ソフトウェア工学 103−18
（1995．3．10）

オブジェクト指向方法論 OMT における動的モデルと機能モデルの整合性の検証

佐々木 健† 片山 卓也††

東京工業大学 情報工学科†
北陸先端科学技術大学院大学 情報科学研究科‡

オブジェクト指向設計手法 OMT では3つのモデルによって仕様を表現する。しかしモデル間の関係が明確ではなく，記述の正当性を検証することは困難である。3つのモデルのうちで，動的モデルと機能モデルはどちらも動作に関する記述である。動的モデルはオブジェクトの状態を表し，状態遷移によって属性値が決まる。一方，機能モデルは属性間の依存関係を与える。したがって，2つのモデル間の記述の一貫性を属性間の関連を通して考えることができる。本論文では OMT の動的モデルと機能モデルに対して記号による記法を与え，2つのモデル間の関連について考察を行なう。これにより記述された2つのモデルの一貫性を検証するための基礎を与える。

Verifying consistency between dynamic model and functional model in object-oriented modeling technique OMT

Takeshi Sasaki† Takuya Katayama††

Department of computer science, Tokyo institute of technology†
School of information science, Japan advanced institute of science and technology ‡

An object-oriented modeling technique OMT describes specifications by using three different models. A drawback of OMT methodology is that relation among these models are too ambiguous to verify their specifications. Both the dynamic model and the functional model describe dynamic aspects of a system. The dynamic model contains state diagrams of object and how to calculate attribute values with state transition. The functional model contains dependency diagrams between attributes. Our idea is that we can check consistency among their attributes' information.

In this paper, we provide symbolic notation for the dynamic model and the functional model, and relation about these models. We propose a foundation of checking consistency between the dynamic model and the functional model.

1 はじめに

近年, オブジェクト指向方法論は問題のモデル化のしやすさで注目を集めている。そのなかでも特に広く用いられているものとして OMT[6] がある。OMT では 3 つのモデルを図によって記述する。これは視覚的な理解の容易さでは優れている。一方, OMT では 3 つの図の間の関係は明確には示されてはおらず, 記述間の一貫性を検証することは困難である。

分析段階での誤りが設計段階・実装段階で現れた場合, これが実際にはどの部分の誤りであるかを判断することは難しい。記述間の一貫性を保証することは分析段階での誤りを検出することで設計・実装段階での誤りを減少させるのに有効である。

OMT のモデルのうち, オブジェクトモデルはシステムの静的な側面を記述する。一方, 動的モデルおよび機能モデルはどちらも動作に関する側面を記述する。動的モデルはオブジェクトの属性を状態という形で抽象化し, イベントによる属性値の変化を状態遷移で表現していると考えることができる。また, 機能モデルはデータフローによってシステムの大域的な属性の依存関係を表現していると考えることができる。そこで, 明示的に現れていない動的モデルの属性の依存関係を表現することができれば 2 つのモデル間の属性の依存関係の上での一貫性を検証することができると考えられる。

本研究では OMT のモデルのうち, 動的モデルと機能モデル間の一貫性を検証することを目的とする。ここでは動的モデルに対して属性に関する記述を付加し, 2 つのモデルを属性の依存関係に着目して一貫性を検証する手法を提案する。

2 方法論の概要

ここでは 2 つのモデルはそれぞれ別に記述されると仮定する。このとき動的モデルと機能モデルでは異なる名前の属性が同じものを表す場合も考えることができる。したがって属性名は 2 つのモデル間で一致させておく必要がある。

検証のための手順は図 1のように与える。まず動的モデルでは属性は明確な形では記述されていない。そこでイベントや状態遷移に対する操作の形で属性に関する記述を動的モデルに付加する。

属性に関する記述を付加した動的モデルと機能モデルを記号による表記に変換する。最後に記号表現に変換したモデルから属性の依存関係を抽出し, この依存関係の上で一貫性を検証する。以下

