ソフトウェア工学 96－1
（1994．1．20）

高レベルペトリネットを用いたオブジェクト指向設計

元木 誠　中島 震

NEC C&C 研究所

〒216 川崎市宮前区宮崎 4-1-1

オブジェクト指向概念を導入した高レベル・ペトリネット (OhNET) を用いたオブジェクト指向設計のための形式モデル–Trinity– を提案する。本稿では、OhNET が動的モデルの記述のみならず静的モデルを統合して表現できる形式モデルであるため、各モデル間の整合性確認が容易になり、高レベル・ペトリネットのシミュレーション機能、および、到達グラフ解析機能を利用して設計全体の振舞いの妥当性の確認が可能になることを示す。さらに、検証記述と OhNET モデルで記述された設計仕様を組み合わせることで、オブジェクト指向フレームワークの参加オブジェクトの振舞いをデモンストレーションできるフレームワークの視覚的なドキュメントを提供できることを示す。

Object-Oriented Design based on High Level Petri Nets

Makoto Motoki　Shin Nakajima

C&C Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216

This paper proposes a formalism for object-oriented design, Trinity, which provides an object-oriented extension of a high level Petri net, OhNET, as the dynamic model and the object model of OMT as the static model. We show that Trinity can integrate object models and OhNET models into one OhNET model describing the entire design so that designers can check consistency and validity of the design with formal analysis methods for high-level Petri nets. This paper also shows a specification coupling OhNET design models and test specifications for validation can be seen as a visual document for OO frameworks which can demonstrate typical threads of the frameworks.

1 はじめに

オブジェクト指向ソフトウェアの開発を分析・設計段階から支援することを目的として、オブジェクト指向分析・設計方法論が数多く提案されている。これらの方法論は複数のダイアグラムを提供しており、設計者は、方法論が示すオブジェクトの識別方法とオブジェクト間関係の構築方法に従ってダイアグラムを作成することで、複数の観点からオブジェクト指向概念に基づいて対象の分析・設計を行なうことができる。

しかし、これらのダイアグラムにはいくつかの問題がある。

検証の困難さ： 各ダイアグラムは非形式的であり複数ダイアグラム間の対応関係は曖昧で、設計者の解釈に依存せざるをえない。したがって、各ダイアグラム間の整合性を確認したり、設計者の意図に対する設計の妥当性を確認することが困難である。

再利用のための設計理解の困難さ： 一般にオブジェクト指向ソフトウェアでは一つのまとまった機能は複数のオブジェクトによって実現されるため、設計を理解するためには、個々のオブジェクトの振舞いを独立に把握するだけでは不十分であり、相互に依存しあうオブジェクトの集団的な振舞いを把握する必要がある[8]。設計仕様は利用者に対してこれら集団的な振舞いを説明するドキュメントとなるべきであるが、現状のOOA/OOD方法論は集団的振舞いを明記する手段を提供できていない。

これらを解決するために、OMTのオブジェクトモデル、OhNETモデル、コントラクト記述からなるオブジェクト指向設計のための形式モデル –Trinity– を提案する。本稿では、OhNETが動的モデルの記述のみならず静的モデルを統合して表現できることから、各モデル間の整合性確認が容易であり、システム全体の妥当性の確認が可能であることを示す。さらに、検証記述とOhNETによる設計仕様記述を組み合わせることで、集団的振舞いの典型的なシーケンスをデモンストレーション表示できるオブジェクトの集団的振舞いの視覚的なドキュメントが提供可能となることについて述べる。

2 Trinity

Trinityは、静的モデルとしてOMT[9]のオブジェクトモデルを提供し、動的モデルとしてColoured Petri Nets[5]にカプセル化・継承・オブジェクト間通信の概念を表現する機構を追加した –OhNET[7]– を提供するオブジェクト指向設計のための形式モデルである。TrinityはOMTとは異なり、機能モデルとして独立したモデルを持ってはいない。クラス内の機能的な側面はOhNETが提供する内部記述言語(関数型言語)によって表現する。以下、ATM専用回線における可変容量サービスシステムを記述例としてOhNETの各モデルを説明する。

