ソフトウェア工学 90−13
（1993. 2. 5）

スパイラル型開発モデルにおける 要求実行例・仕様・プログラム相互間の対話的検証手法

大塚 玲，　菅谷 光啓
(株) 野村総合研究所

要求を満足するソフトウェアの開発にはスパイラル型の開発モデルの実施が理想的である．この開発モデルの実施には，仕様変更に要する負担の軽減が不可欠である．このため，仕様記述全体の一貫性を維持できる検証手法の確立が急がれている．本論文では，オブジェクト指向ソフトウェア仕様の特にオブジェクトの振る舞いの側面に着目し，プロセス代数 ACP$^{\tau}$によるモデル化および，このモデルを用いた検証手法を示す．ここで示す検証手法は，(1) 要求を表す実行例記述により仕様およびプログラムの妥当性を検証する，(2) 検証結果をもとにした利用者との対話を通して段階的に，完全な仕様記述を獲得する，(3) 仕様記述の誤りを例により具体的に指摘できるなどの特長を持つ．

A Verification Method among Example, Specification and Program in the Spiral Model of Software Development

Akira Otsuka,　Mitsuyoshi Sugaya
Nomura Research Institute, Ltd.
134 Godo-cho, Hodogaya-ku, Yokohama, Kanagawa 240, Japan

The spiral model of software development is ideal to satisfy user-requirements. In executing this development model, it is essential to reduce the cost of modifying specification. Therefore, a verfication method is strongly expected that can maintain the consistency of whole specification. This paper mainly focuses on the mathematical model and the verification method for object-oriented software specification. Especially on the behavioral aspect of object, based on process algebra ACP$^{\tau}$. Our verification method can (1)validate specification using execution examples, by which user-requirements are easily demonstrated, (2)acquire complete specification through user-interaction, based on verification results, and (3)demonstrate error cases by example.

1 はじめに

真に要求を満足するソフトウェアの開発には，プロトタイプの試用/運用を通して具体的に得られる改善要求をもとに，仕様変更/改良を行なうスパイラル型の開発モデル[1]の実施が理想的である．しかし，スパイラル型開発モデルは再設計を積極的に促すため，仕様変更に伴う修正作業の負担が大きい現状では仕様書とプログラムの乖離を避けられない．

仕様書とプログラムの乖離は，仕様書の変更がプログラム動作に影響しないために生じる．従って，乖離を避けるには，仕様書が単にドキュメントとしてだけではなく，プログラムと強い関連を持つ仕様記述としての機能を果たさなければならない．

オブジェクト指向型ソフトウェアは自然な計算モデルを持つため，仕様書に記述される分析/設計モデルの構造とプログラム構造の類似性が高い．この類似性を用いれば，仕様書とプログラムの一貫性を検証できる可能性がある．

検証性の高い仕様記述方式として，代数的仕様やLOTOSなどが提案されている．しかし，これらを直接記述するには，一般に高度な知識と熟練を要するため，(1)記述した仕様の要求に対する合致性の検証が難しく，(2)設計者に十分な教育が必要になる，などの問題がある．

従って，このような高度な知識や教育を要求せずに，仕様書の一貫性を保証できる枠組が必要になる．我々は，オブジェクト指向ソフトウェアモデリングに基づいた仕様書の総合的な一貫性保証の枠組について考察し，この中で，(1)要求する実行例記述の導入により仕様・プログラムの妥当性が判定できること，(2)仕様記述の意味と表現の分離，および対話により段階的に必要な情報を獲得することにより，多様かつあいまいな記述を仕様記述に導入できることを見い出した．

本論文では，これらの考察をまとめるとともに，(3)プロセス代数ACP$^{\tau}$によりオブジェクトの振る舞いの側面をモデル化し，要求実行例・仕様・プログラム相互間の検証手法を示す．この検証手法は，必要な情報を対話的に獲得できる特長をもつ．

2 仕様記述の表現と意味

2.1 多様な図表を許す仕様記述方式

形式的な仕様記述は，記述の無矛盾性などを自動的に判定できる利点がある．特に，再設計を促すスパイラル型の開発モデルでは，仕様変更により矛盾の発生する箇所を自動的に判定できる形式仕様が不可欠である．しかし一方で，仕様書は利用者にも理解してもらう必要があり，一般にわかりにくくなる形式的仕様記述はこの要求を満たさない．