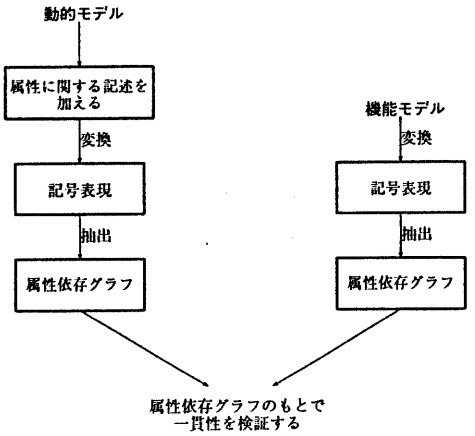

図 1: 方法の概要

では動的モデルに対して属性に関する記述が与えられていると仮定して話を進める。

3 モデルの表現

3.1 動的モデルの記号表現

3.1.1 準備

OMT の動的モデルは Statechart[3] で記述されている。記号表現を導入するに当たって以下の制約を与える。

- 単一階層の状態図
  Statechart では状態を階層的に記述することができる。階層的な状態図を考えるときには, 状態の内部に状態図を考える必要がある。このとき, 例えば一度階層化された状態から別の状態に遷移し, 再びその状態に遷移して来たときに内部の状態図がどのようになっているかを考える必要がある。今回は問題を簡単にするために単一階層の状態図で考えることにする。

- 動作に対する制約
  状態図で扱う操作についても制約を加える。明示的に記述するアクションはイベントの出力に限定する。その他のアクションは属性に対する操作として記述するものとする。

- 属性に関する記述の付加
  記号表現を行なうモデルでは動的モデルに属性に関する記述を付加したものとする。ここで属性に関する記述とは，イベントに付加した属性の記述，遷移の条件の属性の記述，遷移の際に行なう属性の操作である。

つぎに記号表現に対する形式化を与える際に用いる記号を以下で与える。

- モデルで現れる属性の集合 $\mathcal{A}$
- モデル全体のイベントの集合 $\mathcal{E}$
  $\mathcal{E}$のある要素には $\mathcal{A}$ で与えられる属性が付加されている。
- 空イベント $\epsilon \in \mathcal{E}$
  イベントの入力 (あるいは出力) がないことを表す。
- 動的モデル内の属性の操作の名前の集合 $\mathcal{F}$
- $\mathcal{A}$ と $\mathcal{F}$から構成される式の集合 $Exp(\mathcal{A},\mathcal{F})$
  以下のように定義する。
  - $a \in \mathcal{A}$ のとき $a \in Exp(\mathcal{A},\mathcal{F})$
  - n 個の引数を持つ操作の記号 $f \in \mathcal{F}$と属性 $a_1,\cdots,a_n \in \mathcal{A}$ について，$f(a_1,\cdots,a_n) \in Exp(\mathcal{A},\mathcal{F})$

3.1.2 状態図の形式化

はじめに 1 つの状態図からなるモデルについて考える。OMT の状態遷移図は基本的にはイベントの入力によって状態の遷移が起こる。したがって CCS[5] のようなプロセス代数に近い表記方を導入できる。しかしここでは状態遷移にともなって属性に対する操作が起こるようなモデルを考えている。プロセス代数は本来外部の振舞いのみを記述する体系であり，このような記述を考えているわけではない。そのため属性の操作も含めて記述するための表現を考える必要がある。

状態図は以下の組の形で表現する。

$$(S, E_I, E_O, C, T, A, s_i, s_t)$$

組の要素はそれぞれ以下の意味を持つ。

- S
  状態図に含まれる状態の集合
- $E_I \in \mathcal{E}$
  入力イベントの集合
- $E_O \in \mathcal{E}$
  出力イベントの集合
- $C \subset Exp(\mathcal{A},\mathcal{F})$
  遷移の際に参照される条件の集合
- T
  状態遷移の式の集合
  状態遷移はプロセス代数に類似した記法を用いる。たとえば図 2のような状態遷移図があるとき，

$$s_1 = e_i[c].\overline{e_o}.s_2$$

と記述する。s_1が複数の遷移を持つ場合の記

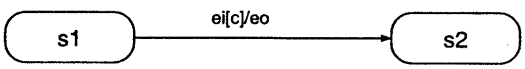

図 2: 状態遷移の例

法として

$$s_1 = e_i[c].\overline{e_o}.s_2 + \cdots$$

を与える。

- A
  状態遷移にともなう属性の操作の集合
  A の要素は (t, e_a) の形で表現される。$t \in T$は状態遷移であり，aは状態遷移にともなう属性操作の式である。e_aは $a = e$の形の式として表現する。ここで$a \in \mathcal{A}$であり，$e \in Exp(\mathcal{A},\mathcal{F})$である。操作 e_aは遷移 tによる状態遷移において，イベントが入力されてから外部へのイベントが出力される間に行なわれるものと規定する。

