いの分析

3.1 で与えた在庫管理システムのパス式に基づき、機能オブジェクトのいくつかの状態遷移に伴うパス値の変化を調べてみる。

例えば、「受付」オブジェクトの「受付中」状態で、「入荷調査開始」メッセージを「不足品リスト」オブジェクトに送り、自身は「入荷調査」状態に遷移している。一方、このメッセージを受け取った「不足品リスト」オブジェクトは「不足品受付中」状態から「不足品調査中」状態に遷移している。この間のパス値の変化は

$(1 \rightarrow 2) \rightarrow (1 \rightarrow 2)$

となり、「受付」オブジェクトが「不足品リスト」オブジェクトを新しい繰り返しに誘い込んでいることを示している。

また、「受付」オブジェクトの「銘柄数量調査」状態で、「依頼数を下まわる在庫数がある」条件のとき、「在庫数を予約し在庫リストから削除」メッセージを「コンテナ」オブジェクトに送り、自身は同じ状態に留まっている。これを「コンテナ」オブジェクトは「指定銘柄在庫中」状態で受け、その銘柄を削除し予約棚への移管を「倉庫」オブジェクトに伝えたのち、銘柄リストが空のときには「搬出要求」を「受付」オブジェクトに送って自身は消滅し、空でないときには「探索要求受付中」状態に遷移している。この間のパス値の変化は

$(2 \rightarrow 2) \rightarrow (+1 \rightarrow +1, +0)$

となる。ここで、「コンテナ」は一時的なオブジェクトであるので、そのパス値に'+'をつけて永続的オブジェクトの場合と区別している。そこで、「コンテナ」の遷移は「受付」に合わせて、$(2 \rightarrow 2, 1)$とみなす。なお、「受付」オブジェクトは引き続いて「倉庫」オブジェクトに「次のコンテナ」メッセージを送っているが、これは先の「予約棚への移管」メッセージが送られた後に送られる必要があり、OPMではメッセージの順序制御が行なわれることを前提としている。

4 おわりに

本稿では、先に提案したプロトタイピングモデルを拡張したモデルを提案し、それにより共通問題とされている「在庫管理システム」の記述を行なってみた。その結果、この程度の問題であっても、各機能オブジェクト間の関係は錯綜し、オブジェクト群全体としての振舞いの見通しが大変に困難になることを実感した。そのためオブジェクトの振舞いを直観的に表すために使われる状態遷移図に代わって、状態遷移をパス式で表すことを提案した。これにより、オブジェクトの状態遷移を構造的に把握することができ、さらにこのパス式でオブジェクトの振舞いの型を定義し、オブジェクト間での型の整合条件を述べた。

しかし、拡張したモデルでも、オブジェクトの属性は導入せず、これらはすべてオブジェクトの状態として分類されるものと考えている。そのため、受信メッセージに対するオブジェクトの働きが条件付であっても、モデル自身で判定することはできず、外部の判断に委ねている。そこで、ソフトウェアの上流工程でのオブジェクトの属性記述やそれに対する計算手段のあり方は今後の課題と考えている。

参考文献

[1] Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*. Vol.8. pp.231-274 (1987).

[2] Coleman, D., Hayes, F. and Bear, S.: Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design, *IEEE Trans. Software Eng.*, Vol.SE-18. No.1. pp.9-18 (1992).

[3] 酒井博敬：オブジェクトの振舞いに関するコントラクトの設計について. 情報処理学会論文誌. Vol.33. No.8. pp.1052-1063 (1992).

[4] 宮本、何：プロトタイピングのためのオペレーショナルモデルＯＰＭ. 情報処理学会 ソフトウェア工学研究会. 93-1. pp.1-8 (1993).

[5] Milner, R.: Calculus of Communication System. *Springer-Verlag* (1980).

[6] 原田：部品の接続を考慮した通信仕様の分解、日本ソフトウェア科学会第 10 回大会論文集、pp.365-368 (1993).

[7] 二村良彦他：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. pp.458-459 (1985).

[8] 柴山悦哉他：並列オブジェクト指向言語 ABCL による在庫管理システムの記述. 情報処理. Vol.26. No.5. pp.460-468 (1985).