修 士 論 文

組み込みシステム設計のための ObTS に基づく記述支援環境に関する研究

指導教官　片山卓也 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

野村 昌男

2000 年 2 月 15 日

要 旨

本稿では, オブジェクト指向方法論に基づく記述モデル ObTS とその仕様記述言語 ObCL を用いて, システムから機能の切り出し, 基本形への変形, 機能同士の合成という手法により, 再利用性の高いシステム開発方法と開発効率を向上する仕組みを提案する.

目 次

図 目 次

第 1 章

はじめに

機器に埋め込まれてそのシステム制御を担うために開発されたコンピュータシステムを機器組み込みシステム (単に, 組み込みシステム) と呼ぶ. 組み込みシステムは電子レンジやエアコンといった家電製品, 自動車のような身近にあるものから工業用ロボット, 飛行機の管制制御システムのような大規模なものまで, 様々な用途に用いられている.

現在, 組み込みシステムに用いられる電子デバイスの性能向上やその利用分野の拡大に伴い, 機器の複雑化は避けられない状況にある. その上, 消費者のニーズに応えるためには, 新製品を次々と市場に送り込まなければならず, そのため企業間での競争力を維持するためには製品寿命の短さをシステムの開発時間の短縮で補う必要がある. このような要求に対応するため, 組み込みシステムの開発者からは設計やコードの再利用, 再構築を促進する仕組みや, 仕様を記述し検証することのできる科学的アプローチが求められている.

組み込みシステムの開発において, 状態遷移図や状態遷移表はよく利用されている. 状態遷移図ではシステムの動作を状態として表現し, 列挙可能なすべての系列を状態遷移として記述することで, システムの制御構造を表現することができるのでシステムを開発する上で有効な手段となっている.

システムを記述する上で重要とされる状態遷移図は, 大規模なシステムを記述する際に, 平面的なので可読性が悪くなったり, ある局面に関する状態をトレースするのが困難になるなどの問題を生じてしまう. そのため, 大規模なシステムに機能追加や拡張を試みる際, それが単純な変更であっても, その状態遷移図の複雑さのために作業が困難になってしまう場合がある. しかし, システムを構成する機能ごとの状態遷移図に対しての変更はシステム全体の動作を強く意識して変更を行なわなくてもよい. ゆえに, システム設計者が機能ごとに行なった変更をシステム全体に反映させることのできるシステム設計方法があれば開発効率の向上が望めると考えた.

本研究では, 以上のような考えを基に状態遷移図を用いた効率の良い設計を行なうための手法を提案するとともに, ObTS[1] に基づく分析/設計を支援する環境を用いて実用規模のシステムを記述する際に ObTS が有効であるということを示す.

状態遷移図を記述する計算モデルとして広く採用されているものに, オブジェクト指向方法論の動的モデル Statecharts[7] がある. Statecharts では階層構造, 並行性, ブロードキャスト通信を状態遷移図に持たせて拡張した記述モデルで, システムを記述するときに状態数と遷移数が爆発的に増加することを押えるのに成功している. ObTS は Statecharts に基づく記述モデルとして提案されており, オブジェクトモデルと動的モデルの構造に関連を持たせ, オブジェクトの階層構造でシステムを記述するという特徴を有している. また ObTS には計算機による支援環境として仕様記述言語 ObCL[4] が用意されている. ObCL は規模の大きいシステムを記述するために, クラス/クラスの継承/フィールド/イベントクラスなどの特性を持つマクロ言語として設計されている. ObTS にはシミュレーション環境として ObML が用意されている. ObML は Standard ML 環境上に構築されており, ObCL で記述されたシステムを ObCL コンバータにより ML コードに変換することで, システムのシミュレートを行なうことができる. またテストスクリプトを ML 言語で記述することができるので, システムに対して柔軟なテストを行なうことができる.

以下に本論文の構成を示す. 第 2 章では本研究の背景を述べる. 第 3 章では本研究の基礎として重要である ObTS/ObCL/ObML について述べる. 第 4 章では状態遷移図の再利用性の高い記述方法の提案とその手法を述べる. また, 提案した記述方法を適用する記述例として架空の ATM を取り上げ, その動的モデルを ObTS/ObCL を用いて記述する例を示す. 第 5 章では第 4 章で提案した再利用法に基づき, 実用規模の組み込みシステムである IrLAP(Infra-red Link Access Protocol) を記述することでその有効性を実証する. 第 6, 7 章でまとめ及び今後の検討を述べる. 付録は第 3 章で提案した再利用法に基づき ObCL で記述した IrLAP のコードである.

第 2 章

背景

本章では, 本研究に関連する事項について簡単に説明する.

2.1 機器組み込みシステム

機器組み込みシステムとは機器に組み込まれてその制御を行なうものをいう. 従来, 機器の付属的な存在として扱われていたが, 最近では組み込み機器自体の高機能化やデジタル化が進み, 製品に占める割合が徐々に高まってきている.

組み込みシステムが利用されている製品として

- 電子レンジやエアコンなどの家電製品
- コピー機や FAX などの OA 機器
- 自動車や電車などの機器制御
- 工業用ロボットや生産ライン監視装置など

が挙げられる.

2.2 Statecharts

オブジェクト指向方法論に基づく動的モデルを記述するために広く利用されているものに Statecharts がある. 原始的な状態遷移図では記述対象のシステムの規模が複雑になってくると, 平面的であるために状態数や遷移数が爆発的に増加してしまう. Statecharts で

はこれを改善するために状態の階層化/並行性/ブロードキャスト通信などを状態遷移図に持たせている.

しかし, Statechart ではデータは大域変数のみで階層化されていないために大規模なシステム記述においてデータに関する可読性が低下してしまう. また, 状態の階層化ではオブジェクトモデルの構造との対応が取られていないため, Statecharts の状態とそれに関わるオブジェクトやオブジェクトが担う状態遷移規則が明確ではないという欠点がある.

2.3 ObTS/ObCL/ObML

2.3.1 ObTS

伊藤 [1] が提案した ObTS は Statecharts の計算モデルに基づく記述モデルとして, 記述対象のシステム構造をオブジェクトの階層構造でモデル化する記述方法で, オブジェクトごとに状態遷移図とデータを持たせることで局所性を強めている.

ObTS では, オブジェクトは属性と状態遷移図を持っている. オブジェクトは動作の委譲のために内部オブジェクトを持つことができる. 内部オブジェクトも同様に属性と状態遷移図を持ち, 階層的に定義できる. オブジェクトが最初に実行する状態遷移を初期遷移とする.

2.3.2 ObCL

久保秋 [4] は, ObTS モデルに基づく記述体系として仕様記述言語 ObCL を考案した.

ObCL ではオブジェクトクラス/イベントクラス/フィールドクラス/属性クラスの集まりと一つのシステム記述によりシステムが表現されている. オブジェクトクラス, イベントクラスはそれぞれのオブジェクトとイベントの記述のためのものである.

2.3.3 ObML

ObML は Standard ML 上に ObTS 本体とそのための補助関数を組み込んだシミュレータである. ObCL で記述されたシステムは ObCL コンバータにより Standard ML コードとして生成され, その ML コードを ObML 上で読み込み, システムのシミュレートを行なうことができる.

2.4 ObTS/ObCL/ObML を研究の基礎とした理由

第 1 章で述べたように, 組み込みシステムの設計では状態遷移図や状態遷移表がよく利用されており, このことはオブジェクト指向分析/設計方法論を用いたシステム開発において動的モデルが有効であることを示している.

ObTS/ObCL/ObML では以下に示すような有用な特徴を持っている.

- オブジェクトの階層構造と並行動作を用いて実用規模のシステムを効果的に記述することが可能である.
- 組み込みシステムを記述する際に状態遷移図を利用できるので、設計するシステムの動作がわかりやすく便利である.
- ObTS/ObCL で記述したシステムを ObML 上で対話的にシミュレーションし, システムの仕様を検討しながら記述することができる.
- Standard ML コードで記述されたテストスクリプトを用意することで, システムに対して柔軟なテストを行なうことができる.

これらの特徴によって状態遷移図を用いてシステムを効果的に記述することが期待できるので, ObTS/ObCL/ObML を本研究の基礎とした.

第 3 章

ObTS/ObCL/ObML

本章では, 本研究の基礎である ObTS/ObCL/ObML について説明する.

3.1 オブジェクト指向仕様記述モデル ObTS

ObTS は記述対象のシステムの構造をオブジェクトの階層構造で表し, システムの動作を各オブジェクトの状態遷移, 関数的な属性計算, 属性付きイベントのブロードキャスト通信によってモデル化している.

ObTS では, オブジェクトは属性と状態遷移図を持っており, オブジェクトは動作の委譲のために内部オブジェクトを持つことが出来る. 内部オブジェクトも属性と状態遷移図を持つオブジェクトであり, 階層的に定義できる.

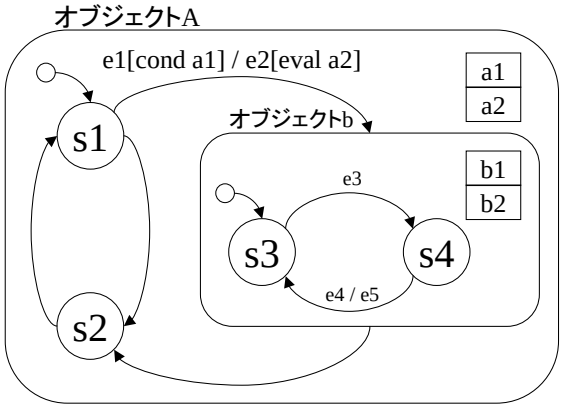

図 3.1: ObTS によるオブジェクトの構成図

図 3.1 は ObTS 図の記述例である．オブジェクトを角の丸い四角, 状態を円, 状態遷移を矢印, 初期遷移を黒丸からの矢印, 属性を四角として記述している．状態遷移を示す弧に遷移の入出力イベント/遷移条件/属性計算式を記述する．オブジェクト A は内部オブジェクトとしてオブジェクト b を持つ.