- $s_i \in S$
  状態図の初期状態

- $s_t \in S$ 状態図の終了状態
  状態図には図3(a)のようにサイクルを形成する場合と3(b)のようにある実行単位でだけ動作する場合の2つがある。(a)の場合は初期状

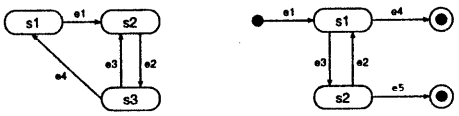

図 3: OMT の 2 種類の状態図

態 s_iが存在し, 終了状態は定義されない。(b)の場合では s_iおよび s_tに対して初期状態, 終了状態を与える。

図4の動的モデルに属性操作に関する記述を含めて記号表現に変換した例は以下のようになる。

待ち状態
=硬貨投入 (投入金.金額).金額計算
  { 自動販売機.投入金額 = 投入金.金額 }
金額計算
= 硬貨投入 (投入金.金額).金額計算
  { 自動販売機.投入金額 = 自動販売機.投入金額 + 投入金.金額 }
+商品選択 (商品.商品名).在庫確認
+取消.釣り銭返却 (釣り銭.金額).待ち状態
  { 釣り銭.金額 = 自動販売機.投入金額,
  自動販売機.投入金額 = 0}
在庫確認
= ϵ[在庫情報.在庫数 > 0].金額確認
+ϵ[在庫情報.在庫数 = 0].再操作要求.金額計算
金額確認
= ϵ[自動販売機.投入金額 $\geq$ 商品.価格].商品搬出 (商品).残高計算
  { 在庫情報.在庫数 = 在庫情報.在庫数 − 1}
+ϵ[自動販売機.投入金額 < 商品.価格].再操作要求.金額計算
残高計算
= ϵ.釣り銭返却 (釣り銭.金額).待ち状態
  { 釣り銭.金額 = 自動販売機.投入金額 − 商品.価格,
  自動販売機.投入金額 = 0}

3.1.3 動的モデル全体の表現

OMT の動的モデルは複数のクラスについて状態図を記述したものである。そこで, 動的モデル全体の記述は個々の状態図の表現の組として表現する。例えば n 個の状態図を持つモデルでは

$$(SC_1, \ldots, SC_n)$$

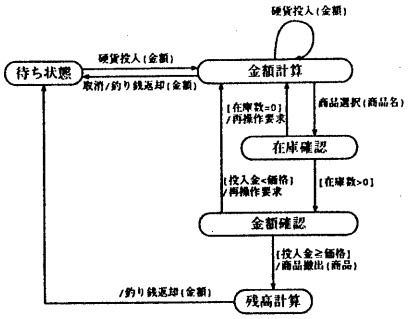

図 4: 自動販売機の動的モデル

である。ここで,

$$SC_1 = (S_1, E_{I1}, E_{O1}, C_1, T_1, A_1, s_{i1}, s_{t1})$$

$$SC_n = (S_n, E_{In}, E_{On}, C_n, T_n, A_n, s_{in}, s_{tn})$$

である。

3.1.4 モデルの状態と動作に関する制約

ある時点でのモデル全体の状態を考えるときには, それぞれの状態図の状態だけではなくそのときにモデル内に存在するイベントも含めて考えなければならない。イベントを含めた状態を考えるにはモデルがどのような動作をするかを表現する必要がある。

一般にはモデル内のイベントは同時に複数存在することができるが, ここでは簡単のためにある瞬間をみたとき, モデル内に存在できるイベントを 1 つに限定する。

モデル全体の状態は以下の 2 つ組で表現する。

$$(S_{\text{model}}, e)$$

ここで,

$$S_{\text{model}} = (s_1, \cdots, s_n)\ (s_1 \in S_1, \cdots, s_n \in S_n)$$

であり, $e \in \mathcal{E}$はモデル内に存在するイベントを表す。

状態図はそれぞれがが独立して動作できる。したがってモデルとしてはある時点で同時に複数の遷移を許すモデルを考えることができるし, その

方がより自然であろう。しかし，ここでは簡単のためにある時点で起こる状態遷移は1つに限定する。

モデル全体の状態遷移については基本的にモデル内に存在するイベントを受理できる状態図のみが遷移を行なうことができる。ただし，イベントなしで遷移することができる場合はいつでも遷移を行なえるものとする。ただし，図5のようにイベントを出力する遷移の場合，モデル内にイベントが存在している場合は遷移は起こらないものとする。また，モデル内にイベントが存在するときに