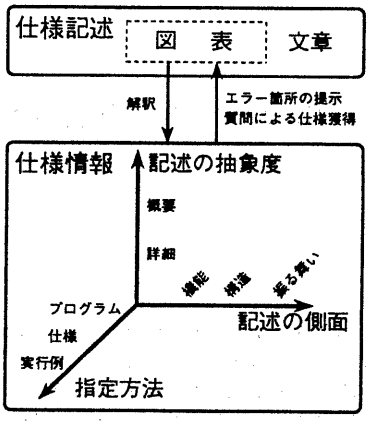


図 1: 仕様書の一貫性維持方式の概念図

実際の仕様書は利用者へのわかりやすさを重視して記述してあるため，多くの記述は非形式的である．しかし，仕様書中で用いられている図や表は既にある程度の形式性を持っており，図表文法の多少の変更によって，もとの解釈と矛盾なく自然に形式性を導入できる．

図表には統一的な表記法がなく，同一意味に別の表記法を用いたり，同一表記法でも意味が異なる．現場の仕様書が利用できるようにするには，これらの違いを柔軟に許す必要がある．この表現上の違いは，記述の表現と意味を分けて扱い，図表の解釈を表現から意味への写像とすることで，独立に表現，意味，解釈を定義することで解決できる．

この方式の概念図を図1に示す．仕様記述の意味を検証性のある数学的モデルで表し，図表の解釈をこの数学的モデルへの写像として与える．一方，図表の形状や配置などは，表現上の情報として考え，(1)仕様情報の獲得，(2)エラー情報の提示など，利用者との対話に用いる．このように表現と意味を明確に分離しておけば，多様な表現形式を導入でき，かつ解釈を自由に変更できるため，現場の仕様書で用いられている図表の導入が容易になる．

2.2 仕様情報の構造

仕様書およびプログラムは，ソフトウェアを多様な側面から記述している．仕様書およびプログラムの一貫性を保証する数学的モデルは，それらの多様な側面を網羅的に表現できなければならない．これらの側面は，(1)段階的詳

細化 (step-wise refinement) による記述の抽象度，(2) 機能，構造，振舞いなどの対象を指定する側面，(3) 例，仕様，実現法あるいはプログラムといった対象の指定方法，に大別される．

2.2.1 記述の抽象度 (概要仕様と詳細仕様)

オブジェクト指向モデリングは，問題領域に存在するオブジェクトを発見する作業から始まる．モデリングの初期には，サブシステムを表すオブジェクトの仕様を記述し，モデリングの進行に伴い徐々に詳細なオブジェクトの仕様記述に移るといった分析プロセスをとる．

記述の抽象度は，相対的な概念である．すなわち，あるオブジェクトは，さらに複数のサブ・オブジェクトから構成され，サブ・オブジェクトもさらに詳細なオブジェクトに分解される．あるオブジェクトが満たすべき外部インタフェイスは，そのサブ・オブジェクト集合も同様に満足する必要がある．異なる抽象度の記述は，共通な外部インタフェイスにより相互に関連する．

2.2.2 機能・構造・振舞い

システム全体の機能は，(1) 個々のオブジェクトの機能が (2) 構造を持って空間に配置され，(3) 振舞いによって因果的に関係付けられた順序に従って実行されることにより達成される．すなわち，実行可能なプログラムはこれらの側面を全て含んでいる．このような 3 つの側面に分割する考え方は，Rumbaugh ら [3] にも指摘されており，対象を単純化して考えられるため，仕様記述およびプログラムの把握が容易になる利点がある．

2.2.3 要求実行例・仕様・プログラム

対象を指定する方法には，(1) 例を示す，(2) 条件により制約する，(3) 実現法を示す，の 3 つがある．従って，仕様書に含まれる指定方法もこの 3 つに分類される．特に，例は指定がもっとも容易で，要求を示すのに都合が良い．このため，例は要求を表すと考えることができ，記述が複雑になる形式仕様あるいはプログラムの要求に対する妥当性の判定に利用できる．

要求実行例，仕様，プログラムは，相互に関連しており，仕様情報の数学モデルはこれら 3 つの情報について，相互に検証できる必要がある．

2.3 仕様情報を表す数学的モデル

仕様情報の構造を全て包含できるような数学的モデルは見い出しにくい．しかし個別の側面については，表 1に示すように，いくつかの数学的モデルが既に存在している．本方式では，仕様記述の意味をこれらの数学的モデルに写像して，仕様情報の獲得および検証を行う．

表 1: 仕様情報の獲得および検証に用いる数学的モデル

	数学的モデル
機能	型，代数的仕様
構造	Cardinality
振舞い	プロセス代数

3 振る舞いのモデル

以下では，後のオブジェクトの振る舞いのモデル化に必要な ACP$^\tau$による振る舞いの式表現をいくつか解説し，これを用いてオブジェクトの振る舞いをモデル化する方法を示す．