3.2 ObTS のための仕様記述言語 ObCL

前節の ObTS は計算モデルとして与えられているため, 具体的なシステムを記述できるように構文を与える必要がある．そのため久保秋 [4] は, ObTS モデルに基づく仕様記述言語 ObCL を用意した.

ObCL で記述したシステムは ObTS モデルに基づくシステム記述に変換可能であり, ObTS モデルにシステム記述のための構文, およびオブジェクトクラス, イベントクラス, フィールドクラス単位での記述の再利用や実用規模のシステムを記述するための特性を与えるマクロ言語的なものである.

ObTS 図と ObCL コードの対応関係を図 3.2 に示す.

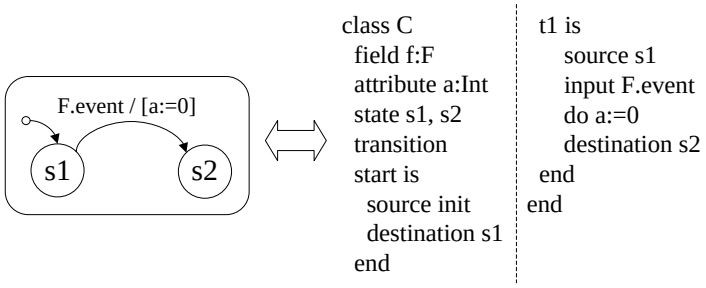

図 3.2: ObTS 図と ObCL コードとの関係

ObTS 図は ObCL 上でオブジェクトクラス (class) として記述される．フィールド (field) はイベント通信路で, イベントはフィールドを介してのみ伝達される．attribute で属性, state で状態を定義する．transition 以下は遷移規則で, ここでは規則 start は初期遷移, 規則 t1 は状態 s1 から状態 s2 への遷移規則を示している.

3.3 ObTS シミュレーション環境 ObML

ObML は関数型言語 Standard ML に構築された ObTS シミュレーション環境である. シミュレーションは, ObTS モデルに従ってステップ動作により実行される. シミュレータはシステムコンフィギュレーションと呼ばれる各ステップごとのオブジェクトの状態とイベントに関するスナップショットをシミュレーション結果として返す. また, 仕様確認のために記述したシステムに対するテストを行なう場合, テストスクリプトを ML 関数で与えることができる.

以上, ObTS/ObCL/ObML を用いることにより, システムを記述してシミュレーションするというサイクルで仕様を検討しながらの設計が可能となる.

第 4 章

ObTS 図による再利用性の高い設計方法

本章では, ObTS 図を用いて再利用性の高いシステム設計の方法を提案する.

4.1 ObTS 図を用いたシステム開発

ObTS 図は状態遷移図と属性, イベントを含んでいる. ObTS 図の再利用性や拡張性, 可読性を向上するための有効な手段の一つとして, システムから機能ごとの状態遷移図の切り出し, 基本形と呼ぶ形への変形, 機能同士の合成という段階的設計方法を提案する. ただし, 合成する際に個々の機能に含まれる属性やイベントは同じものとみなす.

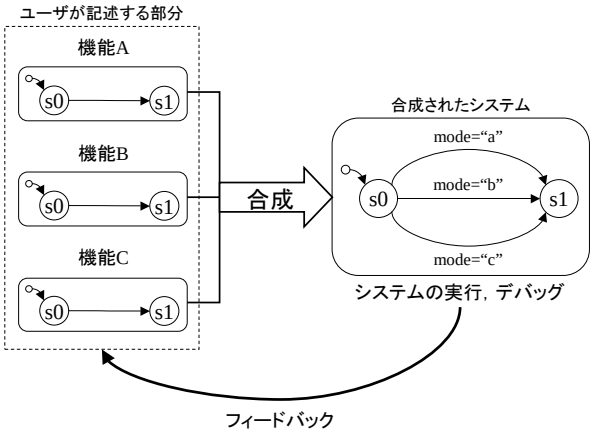

図 4.1: 本研究で提案するシステム設計のサイクル

1. 機能の切り出し
   システム全体の状態遷移図から, そのシステムを構成している機能を表す状態遷移図を切り出す.
2. 基本形への変形
   状態遷移図の可読性を高くするために各機能の状態遷移図を共通な形に変形する.
3. 機能同士の合成
   機能同士を合成し, 一つのシステムとしてまとめる.

まず, システム全体を示す状態遷移図から機能の動作を表す状態遷移図を切り出す. これにより, 機能ごとの設計やテストを可能にする. 次に, 基本形と呼ばれる形に機能の動作を表す状態遷移図を変形する. 基本形を導入した狙いは, システムの可読性の向上と機能ごとの変更箇所を明確にすることである. 最後に, 機能同士を同じ状態名の所で合成し一つのシステムとする.

設計者は機能ごとにシステムの設計を行なうことができ, 合成されたシステム記述に対して変更を加える必要はない. このためシステム全体の動作を把握しなくても各機能に対しての設計に集中することができる. ObTS/ObCL でシステムを記述することで, ObML を用いて各機能ごと実行・デバック・テストをすることができる. さらに合成された後のシステムも実行・デバックをすることができる. また, 合成後のシステムではなく各機能へテスト結果をフィードバックさせることで, 設計者は機能ごとの設計のみ行なえばよいので開発効率が上がる. この設計方法により, システムの可読性を向上させることができ設計者の意図した設計を行なうことが期待できる.

4.2 状態遷移図の切り出し/ 変形/ 合成

本節では本研究で提案する状態遷移図の切り出し/変形/合成方法の具体的な説明をする.

4.2.1 機能の切り出し

システムの設計において, 機能追加や拡張を行なわなければならない箇所が明確であれば変更は容易になる. また, システム中の特定の機能だけをシステムとは独立に実行させたり, テストを行なうことができる仕組みがあると, システム全体に対して行なうよりも短時間で済み, 結果, 開発効率の向上が望める.

そこで, 各機能の動作を示している状態遷移図をシステムから切り出すという作業を行なう.

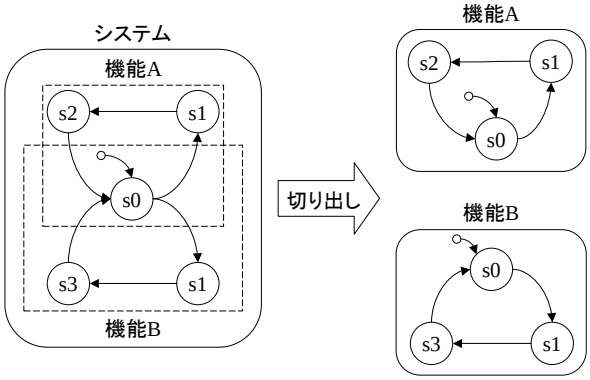

図 4.2: システムからの機能切り出し

図 4.2 において左図のシステムから機能 A, B を切り出す.

このシステムにおいて, 各機能の動作は

- 機能 A: 初期状態 `s0` → 状態 `s1` → 機能 A の動作を表す状態 `s2` → 初期状態 `s0`
- 機能 B: 初期状態 `s0` → 状態 `s1` → 機能 B の動作を表す状態 `s3` → 初期状態 `s0`

というような状態遷移となっているので, 右図のように機能ごとの状態遷移図としてシステムから切り出す. この作業により, システム全体としてではなく各機能として独立に実行やテストを行なうことが可能となる.

機能を切り出す際の条件は記述するシステムの仕様にもよるが, 一般的に組み込みシステムにおける機能では, おおむね

- 各機能の初期状態 → 各機能の実行状態 → 各機能の終了状態 (初期状態)

という状態遷移をしている場合が多い.

また, 仕様によっては初めから機能ごとの状態遷移図 (状態遷移表) で記述されていることもある. そのときは, この切り出しの作業は必要なく, 次の基本形への変形からシステム設計を行なえばよい.

4.2.2 基本形への変形

次に各機能ごとの状態遷移図を基本形と呼ばれる形に変形する. 基本形とは, 記述対象のシステム全体の動作を抽象的な状態遷移図で示したものである. つまり, 基本形はシステムの動きを簡単な形に表したものなので, システム全体の大まかな動きを把握するのにも役に立つ.

また, 各機能ごとの状態遷移図はこの基本形に則した形に変形できるものとする.

以下に, 基本形へ変形する手順や規則を例をもって説明する.

1) 基本形への変形: 例 1

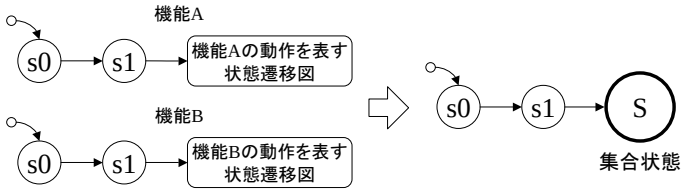

図 4.3: 基本形への変形 1

各機能の状態遷移が状態 s0 から状態 s1 まで共通で, その後の状態遷移が各機能で異なっている場合, 基本形は

- 状態 s0 → 状態 s1 → 各機能の動作を示す状態 (集合状態)S

とする. ここで状態 S とは機能の動作を表す状態で, その機能固有の動作を示す状態遷移図の集合である. 以後, そのような状態遷移の集合を一つの状態とみなしたものを集合状態という.

2) 基本形への変形: 例 2

各機能の状態遷移が状態 s0 と状態 s1 で共通で, 状態 s0 と状態 s1 の間の状態遷移が各機能で異なっている場合, 基本形は

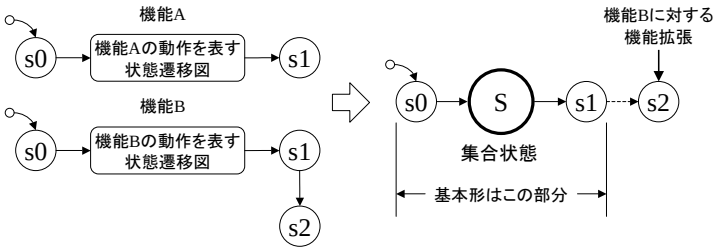

図 4.4: 基本形への変形 2

- 状態 s0 → 集合状態 S → 状態 s1

とする.