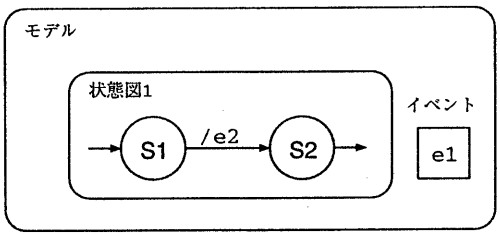

図 5: イベントなし遷移が制限される場合

そのイベントを入力として遷移ができる状態図が複数存在する場合は，そのうちのどちらか一方だけが遷移できるものとする。

動作に関して以下の制約を与えることでモデル全体を1つの状態図とみなすことが可能になる。これによって後述する状態遷移に関係する属性の依存関係を導出する手続きが容易に構築できる。

全体の状態遷移を次のように表す。遷移 $t : s_{i1} = e_{i1}.\overline{e_{i2}}.s_{i2} \in T_i$ のとき，モデル全体では

$$((s_1, \cdots, s_{i1}, \cdots, s_n), e_{i1}) \xrightarrow{t} ((s_1, \cdots, s_{i2}, \cdots, s_n), e_{i2})$$

のように遷移する。

3.2 機能モデルの記号表現

3.2.1 扱うモデルについて

OMT の機能モデルではプロセスを通じてデータの流れや属性の依存関係，制御フローなどの情報を記述できる。

本研究では属性の依存関係に着目した一貫性の検出を考えている。そこで，機能モデルについては属性のみに着目し，制御フローについては考慮しない。

3.2.2 表現

機能モデルの表現は以下の組で与えられる。

$$(A, P, D, E)$$

それぞれの記号は以下の意味を持つ。

- $A \subseteq \mathcal{A}$
  機能モデルで用いる属性の集合
- P
  プロセス名の集合
- D
  データストア名の集合
- E
  データフロー図の関数としての記述の集合

図 6の機能モデルを記号表記で記述すると以下のようになる。

自動販売機.投入額 = 投入額計算 (自動販売機.投入額, 投入金.金額)
(商品, 在庫情報) = 商品選択 (商品.商品名)
(商品, 在庫情報.在庫数) = 商品搬出 (商品.商品名, 在庫情報.在庫数)
釣り銭.金額 = 残額計算 (自動販売機.投入額, 釣り銭.金額)

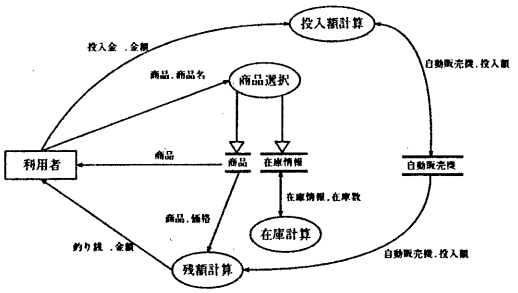

図 6: 機能モデルの例

4 依存関係の導出と一貫性の検出

4.1 属性依存グラフ

属性の依存関係はグラフとして表現することができる。この論文ではこのグラフを属性依存グラ

フと呼ぶ。属性依存グラフは

$$(A, D)$$

から構成される。

Aは属性の集合で, $A \subset \mathcal{A}$である。

Dは属性の依存関係の集合で, $D \subset A \times A$である。2つの属性$a_1, a_2 \in A$があるとき, $(a_1, a_2) \in D$は属性a_2の値は属性a_1の値に依存していることを表す。

4.2 機能モデルの依存グラフへの変形

機能モデルはプロセスを間において属性間の依存関係を表す表現になっている。そこで関数の入力と出力をそのまま依存関係に写像すれば属性依存グラフに変形できる。n個の引数を持つプロセスpについて

$$a = p(a_1, \cdots, a_n)$$

という表現があたえられたとき,

$$\begin{aligned} A &= \{a, a_1, \cdots, a_n\} \\ D &= \{(a_1, a), \cdots, (a_n, a)\} \end{aligned}$$

という形で属性依存グラフを表す。

データストアについては特定の属性を指す場合はその属性を, 全体を指す場合はデータストアが持つ属性すべてを指すものとする。データストアの属性依存グラフへの変形の例は図7のようになる。図7(a)で示したデータストアは図7(b)の依存グラフに変換される。図6の機能モデルを属性