また，このモデルと仕様書の図表との関連を示すために，代表的な図式と ACP$^\tau$での解釈を例を用いて示す．

3.1 ACP$^\tau$

ACP$^\tau$は原子アクションの集合 A と通信関数γで定められる代数系で，以下の定数と演算子を持つ．ACP$^\tau$の公理系を表 2に示す．ACP$^\tau$の厳密な定義は文献 [2] を参照．ここでは，ACP$^\tau$を直観的に説明する．

- 定数
  - a 原子アクション ($a \in A$)
  - δ デッドロック ($\delta \notin A$)
  - τ 内部アクション ($\tau \notin A$)
- 2項演算子
  - $+$ 選択
  - $\cdot$ 連接
  - $\parallel$ 並列 (merge)
  - $\between$ 左並列 (left-merge)
  - $\mid$ 通信
- 単項演算子
  - ∂_H 隠蔽オペレータ
  - τ_I 抽象化オペレータ

(プロセス) プロセスは，ACP$^\tau$[2] により振る舞いが厳密に決められた動作主体である．後に，オブジェクトの振る舞いをこのプロセスとして定義する．

(原子アクション a) 原子アクションは，プロセスの動作の最小単位である．後に，メッセージをこの原子アクションとして定義する．

(プロセス動作式) プロセス動作式は，プロセス変数 (あるいは単にプロセス)$x, y, \ldots$ を左辺に持ち，ACP$^\tau$の項を右辺に持つ等式で定義される．

$$x = t$$

項 t が左辺の変数を含むとき再帰的であると言う．さらに，下式のように t が別のプロセス変数 yを含み，yのプロセス動作式が右辺に x, yを含むとき，相互再帰的である

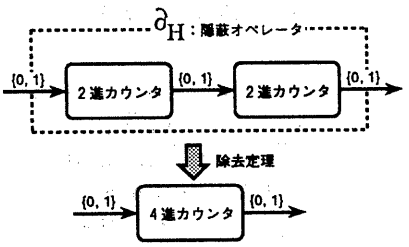


図 2: プロセス関通信の例

という．

$$\begin{cases} x = t(x,y) \\ y = t'(x,y) \end{cases}$$

(プロセス間通信) プロセス x,yによる通信を，$\partial_H(x \parallel y)$ のように書く．この$\partial_H(x \parallel y)$ の通信は，通信路を定義する関数γと，内部通信路を隠蔽して外部インタフェイスだけを取り出す隠蔽オペレータ∂_Hにより定義される．

(例 1) 図 2に示す 2 つの 2 進カウンタ x,yの通信 $\partial_H(x \parallel y)$ を考える．2進カウンタ x,yは，次のように定義できる．但し,$0^1, 1^1$は通信路 1 への 0,1 出力，$\dot{0}^1, \dot{1}^1$は通信路 1 からの 0,1 入力を表す．

$$\begin{cases} x = 0 \cdot 0^1 \cdot x + 1 \cdot 0^1 \cdot x_1 \\ x_1 = 0 \cdot 0^1 \cdot x_1 + 1 \cdot 1^1 \cdot x \\ y = \dot{0}^1 \cdot 0 \cdot y + \dot{1}^1 \cdot 0 \cdot y_1 \\ y_1 = \dot{0}^1 \cdot 0 \cdot y_1 + \dot{1}^1 \cdot 1 \cdot y \end{cases}$$

原子アクション全体の集合 A，通信路を決める通信関数 γ，および外部から隠蔽する原子アクションの集合 Hを下のように定義する．

$$A = \{0, 1, 0^1, 1^1, \dot{0}^1, \dot{1}^1, \ddot{0}^1, \ddot{1}^1\}$$
$$\gamma(0^1, \dot{0}^1) = \ddot{0}^1$$
$$\gamma(1^1, \dot{1}^1) = \ddot{1}^1$$
$$H = \{0^1, 1^1, \dot{0}^1, \dot{1}^1\}$$

通信関数γは，特定の通信路上の受信アクション $0^1, 1^1$と送信アクション$\dot{0}^1, \dot{1}^1$の正しい組み合わせに対して定義する．式中の原子アクション$\ddot{0}^1, \ddot{1}^1$は，通信が正しく行われたことを示す記号で，後で通信の正しさを検証するときに用いる．

(除去定理 elimination theorem) $\{\parallel, \between, |\}$ を含む項 t は，公理を用いて並列および通信を含まない項に変形できる．この変形は，項書換え系により自動化できる．