左図の機能 B には状態 s1 の後の状態遷移として状態 s2 があるが, これは機能 B に対しての機能拡張と考えるので基本形には含まないとする.

3) 基本形への変形: 例 3

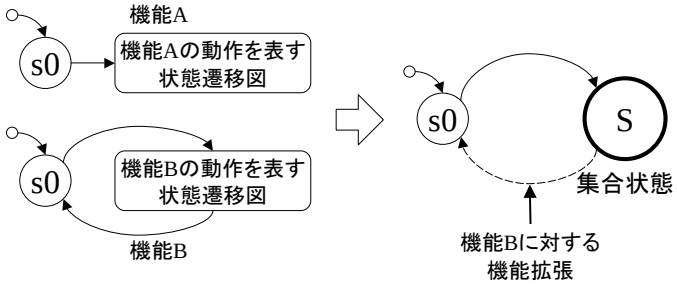

図 4.5: 基本形への変形 3

各機能の状態遷移が状態 s0 のみ共通で, その後の状態遷移が各機能で異なっている場合, 基本形は例 1 と同様に

- 状態 s0 → 集合状態 S

とする.

左図の機能 B には機能 B の動作を示す状態遷移図から状態 s1 への遷移があるが, これも例 2 と同様, 機能 B に対しての機能拡張とみなせるので基本形には含まないとする.

各機能を基本形に変形するときの方法として, 集合状態をどのように決めるかがある. 本研究では, 各機能を示す状態遷移図の中から同じ状態名に注目し, それを基準にして基本形を作成している. 集合状態の範囲を大きく決めておけば, システム全体の動作も大まかに把握することができるし, 機能に新たな動作を示す状態を追加する際にもその変更箇所を比較的容易に見つけることができる. ゆえに, 基本形を作成することはシステムの可読性や拡張性の向上につながる. また, システムの設計段階において, 始めは集合状態の範囲を大きく決めておき, システムの詳細化が進むにつれて基本形も詳細化していくという設計方法も考えられる.

以上より, 図 4.2 の各機能は上記の基本形の変形例を用いることにより, 図 4.3 のように変形することができる.

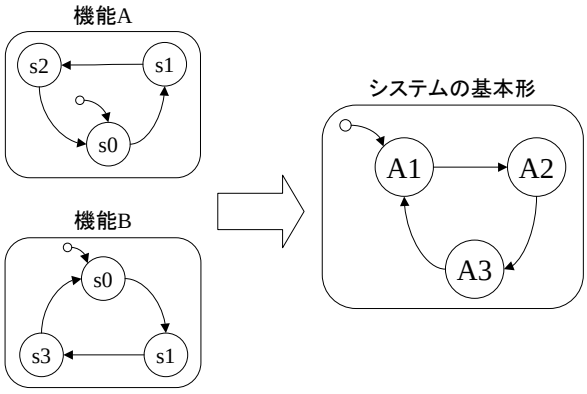

図 4.6: 基本形への変形

また, 基本形に変形することが難しい機能をシステムに追加する場合の設計方法は後節で示す.

4.2.3 機能同士の合成

機能の動作を示す状態遷移図を基本形に変形する利点は, システムの可読性, 拡張性の向上であることは前節において説明した.

もう一つの利点は, 基本形に変形した機能同士を合成し再び一つのシステムにまとめることにある.

機能ごとにシステムから切り出すことで, 各機能に対し実行/テスト/デバックを行なう

ことができる. しかし, システム全体としての実行やテストも重要である. そこで, 切り出された機能同士を合成し, 再び一つのシステムとして記述することで

- 各機能における実行/テスト/デバック → システム全体として実行/テスト/デバック → 各機能に結果をフィードバック

というシステム設計方法を実現する.

合成する時の注意点は, 本研究で提案した設計方法を適用したシステムと適用しなかったシステムとで実行結果が変わらないようにすることである. 本節で提案する合成方法は, 基本形に変形した機能同士を同じ状態名のところで重ね合わせるという手法であるが, そのまま合成すると, 一つの状態からシステムが保有している機能の数だけ遷移が出てしまうので, 遷移先が非決定的でシステムを実行するたびに結果が変わってしまう.

そこで機能ごとに属性を追加することで遷移先を決定的にする. この属性は機能やシステム全体の動作や実行結果には全く影響を与えないものである.

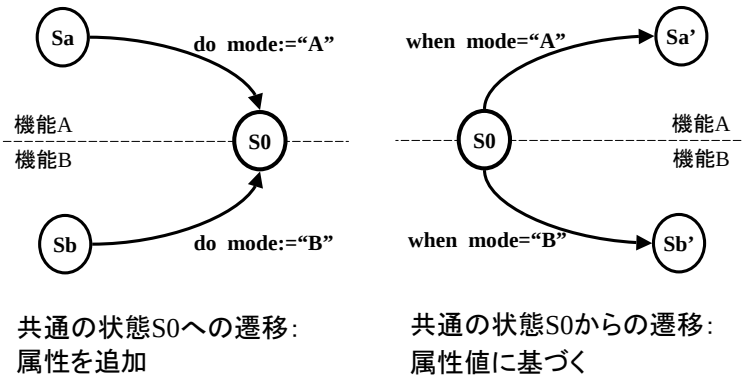

図 4.7: 合成の規則

共通の状態へ各機能からの遷移がある場合, この例では `mode` という属性に値を代入する. 共通の状態から各機能への遷移は, `mode` の値に依存する.

この合成の規則を適用することにより, 図 4.3 は以下のように合成することができる.

この例では, 属性 `mode` の値が `A` であるとき, システムは機能 A の動作をする, 値が `B` であるときは機能 B の動作をする.

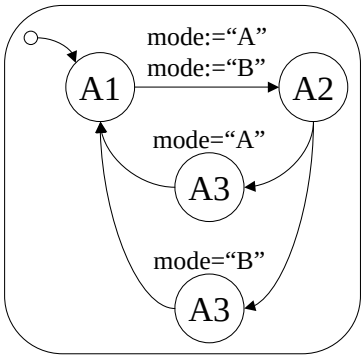

図 4.8: 機能同士の合成

4.3 テスト

前節で提案したシステム設計方法を用いてのテストについて説明する.

状態遷移図の切り出し/変形/合成方法を用いれば, 機能ごとのテストや合成後のシステムへのテストを行なうことができるようになるので, システムの開発効率が上がる.

ObML シミュレータでは, ステップ実行によるテストとテストスクリプトによるテストを行なうことができる.

ステップ実行によるテストは ObTS モデルでのステップ動作にしたがって進行する. シミュレータと対話的にテストをすることができるので, 動作確認をしながら仕様を検討することができる.

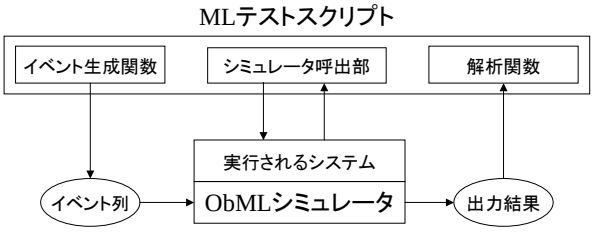

図 4.9: テストスクリプトの構造

テストスクリプトによるテストは, システムコンフィギュレーション (各ステップごとのオブジェクトの動作状態とイベントに関するスナップショットのリスト) を ML 言語スクリプトとして記述したものである. このスクリプトを用いることでシステムに対して連

続的にテストをすることができる.

図 4.9 は ML 言語で記述されたテストスクリプトの構造を示している. 入力イベント列を発生させるイベント出力関数と, そのイベントによりシミュレータが出力した結果を解析する解析関数はともに ML 言語で記述されている.

設計者はテストしたいシステムに対して, 入力イベントとそれに対応する出力イベントをテストスクリプトに記述することで, 設計者の意図したイベントごとの動作を柔軟にテストすることができる.

このテストスクリプトをシステムから切り出だされた各機能に対して用意することにより, 機能別にテストをすることが容易になる.

さらに, 各機能のテストスクリプトを再利用することで合成後のシステムに対するテストスクリプトも簡単に構築することができる. ゆえに, 合成後のシステムに対しても柔軟なテストが行なえる.

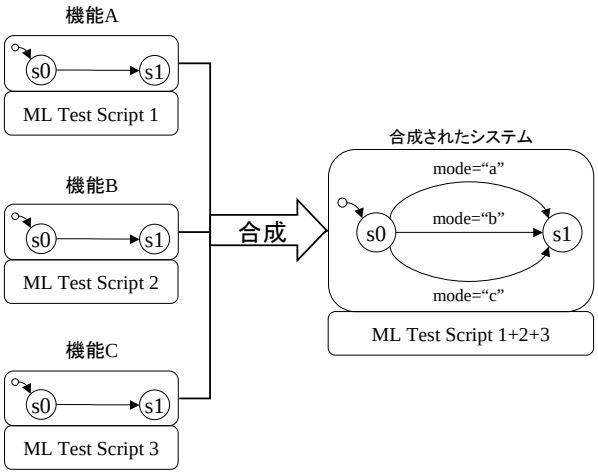

図 4.10: テストスクリプトを用いたテスト方法

4.4 切り出し/変形/合成方法に基づく記述 (例題:ATM)

本節では, ATM を設計する際に前節で提案した手法を適用した例を示す.

4.4.1 ATM の仕様

今回想定した ATM は以下のような仕様である.

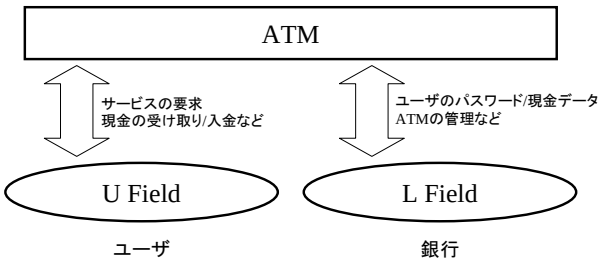

図 4.11: ATM とフィールド

- ATM のサービスは
  - 「引き出し (イベント名:`getmoney`)」
  - 「預り入れ (イベント名:`putmoney`)」
  - 「通帳記入 (イベント名:`write`)」

  である.
- パスワードの間違いは 2 回まで有効で, 3 回目では初期状態へ戻る.
- ユーザーの預金残高とパスワードは銀行が把握しているものとして, ATM 自身では管理をしない.
- ATM 自身が持っている金額は 100 万円とする.