図1はシステムのオブジェクトモデルである[1]。ユーザには契約にしたがってサービスクラスで表された専用パス(仮想パス)が割り当てられており、パスの容量を随時変更できる可変容量パス(可変パスクラス)と予約によって容量を変更する予約パスが存在する。各パスはノードシステムを介して複数のリンクによって構成されている。図1では構成リンク・クラスが一つのパスを構成するリンク(リンク・クラス)を集約するクラスとなっている。

2.1 OhNET モデル

OhNETはクラスの振舞いを記述するクラス動的モデルとクラス間のイベント送受信を記述するクラス間通信モデル(OMTのイベントフロー図に相当)からなる。クラス動的モデルは、クラス単位のカプセル化を表す機構、継承関係を反映するネット構造、関連、集約関係などのクラス間関係を表現する機構をもつ高レベル・ペトリネットである。OhNETではオブジェクトの概念を明確に表現するためにイベント、状態を表すプレースを区別して表現し、トランジションについてもいくつかのプリミティブを用意している[7]。

図2はOhNETによる「リンク」クラスの動的モデル表現である。円形は状態を表し、矩形と円形のプレースがイベントを表す。各トランジションは状態の遷移を表し、ガード条件を満たし入力イベントプレースと状態プレースにトークンがある時に遷移する。

図2では省略されているが、各クラス動的モデルはクラスの属性を保持する抽象データプレースを1つ保持する。OhNETでは1つのオブジェクトはオブジェクトの現在状態を表す状態プレースのトークンと現在の属性値を表す抽象データプレースのトークンの2個のトークンによって表現される。抽象データプレースはクラス動的モデル内のすべてのトランジションの入力プレースおよび出力プレースとなっており遷移の際に値の参照・更新が自由にできる。

一方で、他のクラスのトランジションとは直接つながっておらず、他のクラスから抽象データプレースのトークンの値を参照する場合にはイベント送受信を介したアクセスを行なう必要がある。このようにして、