(例 2) 前述の 2 進カウンタの通信は正規化手続きにより 4 進カウンタに変形できる．尚，$\{\ddot{0}^1, \ddot{1}^1\}$ は内部通信の結果を表す原子アクションである．

$$\partial_H(x \parallel y) = z, \begin{cases} z = 0 \cdot \ddot{0}^1 \cdot 0 \cdot z + 1 \cdot \ddot{0}^1 \cdot 0 z_1 \\ z_1 = 0 \cdot \ddot{0}^1 \cdot 0 \cdot z_1 + 1 \cdot \ddot{1}^1 \cdot 0 z_2 \\ z_2 = 0 \cdot \ddot{0}^1 \cdot 0 \cdot z_2 + 1 \cdot \ddot{0}^1 \cdot 0 z_3 \\ z_3 = 0 \cdot \ddot{0}^1 \cdot 0 \cdot z_3 + 1 \cdot \ddot{1}^1 \cdot 1 z \end{cases}$$

表 2: ACP$^\tau$の公理

$x+y = y+x$	A1
$(x+y)+z = x+(y+z)$	A2
$x+x = x$	A3
$(x+y)z = xz+yz$	A4
$(xy)z = x(yz)$	A5
$x+\delta = x$	A6
$\delta x = \delta$	A7
$x \parallel y = x \between y + y \between x + x \mid y$	CM1
$a \between y = ay$	CM2
$ax \between y = a(x \parallel y)$	CM3
$(x+y) \between z = x \between z + y \between z$	CM4
$ax \mid b = (a \mid b) \cdot x$	CM5
$a \mid bx = (a \mid b) \cdot x$	CM6
$ax \mid by = (a \mid b) \cdot (x \parallel y)$	CM7
$(x+y) \mid z = x \mid z + y \mid z$	CM8
$x \mid (y+z) = x \mid y + x \mid z$	CM9
$\partial_H(\tau) = \tau$	D0
$\partial_H(a) = a$ if $a \notin H$	D1
$\partial_H(a) = \delta$ if $a \in H$	D2
$\partial_H(x+y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4
$x\tau = x$	T1
$\tau x = \tau x + x$	T2
$a(\tau x + y) = a(\tau x + y) + ax$	T3
$a \mid b = \gamma(a,b)$ if $\gamma(a,b)$ defined	CF1
$a \mid b = \delta$ otherwise	CF2
$\tau \between x = \tau x$	TM1
$\tau x \between y = \tau(x \parallel y)$	TM2
$\tau \mid x = \delta$	TC1
$x \mid \tau = \delta$	TC2
$\tau x \mid y = x \mid y$	TC3
$x \mid \tau y = x \mid y$	TC4
$\tau_I(\tau) = \tau$	TI0
$\tau_I(a) = a$ if $a \notin I$	TI1
$\tau_I(a) = \tau$ if $a \in I$	TI2
$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$	TI3
$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	TI4

3.2 オブジェクトのモデル化

オブジェクトをプロセスとし，メッセージを原子アクションとして ACP$^\tau$によりモデル化する．モデル化に際して，(1) 仕様記述の上流では，オブジェクトの振舞いが実世界を反映して並行動作になること，(2) オブジェクトの協調動作の際に，メッセージの到着順序に関して約束 (プ

ロトコル)があること，を考慮する必要がある．ここでは，(1) メッセージ受信を非同期的として複数メソッドの並列実行を許し，(2) オブジェクトに状態概念を導入して，個々の状態で受理できるメッセージを限定することによりプロトコルを表現する．

定義 1 入出力メッセージ

記述に現れる全メッセージの集合を $M = \{m_1, \ldots, m_n\}$ と書く．また，入出力を区別して，

$$\dot{m}_i \quad : \text{入力メッセージ}$$
$$m_i \quad : \text{出力メッセージ}$$

のように表す．ここで，m_iは記述中のメッセージで $m_i \in M$である．以後，単に m_iと書いて $m_i \in M$を表す．メッセージを原子アクションと考え，$A = M \cup \dot{M}(\dot{M} = \{\dot{m}_1, \ldots, \dot{m}_n\})$ とする． □

定義 2 動作仕様

動作仕様はメソッドの振る舞いである．動作仕様は，メッセージ式，連接，選択，繰り返しなどのプログラムの処理構造に対応するプロセス動作式 p, p', p''により表現される．

メッセージ式	$p = m_i$
選択	$p = \tau p' + \tau p''$
連接	$p = p' \cdot p''$
繰り返し (while)	$p = \tau \cdot p' \cdot p + \tau$

□

一般に，機能は複数のオブジェクトの協調動作により実現される．個々のオブジェクトは，処理の一部を別のオブジェクトに依頼して自己の機能を実現し，この処理の依頼は，一般には複数のメッセージを送る必要がある．この処理依頼に関わるメッセージ通信には，ある守るべきプロトコルが存在する．これには，例えば初期化メッセージの後でないと誤動作するメッセージなどが考えられる．このプロトコルは，処理を提供する側のオブジェクトに定義すべき仕様であるから，オブジェクトの入力仕様として考える．