ここでユーザと ATM 間のイベント通信路であるフィールドを `U`, ATM と銀行間のイベント通信路であるフィールドを `L` とおく. この仕様を基に ObTS で記述したシステムを図 4.11 に示す.

各サービスの状態遷移は

- 引き出し `U.getmoney`
  サービス要求 → カードまたは通帳挿入 → パスワード入力 (入力ミスで初期状態へ) → 金額入力 → 金とカードまたは通帳を排出し, 初期状態へ

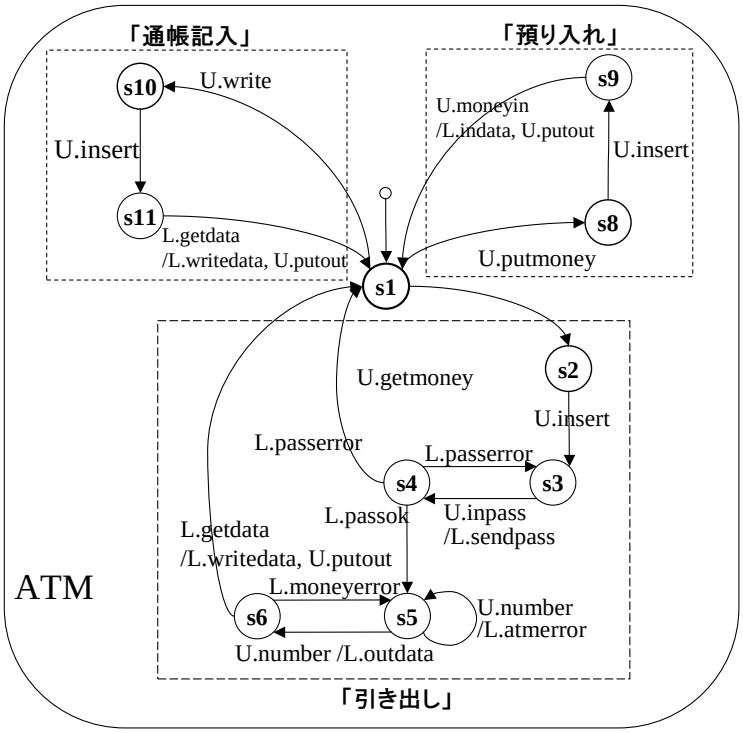

図 4.12: ATM の状態遷移図

- 預け入れ `U.putmoney`
  サービス要求 → カードまたは通帳挿入 → 入金後, カードまたは通帳を排出し, 初期状態へ
- 通帳記入 `U.write`
  サービス要求 → カードまたは通帳挿入 → 残高記入後, カードまたは通帳を排出し, 初期状態へ

となっている.

ここで, ATM の引き出しサービスの遷移規則を ObCL で記述したコードを示す.

```
  --- Get Money Sequence(引き出しサービス)
  t1a is
    source s1
    input U.getmoney --- 引き出しサービス要求
    destination s2
  end
  t2 is
    source s2
    input U.insert --- カードを挿入
    destination s3
  end
  t3 is
    source s3
    input U.inpass --- パスワードを入力
    do a:=a-1;
        L.sendpass.pass:=U.inpass.pass
    destination s4
    output {L.sendpass}  --- 銀行へパスワードを送信
  end
  t4a is
    source s4
    input L.passerror --- もしパスワードを間違えたら
    when a>0
    do D.string.msg:="Password Error! Try again."
    destination s3
    output {D.string}
  end
  t4b is
    source s4
    input L.passerror --- もし三回パスワードを間違えたら
    when a=0
    do a:=3;
       D.string.msg:="Your password fault! Try again, first."
    destination s1
    output {D.string}
  end
  t4c is
    source s4
    input L.passok
    do D.string.msg:="Your password OK!"
    destination s5
    output {D.string}
  end
  t5a is
    source s5
    input U.number --- 引き出す金額を入力
    when c<U.number.val  --- ATM の Capacity 以上の金額を入力したら
    do D.string.msg:="ATM Capacity Error! Try again."
    destination s5
    output {L.atmerror, D.string}
  end
  t5b is
    source s5
    input U.number --- 引き出す金額を入力
    when c>U.number.val or c=U.number.val
    do b:=U.number.val;
L.outdata.val:=U.number.val
    destination s6
    output {L.outdata} --- 銀行へ金額データを送信
  end
  t6a is
    source s6
```

```
    input L.moneyerror --- 銀行からの預金残高エラーを受信
    do b:=0;
       D.string.msg:="You don't have such money. Try again."
    destination s5
    output {D.string}
  end
  t6b is
    source s6
    input L.moneyout --- 銀行から引き出し許可を受信
    do c:=c-b; b:=0; a:=3;  --- 初期化
U.putout.val:=b
    destination s1
    output {U.putout} --- カード or 通帳を返却
  end
```

4.4.2 本研究の手法による状態遷移図の変形

各サービスは図のように

- 各サービス要求 → カード/通帳挿入 → サービス実行

といった大まかな流れで状態遷移している.

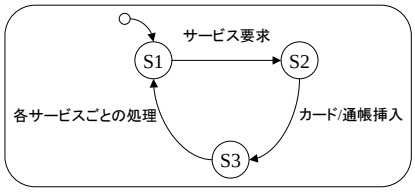

図 4.13: ATM の基本形

よって上記の動作を基本形とし, 各機能ごとの状態遷移図を以下の手順により変形する.

[状態遷移図の変形の手順]

(1) 状態遷移図を各サービスごと (引き出し, 預り入れ, 通帳記入) に切り出す.

(2) 各サービスの動作を示す状態遷移図では, 初期状態 S1 から S2 への遷移はすべての状態遷移で等しいので, 前章に従って基本形に変形する.

(3) 基本形に変形した状態遷移図を以下に従って合成する.

a) 属性の導入
この例では, 各サービスが常に U.insert(カード/通帳挿入) を伴っているので, この部分を合成するときに遷移先の制御を行なうため, 新たな属性 (この例では'e') を各サービスの状態遷移に追加する.
この属性は, 合成され各状態遷移のうちどれが実行されるのかをその値によって決定するもので, ATM の金額データなどを格納している他の属性やイベントには影響を及ぼさない.

b) 集合状態
各サービスのうち, 引き出し (U.getmoney) サービスでは図 4.13 の状態 S3 に当たる部分が複数存在する. これら (状態 s3, 状態 s4, 状態 s5, 状態 s6) を状態 S3 の内部状態とみなし, 一つの状態としてまとめる.

(4) 以上より初期状態 S1 を始点としてすべての状態遷移を重ね合わせる.

これにより図 4-11 の状態遷移図を変形した図を図 4-13 に示す.

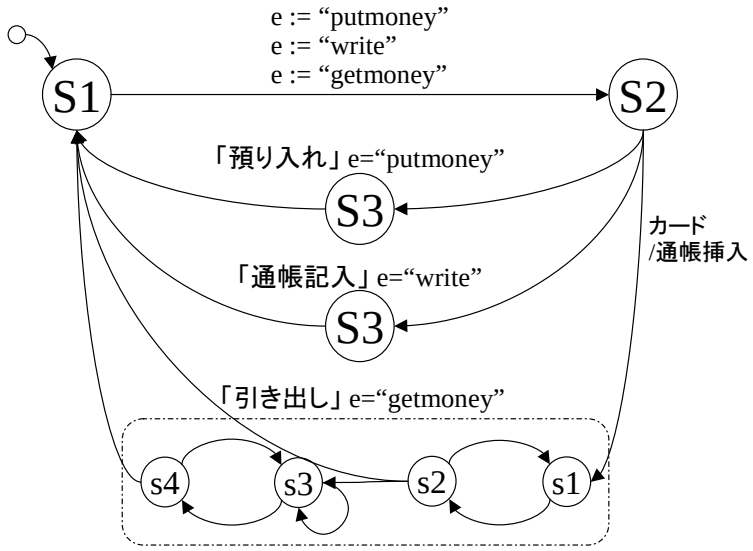

図 4.14: 切り出し/変形/合成方法を適用した ATM

4.4.3 機能追加

本節では切り出し/変形/合成方法によって構築されたシステムに対しての機能追加の例を示す.

この例では ATM に「残高照会」という新たなサービスを追加することを考える.

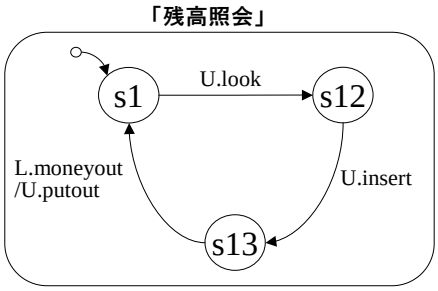

図 4.15: 残高照会の状態遷移図

「残高照会」の仕様は以下の通りである.

- イベント名は write である.
- ユーザが挿入した通帳に現在の預金残高を記入する.
- ユーザの口座データは銀行が把握しているものとする.

この「残高照会」も他のサービスと同様に

- 各サービス要求 → 通帳挿入 → サービス実行

という遷移構造になっているので基本形に変形し同様に合成する. このとき新たに属性 e に look という属性値を追加する.

以上より , 本研究で提案した切り出し/変形/合成方法を用いいれば, システムへの機能追加を容易に行なうことができる.