[1] クラス名および属性名の横にある()で括られた名前はOhNETとの統合の際に用いる形式名である。

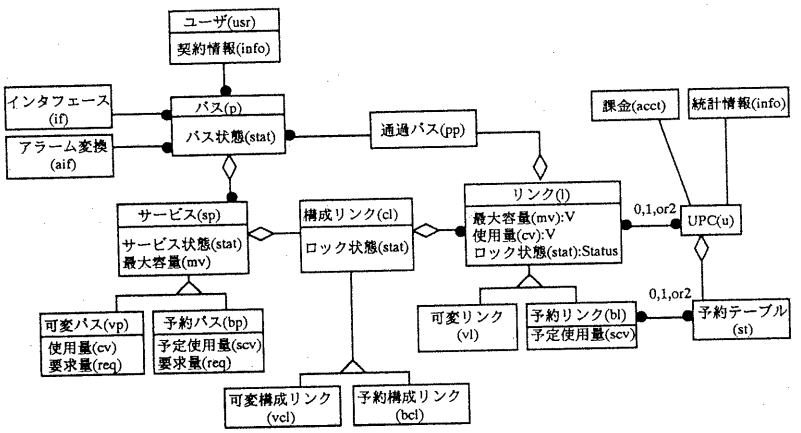

図 1: ATM サービスのオブジェクトモデル

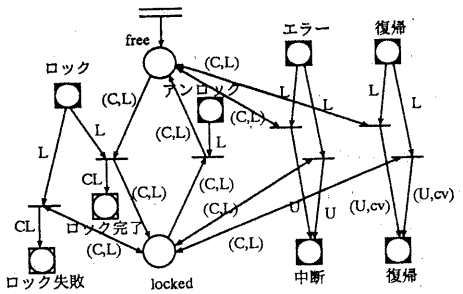

図 2: リンククラスのクラス動的モデル

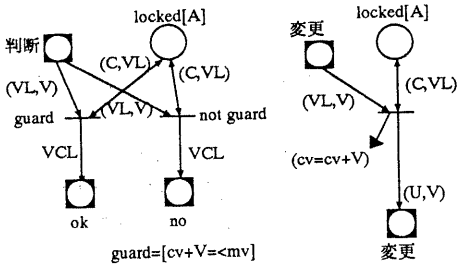

図 3: 可変リンククラスのクラス動的モデル

クラス動的モデルは内部データを外部クラスから隠蔽し、イベント受信インタフェースを通してのみデータを公開するカプセル化の概念を実現している。

上位クラスを継承する下位クラス OhNET モデルは、上位クラスとの差分の記述によって表現できる。差分と上位クラス記述の結合方法として、差分を記述したサブネットの特定のプレース ([A] とラベルづけする) と上位クラスの同一名のプレースとを融合するプレース融合 [4] と、上位クラスの 1 つのトランジションを差分を記述したネットで置換するトランジション置換 [4] の 2 通りを提供する。図 3は可変容量サービスのための機能の差分記述によって表した可変リンククラスの動的モデルである。

2.2 統合モデル

OhNET のプレースは一般の高レベル・ペトリネット同様に、データ型を持ちプレースに入るトークンのデータ構造を定義することができる。OhNET では、各クラスの抽象データプレースのデータ型を、オブジェクトモデルで定義された各クラスの属性の型、関連・集約関係を反映するデータ型によって定義することで、オブジェクトモデルのクラス定義、クラス間関係定義を OhNET モデルに統合する。

図 2のリンククラスの場合、抽象データプレースの型は自身のクラスのオブジェクト識別子を表す属性 $\{L:L\}$[2]、図 1で定義された属性 $\{mv:V, cv:V, stat:Status\}$、および、関連、集約関係にあるクラスのオブジェクト識別子を保持する属性 $\{CL : CL, U : U, PP:PP\}$ をすべて結合した、$\{L:L, mv:V, cv:V, stat:Status, CL:CL, U:U, PP:PP\}$ として定義される。

さらに、OhNET ではオブジェクトモデルで継承関係が定義されているクラス間に関しては、

[2]オブジェクト識別子の属性名、属性の型はともにクラス形式名で表す

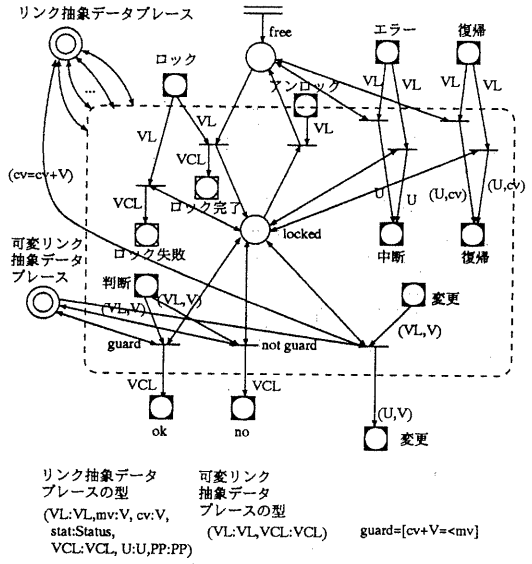

図 4: 可変リンククラス表現 (統合 OhNET モデル)

- 継承関係の上位クラスすべてのコピーを生成し、プレース融合、トランジション置換いずれかの方法にしたがってコピーと下位クラスネットとを融合する
- オブジェクト識別子、関連・集約関係にあるオブジェクト識別子を保持する属性を下位クラスの定義に合わせて修正する

ことによって継承関係を反映するネット構造に変換する。ネット変換によって得られる可変リンククラスの完全な動的モデルは図 4のようになる。

2.3 整合性確認

Trinity では設計の機能的な側面を動的モデル内で記述するため、設計の整合性とは静的モデルと動的モデルあるいは動的モデル間の構文的な無矛盾性を指す。上述のように、Trinity はオブジェクトモデルに記述されたクラス定義、クラス間関係、継承関係を OhNET モデルに組み込んで、静的モデル (オブジェクトモデル) と動的モデル (OhNET モデル) を統合した統合 OhNET モデルを生成することができる。

一方、OhNET モデルではトランジションにおいて参照可能な値は隣接するプレースのトークンが保持する値に限られており、違反する記述は構文エラーとして確認することができる。