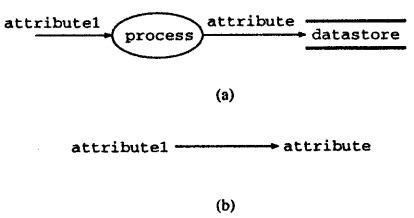

図 7: データストアの属性依存グラフへの変換

依存グラフに変換したものは図8のようになる。

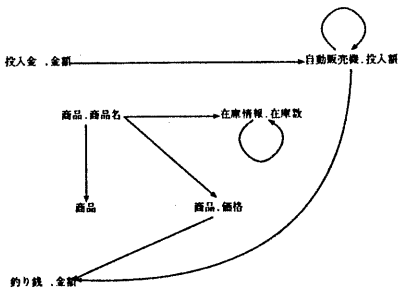

図 8: 図6の機能モデルの属性依存グラフ

4.3 動的モデルからの属性依存グラフの導出

動的モデル上では記述した属性の間に2種類の依存関係を考えることができる。ひとつはイベントに付属して状態図に渡される属性の間の依存関係であり, もうひとつは属性の操作にともなう依存関係である。

4.3.1 遷移にともなう依存関係

遷移にともなう依存関係は

$$(A_T, D_T)$$

で表す。

A_Tは遷移に関係する属性の集合である。A_Tにはイベントに付属して値が渡される属性と条件つき遷移の中で値が参照される属性がある。

D_TはA_Tの上の依存関係である。属性$a_1, a_2 \in A_T$があるとき, (a_1, a_2)はa_1が関係する遷移がa_2の遷移より時間的に前に起こっていることを表現する。

3つのイベント$e_1(a_1), e_2(a_2), e_3(a_3)$が存在して

$$e_1 \longrightarrow e_2 \longrightarrow e_3$$

という順序でイベントが現れるとき, (a_1, a_2), (a_2, a_3)という関係と同時に(a_1, a_3)という関係も成立する。つまり遷移にともなう依存関係では推移関係を仮定する。

4.3.2 属性操作にともなう依存関係

属性操作にともなう依存関係は

$$(A_O, D_O)$$

で表す。

A_Oは属性操作に現れる属性名である。

D_Oは A_O の上の依存関係で, $a_2 = f(a_1)$ のとき, (a_1, a_2) という関係が存在する。

属性操作にともなう依存関係は属性操作の式に現れる属性間にのみ現れ, 他に影響を与えることはないものとする。

4.4 依存グラフの導出

動的モデルからから属性の依存関係を導出することは基本的にはモデルが受理できる入力イベントの列を構成することと等価である。しかし, 無限のイベント列を求めることは不可能なのでイベント列のある部分から依存関係を導出することを考える。

上で与えた依存関係のうち, 属性操作にともなう依存関係はモデルが与えられた時点で一意に定まる。したがって問題になるのは状態遷移にともなう依存関係を導出することである。

状態遷移にともなう依存関係を導出する基本的な考え方は以下の通りである。まずある状態から可能な状態遷移をすべて求め, 遷移に関係する属性を取り出す。つぎに状態遷移が起こった後の状態からの依存関係を求め, 前に求めた属性をそこに付加する。ここで無用な探索を避けるために一度通った状態遷移にはマークをつけて区別する。

モデルの初期状態として i を定義する。ここで

$$i = ((s_{i1}, \cdots, s_{in}), \{\epsilon\})$$

である。i は状態図がすべて初期状態にあり, モデル内にイベントが存在しない状態である。

ある状態 s を与えたときに状態遷移にともなう属性の依存関係を導出する手続き get_dependency を以下で与える。

```
get_dependency(DM : 動的モデルの表現, s : DMのある状態)
/* 出力 : 状態遷移にともなう属性依存グラフ (A_T, D_T) */

A_T ← ∅;
D_T ← ∅;
s から可能な状態遷移の集合 T_s を求める。
while T_s = ∅ do
      T_s の要素 t を取り出す;
      t の状態遷移にともなう属性の集合 A を求める;
      if (マークが付いていない (t) or
        イベントなし遷移 (t)) then
        マークをつける (t);
        /* この操作は大域的なものである。 */
        t による状態遷移の後の状態 s' を求める;
        (A_T1, D_T1) =get_dependency(DM, s');
        /* s' からの属性依存グラフを再帰的に求める。*/
        D_T = D_T ∪ D_T1 ∪ {(a, a1)|a ∈ A, a1 ∈ A_A1};
        A_T = A_T ∪ A_T1 ∪ A;
      else
        A_T ← A_T ∪ A;
      fi
      od
(A_T, D_T) を返す;
```