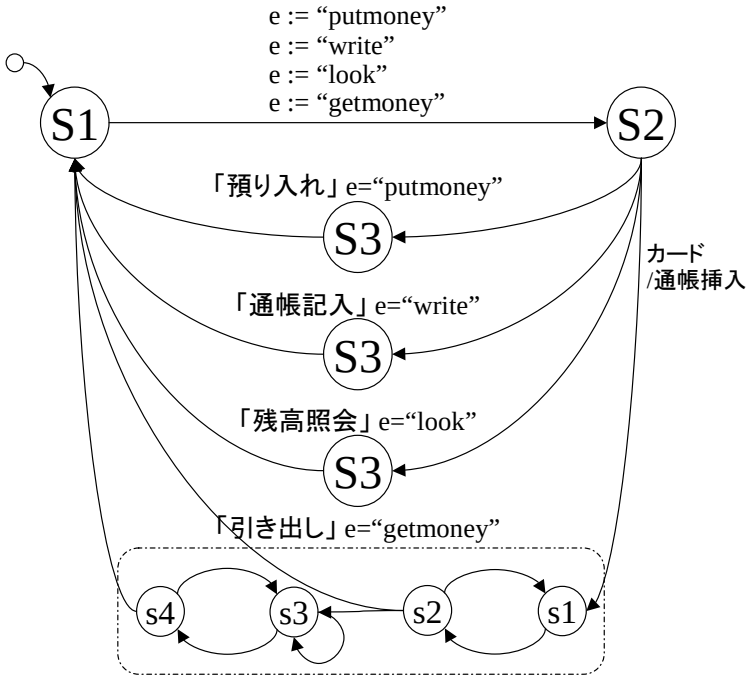

図 4.16: 残高照会を追加した ATM の状態遷移図

4.4.4 ATM に対するテスト

本研究で提案した開発手法において, 機能ごとのテストとシステム全体に対するテストについて説明する.

機能ごとのテスト

以下の ML コードは「引き出し」に対するテストスクリプトである. これは ATM から引き出した金額と口座の金額が同じであるかどうかを確かめるためのテストである.

```
(*
 ATM 動作チェックスクリプト
 *)

local
    val attr = Attribute.attr;
    val makeEventAttr = Event.makeEventAttr;
    val system = SysConf.system;
    open AttrType;

    (* 属性 *)
    val CorrectPass = attr ("pass", String "correct");
    val IncorrectPass = attr ("pass", String "incorrect");
    fun AttrVal n = attr ("val", Int n);

    (* イベント *)
    val Null = [Event.Null];
    val GetMoney = [makeEventAttr ("U.getmoney",[])];
    val Insert = [makeEventAttr ("U.insert",[])];
    val Pass1 = [makeEventAttr ("U.inpass",[CorrectPass])];
    val Pass2 = [makeEventAttr ("U.inpass",[IncorrectPass])];
    fun NumIn n = [makeEventAttr ("U.number",[AttrVal n])];

in
    (* イベント全パターン生成関数 *)
    fun makeSeq (ret,0) = ret
      | makeSeq (ret,n) =
        let
            fun appendEv ev [] = [[ev]@[Null]@[Null]]
              | appendEv ev l =
                map (fn a => a@[ev]@[Null]@[Null]) l;
        in
            makeSeq (appendEv Pass1 ret,n-1)@
            makeSeq (appendEv Pass2 ret,n-1)@
            makeSeq (appendEv (NumIn 10) ret,n-1)
        end

    (* シミュレータ呼出部 *)
    fun execAtm seq = SysConf.system TopList seq 200;

    (* ログから指定したイベント部分だけ抜き出す *)
    fun checkEvent log evl =
        let
            fun contain e [] = false
              | contain e (f::es) =
                if (Event.name e=f) then
                    true
```

```
                else
                    contain e es;
            fun refinel [] fs = []
              | refinel (e::es) fs =
                let
                    val e2 = List.filter (fn el => contain el fs) e
                in
                    if null e2 then
                        refinel es fs
                    else
                        e2::refinel es fs
                end
        in
            refinel (SysConf.sysevlist log) evl
        end

    (* ログから指定したイベントの val 属性の値(Int)だけ抜き出す *)
    fun checkVal log ev =
        map
        (fn a =>
         (unpackInt (Event.eventAttr ev "val" a)
         ))
        (checkEvent log [ev]);

    (* ログから指定したオブジェクトの指定した属性(Int)の変分を調べる *)
    fun checkAttrDiff log obj atr =
        let
             fun getval oc =
                let
                    val a:Attribute.T list list =
                        (ObjConf.objAttr o SysConf.objConf) oc obj;
                in
                    unpackInt (Attribute.pickupAttr atr (hd a))
                end
            val firstval =  getval (List.hd (List.tl log));
            val lastval = getval (List.last log);
        in
            lastval-firstval
        end

    (* GetMoney チェック関数 [結果が 0 なら正常] *)
    fun checkGetMoney n =
        let
            (* イベント全パターン生成 *)
            val seqAll = makeSeq ([[GetMoney]@[Insert]],n);
            (* 1 パターンチェック *)
            fun checksub e =
                let
                    val log = execAtm e;
                    val outs = List.foldr (op+) 0 (checkVal log "U.putout");
                    val change = checkAttrDiff log "ATMSYSTEM_bank" "val";
                in
                    (outs+change<>0)
                end;
        in
            (* 全パターンチェックして、エラーになった個数を返す *)
            List.length (List.filter (checksub) seqAll)
        end

end
```

図 4.17: 引き出し用テストスクリプトの構造

このコードのうち, イベント全パターン生成関数 `makeSeq` はテストしたいイベント列を生成する関数である.

機能ごとのテストスクリプトは, 上記のように ML 言語を用いて簡単に記述することができる. また, このテストスクリプトは切り出された「引き出し」サービスだけに適用できるだけではなく, 合成後のシステムの「引き出し」サービスにも用いることができる.

システム全体に対するシステム

他のサービス (通帳記入, 残高照会, 預り入れ) のためのテストスクリプトも上記と同様に記述すると, ATM システム全体のテストスクリプトは以下のように記述できる.

```
fun checkAll n =
    checkGetMoney n + checkPutMoney n + checkWrite n;
```

このように, 機能ごとのテストスクリプトを用いることで, 合成された後のシステム全体のためのテストスクリプトも簡単に記述することができる.

4.5　切り出し/変形/合成方法に基づく発展的開発方法

全節より, システムの機能ごとの状態遷移図に注目し, それを切り出して基本形に変換し, 同じ状態名同士を合成するという手法を提案した.

基本形に変形する手順を例で示したが, 基本形に変形することが難しい機能をシステムに追加しなければならない場合もある. そこで, 基本形を別の基本形で記述する方法を示すことで, システムを発展的に記述する開発方法を提案する.

例えば, 図 4.16 の上図に基本形 1 に従って設計したシステムがある. このシステムに基本形 1 にはそぐわない機能 c を新たに追加しようとする場合を考える.

ここで提案することは, 新たな機能 c に対する基本形 2 を構築し, それを基本形 1 の基本形とみなすことにより機能をシステムに合成していく方法である.

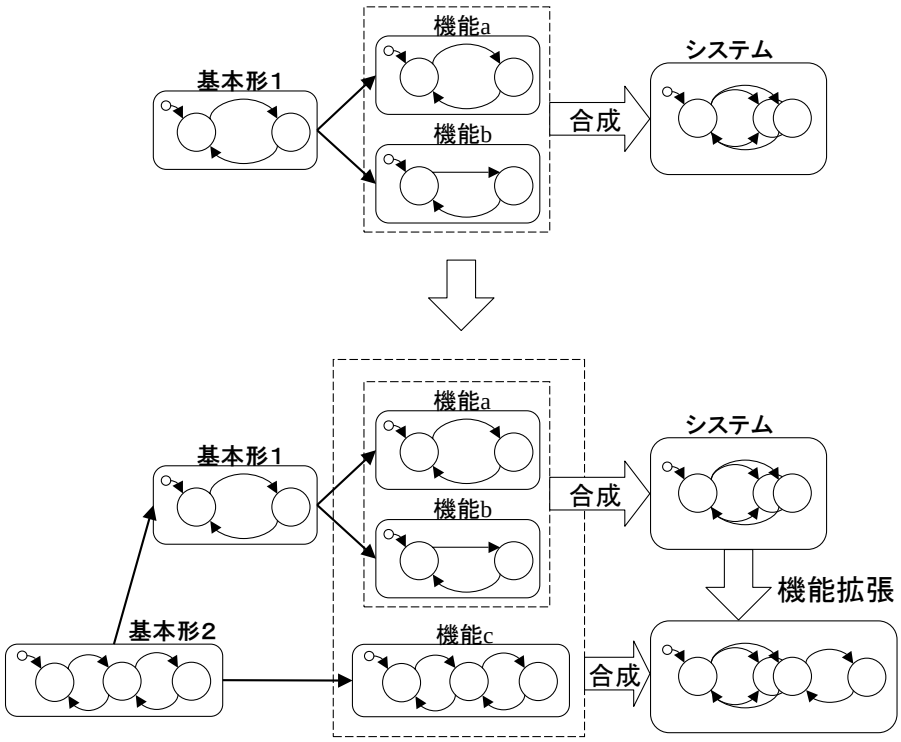

図 4.18: 基本形を用いた開発方法

この方法により機能 c はシステムに合成することができ, 結果, システムにとって機能拡張となる. さらに, 基本形 1 と基本形 2 にもそぐわない機能を追加する場合においても同様で, 新たな基本形を作成することで解決することができる. ゆえに, 基本形を段階的

に作成し基本形同士の関係を木構造化することにより, システムを記述していくことが可能となる.

この提案により, システム構築を発展的に行なうことが期待できる.

第 5 章

実用規模システムへの適用

前章では, 本研究で提案した状態遷移図の切り出し/変形/合成方法を用いて架空の ATM に対する記述例を示した. そこで本章では, 実用化されているシステムの仕様を ObTS/ObCL で記述することにより, 前章で提案した設計方法の有効性を示すことにする.

5.1 事例研究

5.1.1 IrDA 規格の概略

本研究では, IrDA(Infra-red Data Association) が規定している赤外線データ通信の規格の一つである IrLAP の仕様書を基に, ObTS/ObCL によるシステム記述を行なった.

ここではまず, 赤外線データ通信の規格について説明する.