定義 3 入力仕様

入力仕様は，オブジェクトの複数の入力メッセージ上に定義されるメッセージ通信のプロトコルである．入力仕様 s は，下に示すような状態遷移の初期状態 s_1を用いて $s = s_1$で与えられる．

$$\begin{bmatrix} s_1 \\ \vdots \\ s_k \end{bmatrix} = \begin{bmatrix} t_{11} & \cdots & t_{1k} \\ \vdots & & \vdots \\ t_{k1} & \cdots & t_{kk} \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_k \end{bmatrix}$$

$$t_{ij} = \begin{cases} \delta & \text{if } I(s_i, s_j) = \emptyset \\ \displaystyle\sum_{m_l \in I(s_i, s_j)} \dot{m}_l & \text{otherwise} \end{cases}$$

但し，$I(s_i, s_j)$は遷移 $s_i \to s_j$のトリガとなる入力メッセージの集合とする． □

実世界の事象は分散的，並列的に進行するため，分析段階の仕様記述はこれを反映して分散並列的にモデリングされる．オブジェクトの動作モデルは，この並列性を記述できなければならない．並列性は粒度により，(1) オブジェクト単位の並列動作とする方法と，(2) メッセージ単位の並列動作とする方法がある．ここでは，因果関係が並列性の本質的な表現であるとの観点から (2) の方法を採用し，メッセージの非同期的な到着に対して並列にメソッドを実行する方式で，オブジェクトの並行動作をモデル化する．

定義 4 オブジェクトの動作

オブジェクトの動作は，前述の動作仕様と入力仕様により記述される．最も簡単な場合として，入力仕様が与えられない時の動作モデルは，オブジェクト obj_iの受理できるメッセージ$\mu(obj_i) = \{\dot{m}_{i1}, \ldots, \dot{m}_{in}\}$ と対応するメソッド $p_{i1}, \ldots, p_{in}$を用いて，

$$obj_i = \dot{m}_{i1} p_{i1} \cdot obj_i \,\|\, \cdots \,\|\, \dot{m}_{in} p_{in} \cdot obj_i$$

として記述される．ここで，$\|$ は並列オペレータで全てのメッセージ$\dot{m}_{ij}$が独立に受信可能なことを示す．

入力仕様 $S = \{s_1, \ldots, s_m\}$ が与えられた時のオブジェクト obj_iの動作は，入力仕様 $s_k \in S$に含まれる各メッセージ$\dot{m}_{ij} \in \mu(obj_i)$ の出現を$\dot{m}_{ij} p_{ij}$で置き換えた新しい入力仕様 $s'_k \in S'$を用いて，

$$\begin{aligned} obj_i = \quad & s'_1 \,\|\, \cdots \,\|\, s'_m \\ & \|\, \dot{m}_{il} p_{il} \cdot obj_i \,\|\, \cdots \,\|\, \dot{m}_{in} p_{in} \cdot obj_i \end{aligned}$$

と表される．ここで，$\{\dot{m}_l, \ldots, \dot{m}_n\}$ は，どの入力仕様にも制約されないメッセージとする． □

定義 5 オブジェクト間の通信

オブジェクト間の通信路は，オブジェクトを節点，通信路を有向枝とする有向グラフで表現され，各通信路にメッセージの集合を定義する．

まず，例 1 と同様に通信に関わるメッセージ名を変更する．今，通信路集合 $C = \{c_1, \ldots, c_m\}$ の各通信路 c_lにメッセージ集合$\mu(c_l)$ が定義されているとする．通信路の入力側および出力側に接続されたオブジェクト obj_j, obj_kのプロセス動作式中の各 $m_i, \dot{m}_i \in \mu(c_l)$ の出現をそれぞれ $m_i^l, \dot{m}_i^l$に変更する．次に，通信関数γも，メッセージ名の変更に合わせて$\gamma(m_i^l, \dot{m}_i^l) = \dot{m}_i^l$のように定義する．

但し，(1) 一つのオブジェクト obj_jから複数の通信路 C_1へ同一メッセージが出力される場合，および (2) 一つのオブジェクト obj_jへ複数の通信路 C_2から同一メッセージが

入力される場合には，メッセージ名をこのように変更できない．この場合は,$m_i, \dot{m}_i$をそれぞれ$m_i^*, \dot{m}_i^*$のように通信路の指定を持たない名前に変更する．また，このときの通信関数γは，