初期状態 i から属性依存グラフを導出するアルゴリズムを以下に示す。

```
入力: DM : 動的モデル, i : DMの初期状態
出力: (A_DM, D_DM) : 動的モデルから得られる属性依存グラフ

DMの属性操作に関する属性依存グラフ (A_O, D_O) を構成する;
DMのすべての遷移のマークをクリアする;
(A_T, D_T) = get_dependency(DM, i);
(A_DM, D_DM) = (A_T ∪ A_O, D_T ∪ D_O);
```

図 4の状態図を含む動的モデルから上のアルゴリズムによって導出される属性依存グラフは図 9のようになる。

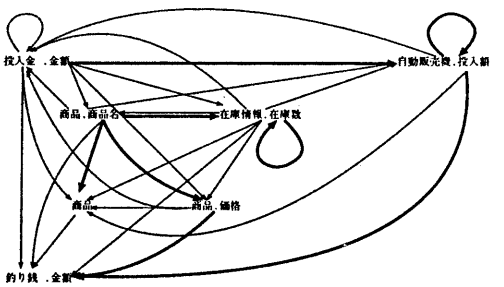

図 9: 動的モデルから導出される属性依存グラフの例

5 記述の一貫性について

動的モデルから得られる属性依存グラフは状態遷移に関する属性の依存関係をすべて圧縮して表現しており, それに属性の操作に関する依存関係が付加されている。これは一般には機能モデルを

変形して得られる属性依存グラフより冗長な情報を保持していると考えてよい。また, 機能モデルから得られる属性依存グラフは時間的あるいは操作的な属性の依存関係のうちで属性の値に対して有効な依存関係のみを表現したものと考えることができる。

そこでこの手法にもとづく2つのモデルの一貫性は以下のように定義する。

> 機能モデルを変形して得られる属性依存グラフが動的モデルから導出される属性依存グラフの部分グラフになっているとき, 機能モデルで記述が与えられる属性に関しては動的モデルと機能モデルは一貫性がとれている。

上の図8の属性依存グラフと図9では, 図9の属性依存グラフの太い線で示した部分は図8の属性依存グラフと一致する。したがってこの2つのモデルは属性の依存関係について一貫性がとれているといえる。

6 評価

- 利点
  この手法では動的モデルが与えられると属性依存グラフを静的に計算することができる点およびクラスレベルでの一貫性の検証を行なうことができる点がある。

- 問題点
  この手法で検証できるのは属性に関する一貫性のみである。機能モデルのプロセスがどのオブジェクトで行なわれるべき計算であるかといった情報を検出することはできない。また, 動的モデルから導出される属性依存グラフでは冗長な依存関係も多く含まれている。そのために誤った依存関係が検出できない場合がある。

- 動作モデルについて
  動作に関するモデルもまだ十分なものであるとはいえない。今回の動作モデルではモデル内に存在できるイベントを1つに制限することでモデル全体を1つの状態図とみなしている。これは属性依存グラフの導出を容易にしている一方で大きな制約ともなっている。

7 まとめ

OMT の動的モデルと機能モデルについて記号による表記を与え, 属性の依存関係にもとづく一貫性の検証手法を提案した。今後の課題としては上にあげた問題点の改善および OMT 全体にわたる検証手法の確立があげられる。

参考文献

[1] Stephen Bear, Phillip Allen, Derek Coleman, and Fiona Hayes. Graphical Specification of Object Oriented Systems. In *ECOOP/OOPSLA '90 Proceedings*, pp. 28–37. ACM, 1990.

[2] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the Formal Semantics of Statechats. pp. 54–64. IEEE, 1987.

[3] David Harel. STATECHARTS: A VISUAL FORMALISM FOR COMPLEX SYSTEMS. *Science of Computer Programming*, Vol. 8, pp. 231 – 274, 1987.

[4] Fiona Hayes and Derek Coleman. Coherent Models for Object-Oriented Analysis. In *OOPSLA '91, Conference on Object-Oriented Prorramming Systems, Languages, and Applications*, pp. 171–183. ACM, 1991.

[5] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[6] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *OBJECT-ORIENTED MODELING AND DESIGN*. Prentice-Hall, 1991.

[7] 大西淳. オブジェクト指向分析における機能モデル検証支援. 情報処理学会研究報告 ソフトウェア工学, pp. 9 – 16, 1994.