この性質を利用して、Trinity ではシステム全体における属性の参照・更新要求と参照可能な属性の間の整合性、送信受信イベントの無矛盾性、属性・イベントの型の整合性などを統合 OhNET モデルの構文エラーとしてチェックできる。

3 集団的振舞いの妥当性確認

3.1 集団的振舞い

一般に一つのまとまった機能は複数のオブジェクトに分散して実現されるため、個々のオブジェクトの振舞いの妥当性を独立に確認するだけでは不十分であり、オブジェクトの集団的な振舞いの妥当性を確認する必要がある [8]。

ここで集団的振舞いとは、オブジェクト集団を「集団としてある機能を実現し、また、ある制約を維持するために依存関係を持つオブジェクトの集まり」と定義すると、「オブジェクト集団が機能の実現や制約の維持を実行する際の、オブジェクト集団に属するすべてのオブジェクトの振舞い」のことである。

3.2 解析手法

OhNET モデルは他の高レベル・ペトリネット同様に形式的な実行セマンティクスを持っており、モデルのシミュレーション、および、到達グラフ解析が可能である。Trinity では、前章で述べた統合 OhNET に対するシミュレーションと到達グラフ解析手法を提供することで、複数のクラスの集団的な振舞いの妥当性の確認を可能とする。

3.2.1 シミュレーション

設計者は、初期マーキングを与えてシミュレーションを行なうことで、設計仕様の振舞いをダイアグラム上のトークンの移動によって視覚的に確認することができる。シミュレーションの際には、トランジションを発火可能とする変数束縛の計算、アークにラベルづけされたアーク関数の計算によって属性値の変化も計算されており、状態遷移のみならず、実行にともなう属性値の変化もチェック可能である。

3.2.2 到達グラフ解析

到達グラフ解析とは、初期マーキングを起点ノードとして、ある到達可能なマーキングから発火可能なトランジション 1 個の発火によって到達可能となるマーキングをすべて有効枝で結んでできた到達グラフを用いて、対象システムの到達性、活性など種々の性質を解析する手法のことである。グラフ生成にかかる計算コ

ストはプレースとトランジションの数の指数オーダーであり、この手法は規模の大きいシステムの振舞い全体を確認する手段としては実用的ではない。

ところが、システムが提供する機能は多くの場合、ある限られたオブジェクト集団によって実現されている。したがって、到達グラフ解析は、ある機能の実現に関与するオブジェクトの集団的な振舞いの妥当性を網羅的に検証するためには有効な手段である。

3.3 到達グラフ解析による妥当性確認

集団的な振舞いの検証に到達グラフ解析を用いるためには、解析対象となる参加オブジェクト、初期マーキング、検証すべき不変表明や事前 / 事後条件などを記述する手段が必要である。ある機能の実現に参加するオブジェクト、振舞いの制約、不変表明、初期設定の仕様記述としてコントラクト [3][10] がある。Trinity では、このコントラクトによる記述をベースにした**検証記述**を検証のための仕様記述として提供する。設計者は、解析対象クラスと不変表明と初期マーキング[3]を定義した検証記述を与えることで、到達グラフ解析による妥当性確認ができる。

図 1の例の場合、パス容量を変更する機能は、パス自身およびパスを構成するリンクすべてにおいて容量変更の可否判断を行なってから容量変更の処理を行なう。ただしこの一連の処理は一度に複数要求される可能性があるためリンク全体をロックして行なわれる。したがって、パス容量変更という機能に参加する可変パス、可変構成リンク、可変リンククラスの各オブジェクトは次のような条件を充足すべきである。以下、「オブジェクト識別子.属性」は「オブジェクト識別子」を持つオブジェクトの「属性」の値を意味する[4]。

1) $\forall Vp \in VP : Vp$.使用量 $\leq Vp$.最大容量.

2) $\forall Vl \in VL : Vl$.使用量 $\leq Vl$.最大容量.