(1) $c_l \in C_1 \rightarrow \gamma(m_i^*, \dot{m}_i^l) = \ddot{m}_i^*$

(2) $c_l \in (C_2 - C_1) \rightarrow \gamma(m_i^l, \dot{m}_i^*) = \ddot{m}_i^l$

のように定義する．

以上に述べたメッセージ名の変更方法および通信関数γの定義を用いて，オブジェクト間の通信を次のように表す．

$$\partial_H(obj_1 \| \ldots \| obj_n)$$

ここで,$H = \bigcup_{c_l \in C} \{m_i^l, \dot{m}_i^l, m_i^* | m_i \in \mu(c_l)\}$ である． □

3.3 記述の解釈

C++プログラム C++プログラムは，実行に関わる全ての情報を含む記述である．C++の実行モデルがメッセージ送受を基本とする点に着目して，記述中のメッセージ式と連接，選択，繰り返し構造から ACP$^\tau$の式を得る．

```
void Viewer::Manipulate (Manipulator* m, Event& e) {
  Listen(allEvents);
  m->Grasp(e);

  do {
    Read(e);
  } while (m->Manipulating(e));

  m->Effect(e);
  Listen(input);
}
```

(a) OBJ-A の C++プログラム [4]

m_1 = **Grasp**
m_2 = **Manipulating**
m_3 = **Effect**
m_4 = **Manipulate**
m_5 = **Listen**
m_6 = **Read**

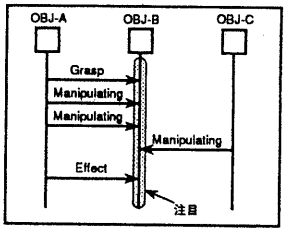
OBJ-A$= \dot{m}_4 m_5 m_1 m_1 m_2 p \; m_3 m_5$OBJ-A
$p = \tau m_6 m_2 p + \tau$

(b) OBJ-A のプログラムの ACP$^\tau$による表現

図 3: C++プログラムの ACP$^\tau$によるモデル化

シーケンスチャート (SC) シーケンスチャートは，前述の MFD で定義された通信路上での動的な側面を表すために，メッセージ通信の実行例を記述する図である．

1. **入力メッセージに注目して解釈した場合**オブジェクトへの入力メッセージの列を抽出し，入力仕様の妥当性の判定に必要な要求実行例を得る．



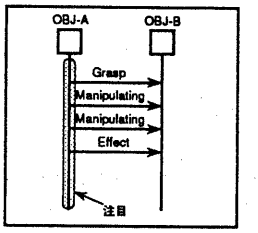
(a) シーケンスチャート

$\dot{m}_1 \dot{m}_2 \dot{m}_2 \dot{m}_2 \dot{m}_3$

(b) (a) の ACP$^\tau$による表現 (1)

図 4: シーケンスチャートの ACP$^\tau$によるモデル化 (1)

2. **出力メッセージに注目して解釈した場合**オブジェクトへの出力メッセージの列を抽出し，動作仕様の妥当性の判定に必要な要求実行例を得る．



(a) シーケンスチャート

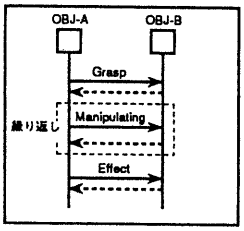
$m_1 m_2 m_2 m_3$

(b) (a) の ACP$^\tau$による表現 (2)

図 5: シーケンスチャートの ACP$^\tau$によるモデル化 (2)

拡張シーケンスチャート (ESC) 拡張シーケンスチャートは，SC に連接，選択，繰り返しの処理構造を導入した図で，オブジェクトの動作仕様の定義に用いる．C++プログラムとは，(1) メッセージ通信に関与する記述のみを含む，(2) 通信路の指定が可能である，などの点で異なる．図中に指定された連接，選択，繰り返し構造から ACP$^\tau$による動作仕様を得る．

状態遷移図 (STD) 状態遷移図は，オブジェクトの状態とメッセージ送受による状態の遷移関係の記述である．ここでは，状態遷移をメッセージ受信のプロトコル定義として捉え，遷移関係と遷移の入力となるメッセージ集合から

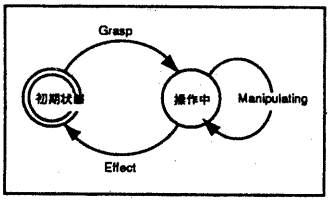


(a) 拡張シーケンスチャート

$$\text{OBJ-A} = m_1 q m_3 \text{OBJ-A}$$
$$q = \tau m_2 q + \tau$$

(b) (a) の ACP$^\tau$での表現

図 6: 拡張シーケンスチャートの ACP$^\tau$によるモデル化



(a) OBJ-B の状態遷移図

$$\text{OBJ-B} = \dot{m}_1 q \text{OBJ-B}$$
$$q = \dot{m}_2 q + \dot{m}_3$$

(b) (a) の ACP$^\tau$での表現

図 7: 状態遷移図の ACP$^\tau$によるモデル化

入力仕様を得る.