図 5.1 は IrDA の規定するプロトコル層を示したもので, 以下にその説明を示す.

- IrSIR 層 (物理層)
  IrDA のハードウエア規格 (赤外線送受信モジュールの信号強度, 指向性, 到達距離, エラーレート, パルス変調方式など) を規定する.
- IrLAP 層 (リンクアクセス層)
  IrDA シリアル赤外線物理層などによって提供される直接半二重シリアル赤外線による物理的な通信メディアにおいて, コンピュータとその周辺機器間の相互通信を実現するための通信プロトコルである.
- IrLMP 層 (リンクマネージメント層)
  IrLAP 層が, 一対一のデータリンクとデータ品質を保証するプロトコルであるのに

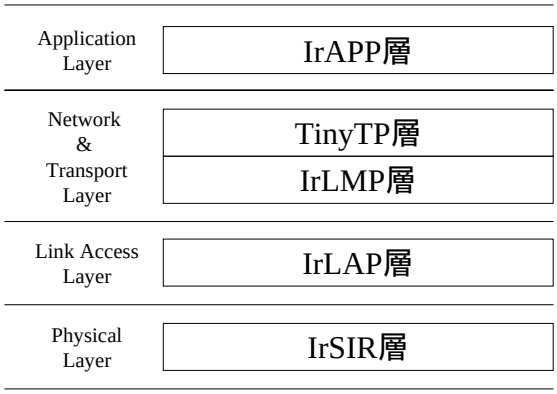

図 5.1: IrDA の規定する 4 つの層

対して, その上位層である LMP では, 一本のデータリンクを複数のリンクアクセスポイントに分割して複数のデータリンクを可能にする層である.

- TinyTP 層 (フロー制御層)
  IrDA の物理的な接続と, その上に乗る論理的な接続での物理通信速度の違いや, リンクアクセスポイントごとのフロー制御の処理を担うのがこの層である.
- IrAPP 層 (アプリケーション層)
  特定の目的を持った赤外線機器に IrDA を使用するために, よりアプリケーションに近い機能を定義した層である.

ここで, IrLAP を事例研究として取り上げた理由を以下に示す.

- 仕様書が公開されていて入手しやすい.
  IrDA のホームページから誰でも仕様書を入手できる.
- 実際に実用化されているデバイスである.
  現在, ノートパソコンや PDA(Presonal Digital Assistant) などの情報携帯端末器の通信方式として採用されている.
- 例題として適度な大きさである.
  IrLAP の各通信フェーズにおける動作を規定するものをここでは手続きと呼んでいる. 各手続きは平均して 5 つぐらいの状態を持つ状態遷移表で記述されており, その状態遷移表が 7 枚集まることで IrLAP の動作が表現されている.

  また, その状態遷移規則は複雑で, 仕様書の規模は A4 で 116 ページに及んでいる.

5.1.2 IrLAP の動作説明

IrLAP のシステムの構成と動作手続きの状態遷移図を示す.

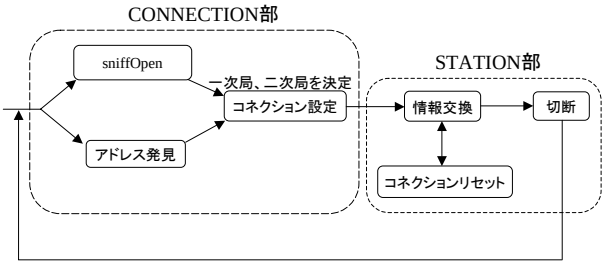

図 5.2: IrLAP の動作手続き

IrLAP で規定されているのは次の 7 つの動作手続きである.

1 リンク開始/終了手続き:
通信が動作可能/不可能時の IrLAP の動作を管理する. この手続きにより IrLAP 層が起動され, これより以下の動作手続きが動作可能となる.

2.a) アドレス発見手続き:
自局の通信可能範囲内に存在する相手局を探し出し手続きである. 自局からデータ通信を行ないたい相手を探す場合に用いる.

2.b) Sniff Open 手続き:
自局が接続の確立を望んでいることをブロードキャストする手続きである.

IrLAP 層では, ユーザの要求にしたがって, アドレス発見手続きか Sniff Open 手続きが選択される.

3) コネクション設定手続き:
アドレス発見手続き時に決定されたデバイスアドレスとの間で接続を設定するための動作手続きである.

アドレス発見手続き, Sniff Open 手続き, コネクション設定手続きを CONNECTION 部と呼ぶ. CONNECTION 部によって情報交換をしたい相手局との接続が確立される.

4) 情報交換手続き:
CONNECTION 部で設定された相手局との接続上で, どのように情報フレームを交換するかを管理する手続きである.

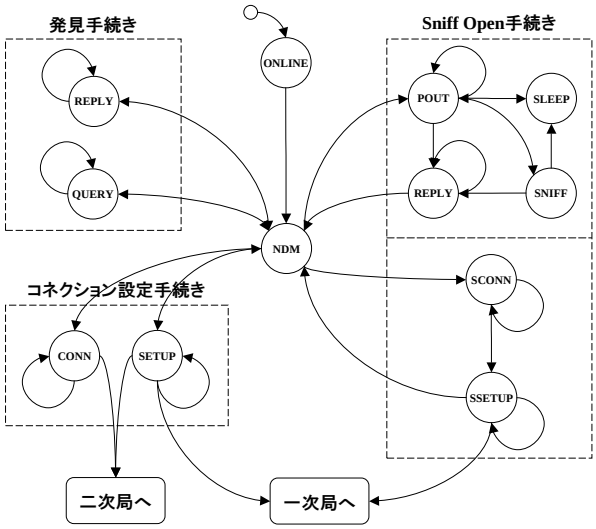

図 5.3: CONNECTION 部の状態遷移図

5.a) コネクションリセット手続き:
設定されている接続をリセットするための手続きである.

5.b) 切断手続き:
設定されている接続を切断するための手続きである.

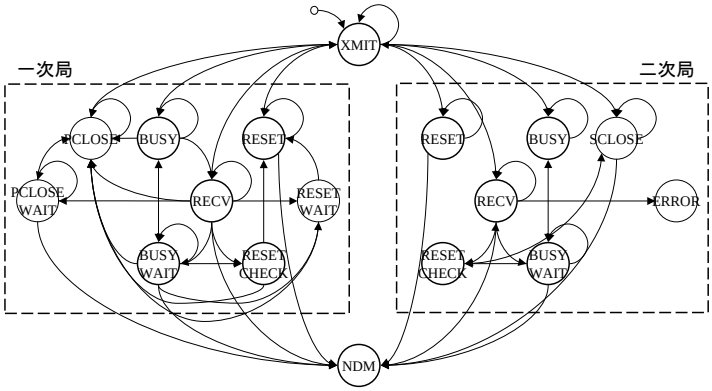

図 5.4: STATION 部の状態遷移図

ここまでの動作手続きを STATION 部と呼ぶ. STATION 部によって接続した相手局と情報交換が行なわれる. 仕様書では, これらの手続きは, 一次局, 二次局手続きとしてまと

めて記述されている.

ここで, 情報フレームとは, IrLAP データリンク上でのすべてのデータと制御の転送を構成している特定のフォーマットである.

これら動作手続きと仕様書の記載から, IrLAP のオブジェクトモデルを示す.

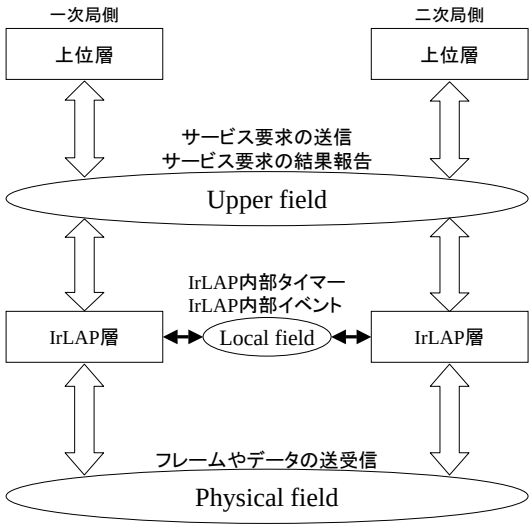

図 5.5: IrLAP とフィールド

ObCL で IrLAP のシステム記述をするために三つのフィールドを設定する.

- Upper field は IrLAP 層と IrLAP よりも上位にある層 (サービスユーザ層と呼ばれる) とのイベント通信路で, ユーザのサービス要求やその結果に関わるイベントが通信される.
- Physical field は IrLAP 層とそれよりも下位にある層 (物理層と呼ばれる) とのイベント通信路で, 情報フレームやデータのやりとりが行なわれる.
- Local field は IrLAP 層内部でのイベント通信路で, IrLAP の内部イベントやタイムアウトを制御しているタイマーオブジェクトとのイベントのやりとりが行なわれる.

以上をもとに, IrLAP のシステム記述を切り出し/変形/合成方法を用いて記述する.

5.2 IrLAP のシステム記述に対する切り出し/変形/合成方法の適用

5.2.1 状態遷移図の切り出し

IrLAP の仕様書では, 発見手続き, Sniff Open 手続き, Connection 設定手続きについては状態遷移表がすでに機能ごとに分けて記述されているので, 状態遷移図の切り出しを行なう必要はない.

しかし, 情報交換手続き, 切断手続き, コネクションリセット手続きについては, それら三つの手続きが合成されて一次局, 二次局手続きとして記述されている. 仕様書には, 情報交換手続き, 切断手続き, コネクションリセット手続きに対する明確な記述は記載されてなかったため, 状態遷移図の切り出しは行なわず, 一次局, 二次局手続きの記述を元に基本形への変形と機能同士の合成を行なった.

5.2.2 基本形への変形

IrLAP を ObTS/ObCL で記述する際, CONNECTION 部または STATION 部の基本形に着目する. つまり,

- CONNECTION 部では
  NDM(正規切断モード) 状態 → 各手続きの処理 → NDM 状態
  というような動作が共通である.