すなわち、参加する可変パス、可変リンクのすべてのオブジェクトは最大容量を超えた使用量を持ってはいけない。

3) $\exists Vp \in VP : Vp.Status == "check" \vee Vp.Status == "update" \Rightarrow \forall Vl \in (Vp.VCL).VL : Vl.Status == "locked"$.

ここで *Status* は OhNET モデルの状態プレース名を表す属性とする。すなわち、可変パスオブジェクトが check もしくは update 状態の時には対応する可変リンクオブジェクトはすべて locked 状態でなければならない[5]。

4) $(VP \leftarrow$ 容量変更 $(V)) \rightarrow \forall Vl \in (Vp.VCL).VL : (Vl.cv)' == Vl.cv + V$.

ここで $Obj \leftarrow E$ は Obj へのイベント E 送信を表すものとする。すなわち、「可変パスオブジェクト (Vp) への容量変更 (Change) イベント送信」という事前条件に対して、「すべての対応する可変リンクの使用量 (cv) の値の変化」という事後条件が満たされることを表す。

Trinity の検証記述では 1) ～ 4) を以下のように記述する。また、参加オブジェクトの初期状態、初期属性値からなる初期マーキングと、グラフ生成の際に考慮すべき外部イベントの記述として「instance」項目を持つ。

```
validation spec VpVclVl::[VP,VCL,VL] {
invariant
  forall Vp in VP: Vp.cv =< Vp.mv          …1)
  forall Vl in VL: Vl.cv =< Vl.mv          …2)
  foreach Vp in VP: Vp.Status == "check"
     or Vp.Status == "update" -->
       forall Vl in (Vp.VCL).VL:
         Vl.Status == "locked"             …3)
  Change(Vp,V) |-->
     forall Vl in (Vp.VCL).VL:
       (Vl.cv)' == Vl.cv+V                 …4)
instance
 ((VP=Vp,mv=150,cv=100,VCL=Vcl)
  (VCL=Vcl,L=Vl-1,Vl-2)
  (VL=Vl-1,mv=250,cv=80,VCL=Vcl)
  (VL=Vl-2,mv=200,cv=100,VCL=Vcl)
  Change(Vp,40))
 (   ...   )
}
```

Trinity は上記の検証記述に基づいて参加オブジェクト (この場合 VP,VCL,VL クラスの 4 つのオブジェクト) に関する初期マーキングからの到達グラフを生成する。続いて検証記述にしたがってグラフをを探索し、「forall P 形式」 (e.g.1) または 2)) に対しては P を満たさないノードを提示し、「P --> Q」 (e.g.3)) に対しては P を満たすノードのうち Q を満たさないノードを提示し、「P |--> Q」 (e.g.4)) に対しては

[3] 不変表明は [3] の invariant と同等であり、初期マーキングは [3] の instantiation の一具体例の表現となる。

[4] したがって、Vp.VCL は可変パスオブジェクト Vp が持つ VCL(可変構成リンク) クラス・オブジェクトの識別子、(Vp.VCL).VL はその可変構成・リンクオブジェクトが持っている VL(可変リンク) クラス・オブジェクトの識別子の集合を指す。

[5] この制約条件は酒井 [10] における状態間制約にあたる。

Pを満たすノードから到達可能なノードの中でQを満たすノードを提示することで、設計の検証条件違反を指摘する。

4 集団的振舞いの視覚化

オブジェクト集団は複数のクラスから構成されており、集団的振舞いを理解するためには、参加するクラス、クラス間の依存関係、各クラスのオブジェクトの振舞い、集団が満たす制約について知る必要がある。これらを説明するドキュメントとして、Trinityは前章で述べた検証記述に一部拡張を施した**コントラクト記述**と統合OhNETモデルを用いる。本稿では、集団的振舞いの記述例として、フレームワーク[6]の記述を例にとる。

4.1 フレームワーク

フレームワークとはクラスの集合として表現された、ある機能の全体あるいは一部を実現する再利用可能な設計要素[6]のことである。フレームワークはある機能を実現する集団的振舞いの骨組み[12]的な部分である。構成要素間には振舞いの依存関係が存在し、再利用するためには設計者はその振舞いの構造を理解する必要がある。