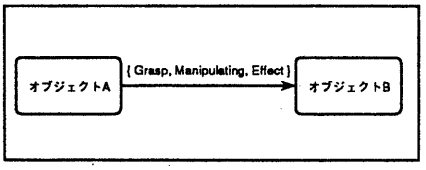
メッセージフロー図 (MFD) メッセージフロー図は, オブジェクトと通信路および通信路上で授受されるメッセージ集合から構成され, ある局面でのシステムの構造を明示するのに用いる. これらの記述から, 通信関数γとメッセージ名の変更に必要な情報 (定義 5 を参照) を得る.

4 検証手法

4.1 実行例と仕様

SC などから得られるメッセージ通信の要求実行例を用いて, STD などから得られる入力仕様の妥当性を検証する.

これは, 実行例を表すプロセス $e = e_1 \cdot e_2 \cdot \ldots \cdot e_n$, $(e_i \in M)$ と, 入力仕様を表すプロセス $s = (s_1 \parallel s_2 \parallel \cdots \parallel s_m) \cdot s$ について, $\partial_H(e \parallel s)$ を調べることにより妥当性を判定できる.



(a) メッセージフロー図

$$\gamma(m_1, \dot{m}_1) = \ddot{m}_1$$
$$\gamma(m_2, \dot{m}_2) = \ddot{m}_2$$
$$\gamma(m_3, \dot{m}_3) = \ddot{m}_3$$

(b) (a) の ACP$^\tau$による表現

図 8: メッセージフロー図の ACP$^\tau$によるモデル化

除去定理により, $\partial_H(e \parallel s)$ は次のように変形できる.

$$\begin{aligned} \partial_H(e \parallel s) = {} & \ddot{e}_1 \cdots \ddot{e}_n \partial_H(s) && \text{(a)} \\ & + \Sigma\{\ddot{e}_1 \cdots \ddot{e}_n \partial_H(x \cdot s) \text{ の形の項}\} && \text{(b)} \\ & + \Sigma\{\ddot{e}_1 \cdots \ddot{e}_k \cdot \delta \text{の形の項} \ (1 \le k < n)\} && \text{(c)} \end{aligned}$$

ここで, x は任意の項で, $H = \{e_1, \ldots, e_n, \dot{e}_1, \ldots, \dot{e}_n\}$, $\gamma(e_i, \dot{e}_i) = \ddot{e}_i (1 \le i < n)$ である.

(a) の項は入力仕様が正しく例を満たしていることを表し, (b) の項は初期状態へ復帰しない状態遷移を表し, (c) の項は途中で失敗する状態遷移を表す.

この結果から, 仕様の妥当性に関して次が言える.

(a) の項があるとき　仕様が例を満たす,
(b) と (c) だけのとき　例の記述が不完全である,
(c) だけのとき　仕様が例を満たさない.

4.2 実行例とプログラム

SC などから得られるメッセージ通信の要求実行例を用いて, C++プログラム, ESC などから得られる動作仕様の妥当性を検証する.

これは, 実行例を表すプロセス $e = e_1 \cdot e_2 \cdot \ldots \cdot e_n$, $(e_i \in M)$ と, 動作仕様を表すプロセス p(定義 2) について, $\partial_H(e \parallel \tau_I(p))$ を調べることにより妥当性を判定できる.

除去定理により, $\partial_H(e \parallel \tau_I(p))$ は次のように変形できる.

$$\begin{aligned} \partial_H(e \parallel \tau_I(p)) = {} & \Sigma\{\tau \cdot \ddot{e}_1 \cdots \ddot{e}_n \partial_H(t) \text{ の形の項}\} && \text{(a)} \\ & + \Sigma\{\ddot{e}_1 \cdots \ddot{e}_n \partial_H(t) \text{ の形の項}\} && \text{(a')} \\ & + \Sigma\{\ddot{e}_1 \cdots \ddot{e}_k \cdot \delta \text{の形の項} \ (1 \le k < n)\} && \text{(b)} \end{aligned}$$

ここで, $H = \{e_1, \ldots, e_n, \dot{e}_1, \ldots, \dot{e}_n\}$, $\gamma(e_i, \dot{e}_i) = \ddot{e}_i (1 \le i < n)$ である. また, $\mu(e)$ を e に含まれるメッセージの集合, $\mu(p)$ を p に含まれるメッセージの集合とするとき,

$\tau_I(p)$ は $I = \mu(p) - \mu(e)$ として，p 中に現れる $m_i \in I$の全ての出現をτに置き換えた動作仕様とする．