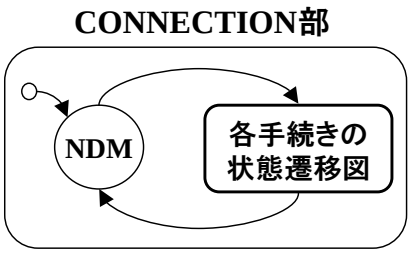

図 5.6: CONNECTION 部の基本形

- STATION 部では
  XMIT(フレームの送受信状態) → `RECV`, `BUSY`, `BUSY WAIT`, `RESET`, `RESET CHECK` → NDM 状態
  というような動作が共通である.

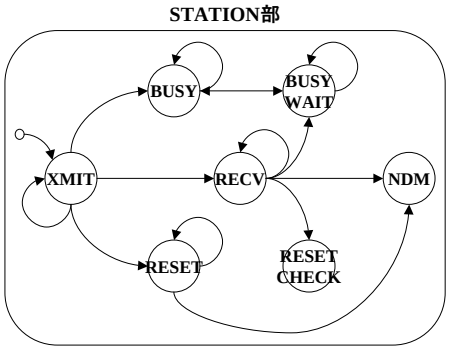

図 5.7: STATION 部の基本形

5.2.3 機能同士の合成

前章の ATM と同様に, CONNECTION 部と STATION 部の基本形を合成して IrLAP のシステムを記述する.

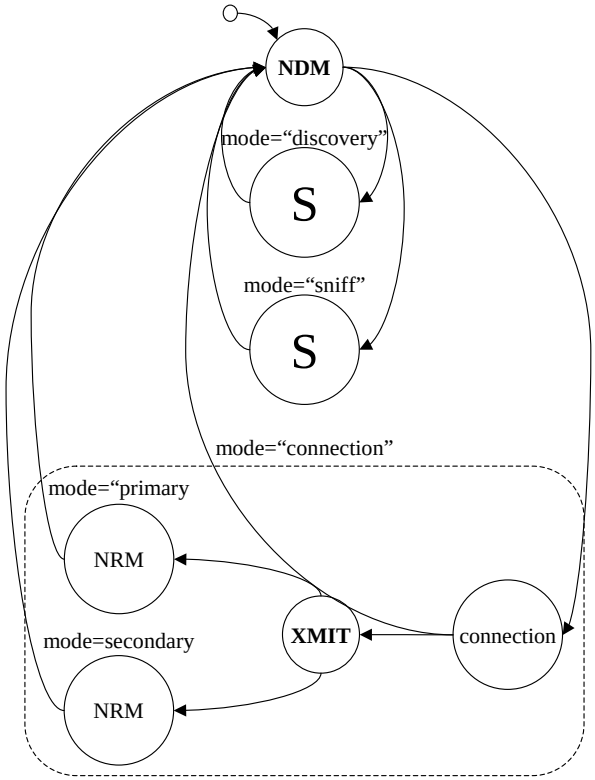

図 5.8: 合成後の IrLAP のシステム

図 5.8 は合成した後の IrLAP のシステムである. ここで S はアドレス発見手続きと Sniff Open 手続きの集合状態, 状態 connection はコネクション設定手続きの集合状態, 状態 NRM は一次局と二次局手続きの集合状態を表している.

これまでの手順により, IrLAP のような複雑なシステムにおいても前章の ATM と同様に本研究で提案した方法を適用できることがわかる. ゆえに各手続きに対して, または切り出し/変形/合成した後のシステムに対してテストやデバックを行なったり, 新たに手続きを追加したりすることが容易になる.

5.3 事例研究のまとめ

5.3.1 本研究で提案した設計方法に対する評価

本研究ではIrDAの提供するIrLAPの仕様書に基づき, IrLAPのシステムをObTS/ObCLにより記述を進めていった.

その際, 本研究で提案した切り出し/変形/合成による設計方法を利用することで, ATMの例と同様に, 手続きごとや合成後のシステムに対する実行・テスト, IrLAPシステムへの機能拡張・追加が容易になることがわかった.

5.3.2 IrLAPの仕様書について

本研究では, ObTS/ObCL/ObMLによりシステムを記述することで仕様書では明確に記されていない箇所や誤った記述を発見することができた.

Send-Data-With-P-Bit-Set. This action is carried out as the result of a *Data-Request* event. If this is the first frame in the window it should be sent after waiting the minimum turn around time. If the event is for reliable data the following actions are carried out:

stop-P-timer
Store[Vs] := data; Aek[Vs] := false
Send i:Vr:Vs:P:*data*
Vs := Vs + 1 mod 8
window := windowSize
AckRequired := false
start-F-timer

自局から“i:Vr:Vs:P”というフレームが送信され,
相手局は“i:Ns:Nr:P”というフレームを受信するはずが・・・

RECV (entry state) (note 5)	*Recv* i:cmd:Ns:Nr:¬P	*Data-Indication* Vr := Vr − 1 mod 8 Update Nr Received AckRequired := true *Start-WD-timer (optional: see note 6)*	RECV
	Recv i:cmd:Ns:Nr:P ∧ *Pending-Data-Requests* ∧ ¬ remoteBusy	*Data-Indication* Vr := Vr − 1 mod 8 Update Nr Received AckRequired := true *Stop-WD-timer*	XMIT
	Recv i:cmd:Ns:Nr:P ∧	*Data Indication* Vr := Vr − 1 mod 8	RECV

図 5.9: 仕様書の具体的な誤り

図5.8は一次局手続きの仕様に対する具体的な誤りを指摘した一例である.

自局 (一次局) から「i:Vr:Vs:P」というフォーマットをしたフレームが物理層へ送られ, 相手局 (二次局) は「i:Ns:Nr:P」というフォーマットのフレームを受信することが説明されている.

しかし, 実際にフレームを受信する側の状態遷移表では, 受信されているフレームのフォーマットが「i:cmd:Ns:Nr:P」となっている.

仕様書の記述を詳しく調べると, 「i:cmd:Ns:Nr:P」のフォーマットの方が正しいことがわかったが, 仕様書通りに実装していた時にはこの誤りのためにフレームの転送がうまくいかず, 結果, ObML 上でのシミュレーション時に IrLAP のシステムが意図した通りの動作を行なわなかった.

また, 遷移先が複数に渡る記述があった. ObTS では遷移先は必ず一つでなければならない決まりがあるので, それを記述することは難しい. 仕様書によると, 遷移先を一つに決定するための条件は,

(i) 内部状態

(ii) 層の処理動作

(iii) 実装時の決定

に基づくと記述されているが, それらについての明確な違いが記載されていなかった. そのため, 実際の記述では特に条件を考慮せずに一つの遷移先を選択した.

このように, IrLAP の仕様書には曖昧な設計があり, 仕様書を元にした記述を行なう際には IrLAP の動作に精通する必要があることがわかる.

第 6 章

まとめ

本研究では, システムを記述する際の再利用性の高い設計手法を提案した. 再利用性を上げる具体的アイディアとして以下の設計手法を考案した.

1) システムの中から機能ごとの状態遷移図を切り出す.

2) 機能ごとの状態遷移図を基本形に変形する.

3) 各機能の状態遷移図を合成する.

架空の ATM の仕様を想定し, その ObTS 図に本研究で提案した切り出し/変形/合成方法を適用すること機能ごとの実行・テスト, 設計方法を適用した後のシステム全体に対する実行・テストを行なうことができ, 機能拡張・追加が容易であることを示した.

実用規模のシステムである IrLAP の状態遷移図に対し, 本研究の設計方法を適用し, その有効性を示した.

また, 仕様書に従い ObTS/ObCL でシステムを記述し, ObML を用いてシミュレーションすることにより, 仕様書の曖昧な箇所や誤りを発見することができた.

第 7 章

今後の課題

7.1 状態遷移図の切り出し/変形/合成方法に関する方法論の提案

本研究では, 状態遷移図を利用してシステムを記述する際, 再利用性の高い記述手法の提案として, 状態遷移図の切り出し/変形/合成という設計手法を示した.

しかし,

- システムから機能ごとの状態遷移図を切り出す方法
- 切り出された状態遷移図を基本形へ変形する方法
- 基本形に変形された状態遷移図を一つのシステムへ合成する方法

に関しては具体例を列挙するに留まり, それらに関しての方法論の構築までには至らなかった. ゆえに, 機能の切り出し, 基本形への変形, 機能同士の合成に対しての方法論を構築する必要がある.

7.2 合成方法の自動化

合成方法の自動化という提案がある.

ObCL で記述されたシステムは, ObCL コンバータにより Standard ML コードに変換され, ObML シミュレータ上でシミュレートすることができる.

この ObCL コンバータに本研究で提案した設計手法を適用する.

自動化を適用された ObCL コンバータは

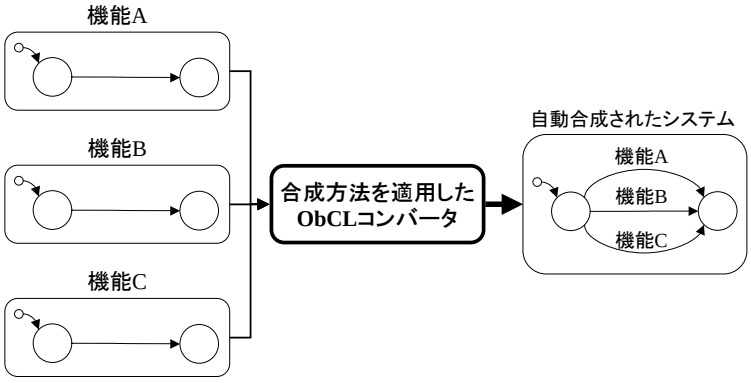

図 7.1: 合成方法の自動化

- 各機能における状態名の同じ箇所を合成する.
- 機能を合成する際に遷移先を制御する属性を追加する.

という作業を ObCL コード → ML コードへ変換する際に行なうことが期待される.

これによりシステム設計者はシステムの機能ごとの設計, 実行, テストのみに集中すればよく, 結果, 開発効率の向上が望める.