振舞いの記述については自然言語とダイアグラムによる記述[6]や形式的仕様言語による記述[3],[8],[10]も提案されているが、参加オブジェクト間の複雑なやりとりなどの振舞いの依存関係を表す場合は、シーケンスなどによる視覚的な表現の方がより理解しやすい[1]。Trinityは、コントラクト記述を前提条件と不変表明の記述手段として提供するとともに、フレームワークを構成する(包含する)オブジェクト集団の典型的な振舞いを、コントラクト記述に示された初期マーキングとOhNETモデルに基づいてシミュレートし、視覚的に表現する手段を提供する。

4.2 コントラクト記述

コントラクト記述におけるフレームワークの参加クラスと制約の記述は、検証記述の参加クラス定義と不変表明の定義をそのまま用いる。フレームワークの典型的な振舞いをシミュレートするため必要となる、典型的な振舞いの初期マーキングおよび関連する外部イベントの記述は「typical instance」項において検証記述の「instance」と同様の形式で記述する。

ただし、この「typical instance」記述は典型的な振舞いの一シーケンスの初期設定を表現した、フレームワークの前提条件の一つの十分条件にすぎない。コントラクト記述では、フレームワークの参加オブジェクトが初期設定時に満たすべき必要条件として参加オブジェクトの「peerオブジェクト[6]に関する条件」と「属性に関する条件」を「instantiation[7]」項によって記述する。

3.3の検証で用いた例のコントラクト記述は以下のようになる。「instantiation[8]」項は、可変パスオブジェクトが1個存在して1個の可変構成リンクオブジェクトをpeerオブジェクトとして持っており、この可変構成リンクオブジェクトは複数個の可変リンクオブジェクトをpeerオブジェクトとして持っていること、可変パス、可変リンクの属性値cvがmvの値を越えないことを表している。

contract VpVclVl::[VP,VCL,VL] {
invariant
(3.3と同一内容につき省略)
instantiation
$m : int$
$\exists_1 Vp : VP \bullet (\exists_1 Vcl : VCL \bullet$
$Vp.VCL = \{Vcl\} \wedge Vcl.VP = \{Vp\} \wedge$
$Vp.cv \leq Vp.mv \wedge$
$\exists_1 U : \mathrm{P}\ VL \bullet$
$\#U \leq m \wedge Vcl.VL = U \wedge$
$\forall Vl : VL | Vl \in U \bullet Vl.VCL = \{Vcl\} \wedge$
$Vl.cv \leq Vl.mv)$
typical instance

```
((VP=Vp,mv=150,cv=100,VCL=Vcl)
 (VCL=Vcl,L=Vl-1,Vl-2)
 (VL=Vl-1,mv=250,cv=80,VCL=Vcl)
 (VL=Vl-2,mv=200,cv=100,VCL=Vcl)
 Change(Vp,40))
```

}

4.3 フレームワークの視覚的表示

フレームワーク記述の表示として、Trinityはコントラクト記述のテキスト表示とともに、「typical instance」で定義された初期マーキングと統合OhNETモデルをもとにして、典型的な振舞いのシーケンスをシミュレータによってデモンストレーション実行しながら表示する。