(a),(a') の項は動作仕様が正しく例を満たしていることを表し，(b) の項は途中で失敗する状態遷移を表す．

この結果から，仕様の妥当性に関して次が言える．

(a)(a') の項があるとき　仕様が例を満たす，

(b) だけのとき　仕様が例を満たさない．

4.3 仕様とプログラム

STD，ESC，C++プログラムから得られるオブジェクトの動作 (定義 4) と，MFD から得られる通信路 (γ，名前の変更規則) を用いて，オブジェクト間の通信の正しさ，すなわちオブジェクトの協調動作の実行可能性を検証する．

オブジェクト $\{obj_1, obj_2, \ldots, obj_m\}$ 間の通信は，定義 5より$\partial_H(obj_1 \| obj_2 \| \cdots \| obj_m)$で表され，これを調べることにより判定できる．

除去定理により，$\partial_H(e \| s)$ は次のように変形できる．

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & & \vdots \\ t_{n1} & \cdots & t_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

ここで，$t_{ij}, u_k (1 \leq i, j, k \leq n)$ は次の形をした項である．

$$t \equiv \Sigma_i a_i \cdot t + \Sigma_j b_j \quad \text{(a)}$$

$$t \equiv \Sigma_i a_i \cdot t + \Sigma_j b_j \cdot \delta + \Sigma_k c_k \quad \text{(b)}$$

$$\delta \quad \text{(c)}$$

$a_i, b_i, c_i \in = M \cup_{c_l \in C} \{\tilde{m}_i | m_i \in \mu(c_l)\}$ である．また，各 t は定義から再帰的に導かれる形をした項である．

この結果から，オブジェクト間通信の正当性に関して次が言える．

(1) 全ての t_{ij}, u_kが (a) か (c) の形のときは通信が正しく行われる．(2) ある t_{ij}, u_kが (b) の形を含み，かつ全てのδを含む項中に$\tilde{m}_i^* \cdot \ldots \cdot \delta$の形の列が出現するとき，通信路の記述が不完全である．(3) 上記 (1)，(2) 以外の場合，通信に誤りがある．

5 対話的な仕様獲得に関する考察

仕様書の記述の一貫性を維持するには，検証により正当性が証明できる厳密な記述が必要である．しかし，当初から全体に厳密な記述を要求することは設計者に大きな負担になる．これには，不完全な記述をもとに検証を行い，対話により段階的に必要な記述の追加を促せば，仕様記述作業の負担が軽減されると考えられる．

本研究で示した検証手法は，

1. 例記述の不完全な箇所を検出でき，その箇所を示して記述作業を促すことができる，
2. 仕様とプログラムの検証の際に対話的に必要な情報を獲得できるため，通信路の指定にあいまいな記述形式を採用できる，

などの利点があり，仕様書のわかりやすさと検証性の両立が期待できる．

6 おわりに

スパイラル型開発モデルの実施には，仕様書間およびプログラム記述との一貫性を保つことが肝要である．

我々は，仕様情報は記述の抽象度，機能・構造・振る舞い，要求実行例・仕様・プログラムの各側面を有するという観点から，総合的な検証の枠組みを提案している．本研究では，オブジェクトの振る舞いに着目して，検証を行う方法を示した．

まず，振る舞いに関する数学的なモデルとして ACP$^\tau$ を採用し，状態遷移図，メッセージフロー図，シーケンスチャート，拡張シーケンスチャート，C++プログラムの各記述がどのようにこのモデルで解釈されるかを検討した．

つぎに，このモデルに写像された情報を元にして，オブジェクトの振る舞いを要求実行例・仕様・プログラム相互間で検証する手法を示した．この検証手法は，対話により段階的に必要な記述を獲得でき，あいまいな記述を許容できる特長を持つ．

今後，ここで提案した検証手法を実装して有効性を実証していくとともに，残された記述側面についても検討する予定である．

本研究を進めるにあたりご支援頂いた，早馬常務取締役および篠原部長，清野室長に謝意を表します．

参考文献

[1] Barry, W. Boehm: "A Spriral Model of Software Development and Enhancement", IEEE Comput., Vol. 21, No. 5, pp. 61-72, (1988).

[2] Baeten, J.C.M., Weijland, W.P.:"Process Algebra", Cambridge Univ. Press., (1990).

[3] J.Rumbaugh, M.Blaha, et al.: "Object Oriented Modeling and Design", Prentice Hall, (1991).

[4] J.M. Vlissides and M.A. Linton: "Unidraw: A Framework for Building Domain-Specific Graphical Editors", ACM Trans. Inf. Syst., Vol. 8, No. 3, pp.237-268, (1990).