振舞いのシミュレーションはOhNET記述上(図5のEditorウィンドウ)でトークン移動によって表現

[6] オブジェクト識別子を知っている関係にあるオブジェクト。

[7] [3]におけるinstantiationに相当する。

[8] Z記法[11]を用いて記述する。

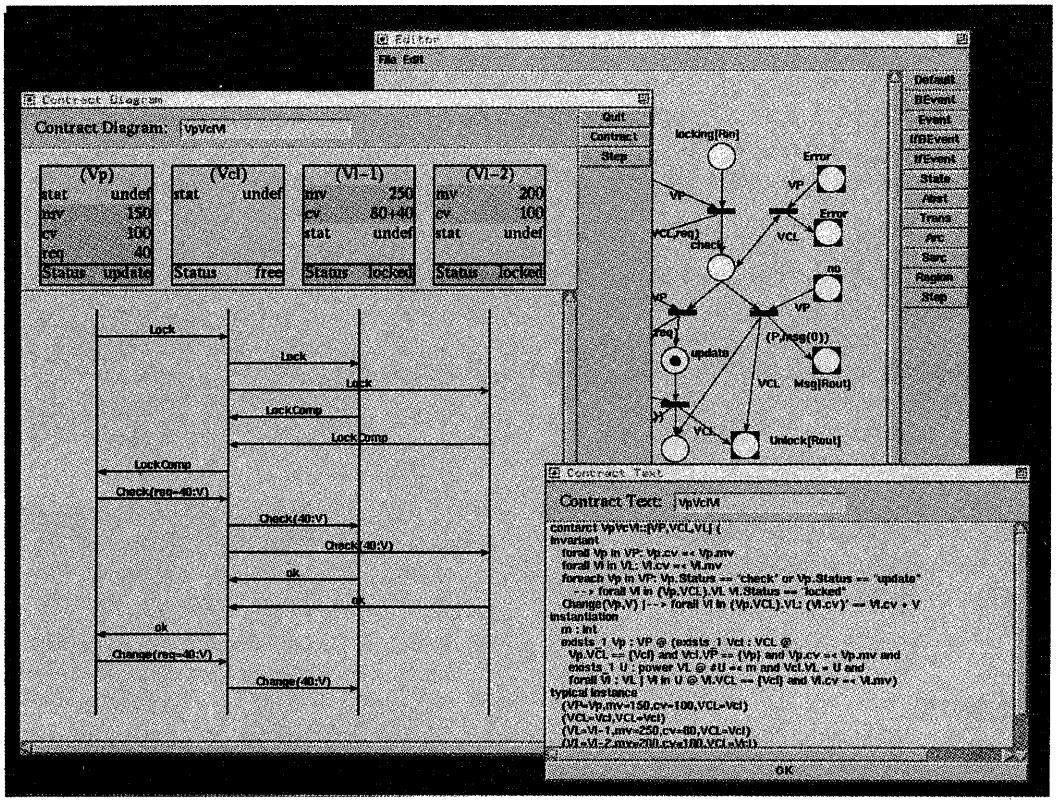

図 5: 容量変更処理フレームワーク記述の表示

できるが、フレームワークの参加オブジェクトとイベントやりとりをより明確に表現するために、 Trinity ではシーケンスチャート表現によって集団的な振舞いを提示する (図 5の Contract Diagram)。

シーケンスチャートは統合 OhNET モデルのステップ実行にともなって順次表示されていき、この間にチャート上部のインスタンスダイアグラムが実行に伴う各オブジェクトの属性値の変化を表示する。インバリアント記述に関連する属性と属性値はハイライトされており、フレームワークがどのような制御スレッドを持ち、振舞いの中でインバリアントがどのように充足されているかを視覚的に確認することが可能である。

設計者は、「instance」項を追加して新しい初期マーキングを設定することができる。新しいマーキングが満たすべき制約は「instantiation」項に記述されており、これを満たすマーキングを設定することで例外処理などを含めた典型的な例以外に参加オブジェクトのとりうるあらゆる振舞いをデモンストレーションによって視覚的に確認することが可能となる。

5 関連研究

Hayes ら [2] は形式的な動的モデルとして Objectchart を導入した OOA モデルを提案した。本研究と同様に、各モデルおよびモデル間の関係に形式的な定義を与え、モデル全体の整合性や妥当性確認を可能にしている。ただし振舞いの妥当性検証はオブジェクト内の 1 操作を実現する 1 イベントトレースを単位として行なわれ、 Trinity が提供するような「ある機能の実現に参加するオブジェクト群の集団的な振舞いを網羅的に検証する」手段はない。また、検証に用いたトレースおよび記述には集団的振舞いに関与する参加オブジェクトや、それらが満たすべき前提条件などは含まれておらず、フレームワークの記述としては不十分である。

Contract[3] は集団的な振舞いの参加オブジェクト、

それらの依存関係や性質を明確に記述でき、フレームワークの記述としては優れている。ただし、OODで必要となるシステムの静的側面の記述、個々のオブジェクトの振舞い記述の手段としては利用できない。これに対しTrinityはOOD記述モデルとしての側面とフレームワーク記述としての側面を合わせ持つモデルである。

RDD[12]はコラボレーション・グラフなど、オブジェクトの集団的振舞いを記述する手段を持ちフレームワークの記述が可能である。ただし、ダイアグラムは非形式的であり、動的な振舞いの記述を持たないため、妥当性検証などは困難であり、フレームワークの処理の流れを明示することができない。

フレームワークの記述としてJohnson[6]はパターン言語に基づくテキスト表現を提案している。記述の自由度は高く理解しやすいドキュメントの記述が可能であるが、非形式的であり設計仕様とドキュメントの間の一貫性を維持するのは困難である。またTrinityの記述のような振舞いのデモンストレーションは提供できない。

6 おわりに

本稿では、オブジェクト指向概念を追加した高レベル・ペトリネット(OhNET)とコントラクト記述を持つオブジェクト指向設計のための形式モデル(Trinity)を提案した。このモデルにおいては、設計の静的側面と動的側面の整合性確認が容易であり、振舞いの妥当性確認に高レベル・ペトリネットが提供する形式的解析が適用可能であることを述べた。さらに、検証記述を設計仕様の一部として残すことで、フレームワークの典型的な振舞いのデモンストレーションが可能なフレームワークのドキュメントを提供できることを示した。

本稿ではフレームワークの振舞いのシーケンスをシミュレーション実行によって動的に求める方法を提案したが、制限された構造の(直列トランジション融合を施すことによって、シーケンスに関与するすべてのトランジションにおける入力イベントプレースと出力イベントプレースの数が等しくなる)OhNETにおいてはシーケンスはS-インバリアントとしてネットの接続行列から静的に求めることが可能である。

現在のTrinityモデルでは最終的に得られる設計モデルの中でメソッドインタフェースは動的モデルの入出力イベントのシグナチャとして表現されているのみで、設計仕様としては不備がある。今後オブジェクトモデル等に明示的に表現する手段を検討していく予定である。

また、フレームワーク記述は参加オブジェクトの振舞いを詳細に立ち入らずに理解するための表現[1][6]であるべきであり、実行トレースをベースとする本方式の場合、適切な抽象度を維持するためには典型例や参加オブジェクトの選択に関する方法論の提供、トレースそのものの抽象化などを検討する必要がある。

参考文献

[1] R.J.A. Buhr and R.S. Casselman : Architectures With Pictures, in Proc. of OOPSLA'92, pp. 466–483 (1992)

[2] F. Hayes and D. Coleman : Coherent Models for Object-Oriented Analysis, in Proc. of OOPSLA'91, pp. 171–183 (1991)

[3] R. Helm, I.M. Holland and D. Gangopadhyay : Contracts: Specifying Behavioral Compositions in Object-Oriented Systems, in Proc. of ECOOP/OOPSLA '90, pp. 169–180 (1990)

[4] P. Huber, K. Jensen and R. M. Shapiro : Hierarchies in Coloured Petri Nets, in Advances in Petri Nets 1990, LNCS 483, pp. 313–341, Springer-Verlag (1990)

[5] K. Jensen : Coloured Petri Nets 1, Springer-Verlag (1992)

[6] R.E. Johnson : Documenting Frameworks using Patterns, in Proc. of OOPSLA '92, pp. 63–76 (1992)

[7] 元木 誠, 中島 震: オブジェクト指向設計のための高レベル・ペトリネット, 第10回日本ソフトウェア科学会大会, pp. 49–52 (1993)

[8] 中島 震: オブジェクトの集団的振舞いの設計, ソフトウェア工学研究会 95-6, pp. 39–46 (1993)

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F.Eddy and W. Lorensen : Object-Oriented Modeling and Design, Prentice Hall (1991)

[10] 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, 情報処理論学会論文誌, Vol.33, No.8, pp. 1052–1063 (1992)

[11] J.M. Spivey : The Z Notation A Reference Manual Second Edition, Prentice Hall (1992)

[12] R. Wirfs-Brock, B. Wilkerson and and L. Wiener : Designing Object-Oriented Software, Prentice Hall (1990)