謝辞

本研究を行うに当たり, 有益な御指導, 助言および援助を頂きました片山 卓也先生に深く感謝の意を表します. また, 本研究についてさまざまな議論をして頂いた権藤 克彦先生に感謝いたします.

片山研究室の伊藤 恵助手には日頃から有益な御助言, 御支援をいただきました. 感謝いたします. キャノンソフトウェア株式会社の久保秋 真氏には格別の御支援を頂戴いたしました. 御礼申し上げます.

本論文をまとめるに当たって御協力いただいたソフトウェア基礎講座のメンバーの皆様方に深く感謝致します。

本研究に関する発表論文

[1] 野村 昌男, 久保秋 真, 伊藤 恵, 片山 卓也: 組み込みシステム設計のための ObTS に基づく記述支援環境に関する研究, 情報処理学会報告 (2000-SE-125), Vol.2000, No.4, pp.91-98, 2000/1

参考文献

[1] 伊藤 恵, 片山 卓也: オブジェクト指向方法論のための動的モデル ObTS, 修士論文, 北陸先端科学技術大学院大学 (1995).

[2] 伊藤 恵, 片山 卓也: オブジェクト指向方法論のための動的モデルにおける Statechart 式通信モデル, 日本ソフトウエア科学会第 13 大会論文集, 27-15-A2, pp445-448 (1996).

[3] 久保秋 真, 伊藤 恵, 片山 卓也: ObTS モデルに基づくオブジェクト指向仕様記述言語と支援環境, 日本ソフトウエア科学会第 14 大会論文集, D11-4, pp589-592 (1997).

[4] 久保秋 真, 片山 卓也: 動的モデル ObTS の大規模組込みシステム記述に関する研究, 修士論文, 北陸先端科学技術大学院大学 (1998)

[5] IBM Corporation, Hewlett-Packard Company, Apple Computer,Inc., Counterpoint Systems Foundry,Inc.: Infrared Data Association Serial Infrared Link Access Protocol (IrLAP) (1996)

[6] 塚本 昌彦, 宇野 裕史: 赤外線通信のデータリンク層プロトコル, インターフェース pp174-185 (1995)

[7] D.Harel, A.Pnueli, J.P.Schmid and R.sherman: On the Semantics of Statecharts, Proc of 2nd IEEE Symposium on Logic in Computer Science, pp54-64 (1987)

[8] J. ランボー, M. ブラハ, W. プレメラニ, F. エディ, W. ローレンセン, 羽生田 栄一: オブジェクト指向方法論 OMT モデル化と設計, トッパン (1992)

付録

IrLAP の ObCL コードを記載する.

```
--- IrLAP Procedure
--- IrLAP 手続き


--- イベントクラス記述部
--- 上位層イベントクラス
event UPEVENT
  inherit GENERIC_EVENT
  attribute num:Int
end
event UPVALEVENT      --- 整数属性を持つイベント
  inherit UPEVENT
  attribute val:Int
end
event UPMSGEVENT      --- 文字列属性を持つイベント
  inherit UPEVENT
  attribute msg:String
end

--- 物理層イベントクラス
event PHEVENT
  inherit GENERIC_EVENT
  attribute num:Int
end
event PHVALEVENT      --- 整数属性を持つイベント
  inherit PHEVENT
  attribute val:Int
end
event PHTWOVALEVENT   --- 整数属性を二つ持つイベント
  inherit PHVALEVENT
  attribute va2:Int
end
event PHMSGEVENT      --- 文字列属性を持つイベント
  inherit PHEVENT
  attribute msg:String
end
event PHVALMSGEVENT   --- 整数と文字列属性を持つイベント
  inherit PHVALEVENT
  attribute msg:String
end
event PHMSGVAL2EVENT  --- 文字列と二つの整数属性を持つイベント
  inherit PHTWOVALEVENT
  attribute msg:String
end

--- IrLAP 層内部イベントクラス
```

```
event INEVENT
  inherit GENERIC_EVENT
  attribute num:Int
end
event INVALEVENT      --- 整数属性を持つイベント
  inherit INEVENT
  attribute val:Int
end
event INMSGEVENT   --- 整数と文字列の属性を持つイベント
  inherit INEVENT
  attribute msg:String
end
event INBOOLEVENT     --- Boolean 属性を持つイベント
  inherit INEVENT
  attribute bool:Bool
end
--- イベント記述部 ここまで



--- フィールド記述部
--- Field for Station User-IrLAP: 上位層 (サービスユーザ層) と IrLAP 間のフィールド
field UPPER
  event Discovery_Request       ---(discovery)
  :UPVALEVENT

  event Connect_Request,        ---(sniffOpen,connection)
        Data_Request,           ---(NRMP,NRMS)
        Discovery_Confirm,      ---(discovery)
        Disconnect_Indication, ---(linkInit,NRMP,NRMS)
        Discovery_Indication,   ---(discovery,sniffOpen)
        Reset_Indication,       ---(NRMP,NRMS)
        GR_ConnectionAdr,       ---(sniffOpen)
        string                  ---(Common) メッセージ表示
  :UPMSGEVENT

  event link_initialize, ---(linkInit) 局の初期化/起動
        link_shutdown, ---(linkInit) 局の終了

        Sniff_Request,          ---(sniffOpen)
        Disconnect_Request,     ---(connection,NRMP,NRMS)
        Reset_Request,          ---(NRMP,NRMS)
        Pending_Requests,       ---(NRMS)
        Pending_Data_Requests,  ---(NRMS)
        No_Pending_Data_Requests, ---(NRMS)
        Connect_Confirm,        ---(connection)
        Reset_Confirm,          ---(NRMP,NRMS)
        Connect_Response,       ---(connection)
        Reset_Response,         ---(NRMP,NRMS)
        Connect_Indication,     ---(sniffOpen,connection)
        Disconnect_Indication,  ---(sniffOpen,connection)
        Data_Indication,        ---(NRMP,NRMS)
        Unidata_Indication,     ---(NRMP,NRMS)
        Reset_Indication,       ---(NRMP)
        Status_Indication,      ---(NRMP)
        Discovery_Collision,    ---(discovery)
        Response_Collision,     ---(discovery)
        Enable_receiver_etc,    ---(sniffOpen)
        Disable_receiver_etc,   ---(sniffOpen)
        Prepare_FRMR_Response   ---(NRMS)
  :UPEVENT
end
```

```
--- Field for Service User-IrLAP: IrDA 物理層と IrLAP 間のフィールド
field PHYSICAL
  event Recv, Send              ---(discovery,sniffOpen,connection,NRMP,NRMS)
  :PHMSGEVENT

  event Recv_Con,               ---(sniffOpen,connection)
        Send_Con                ---(sniffOpen,connection)
  :PHMSGVAL2EVENT

  event Recv_dCmd,              ---(discovery,sniffOpen)
        Send_dCmd               ---(discovery)
  :PHTWOVALEVENT

  event Recv_dRsp,              ---(discovery,sniffOpen)
        Recv_sRsp,              ---(sniffOpen)
        Send_dRsp,              ---(discovery,sniffOpen)
        Send_sRsp               ---(sniffOpen)
  :PHVALMSGEVENT

  event physical_layer_down ---(linkInit) 物理層の使用不可
  :PHEVENT
end

--- Field for IrLAP InnerEvent: IrLAP 層の内部イベントフィールド
field LOCAL
  event Pending_Busy_Cleared    ---(NRMS)
  :INBOOLEVENT

   event slot_timer,       ---(discovery)
         query_timer,      ---(discovery,sniffOpen)
         sense_timer,      ---(sniffOpen)
         sniff_timer,      ---(sniffOpen)
         sleep_timer,      ---(sniffOpen)
         P_timer,          ---(sniffOpen,NRMP)
         F_timer,          ---(connection,NRMP)
         WD_timer,         ---(connection)
         countup           ---(Common inner)
  :INMSGEVENT

  event close_connection,       ---(linkInit) クリーンアップ動作の実行

        ADC_Parameters,         ---(linkInit,NRMP)
        AC_Parameters,          ---(sniffOpen,connection)
        NC_Parameters,          ---(sniffOpen,connection)

        Discovery_Abort_Condition, ---(discovery)
        Parform_Random_Backoff, ---(connection)
        Local_Busy_Detected,    ---(NRMP)
        Local_Busy_Cleared,     ---(NRMP)
        WMT_Delay,              ---(NRMP)
        RB_Data,                ---(NRMP)
        Update_Nr_Received,     ---(NRMP)
        resend_rejected_frames  ---(NRMP)
   :INEVENT
end
--- フィールド記述部 ここまで



--- クラス記述部
class IRLAP
  field O:UPPER; P:PHYSICAL; L:LOCAL
  attribute num, dest, ca, sa, N1, N2, N3, NA, mode,
```

```
            S, s, slot, maxSlot, slotCount,
            window, windowSize, Vr, Vs, retryCount :Int
  attribute Store_Vs :String
  attribute frameSent, mediaBusy, remoteBusy,
            Ack_Vs, AckRequired, xmitFlag :Bool
  state OFFLINE,             --- オフライン状態
        NDM,                 --- 正規切断モード
        QUERY, REPLY,        --- (discovery,Sniffing)
        POUT, SNIFF, SLEEP, --- (Sniffing)
        SCONN, SSETUP,       --- (Connect to Sniffer)
        CONN, SETUP,         --- (Sniffing,connection)
        XMIT,                --- (NRMP,NRMS)
        RECV,                --- (NRMP,NRMS)
        RESET_CHECK, RESET, --- (NRMP,NRMS)
        BUSY, BUSY_WAIT,     --- (NRMP,NRMS)
        RESET_WAIT,          --- (NRMP)
        PCLOSE, PCLOSE_WAIT,--- (NRMP)
        ERROR, SCLOSE        --- (NRMS)
  transition
  start is
    source init
    do  window:=6;
        slot:=0;  --- Generate_Random_Time_Slot(S,s)
mode:=0   --- モードを初期状態にする
    destination OFFLINE
  end

  --- リンク初期化とシャットダウン手続き
  --- Offline State: オフライン状態
  init10 is
    source OFFLINE
    input O.link_initialize
    when O.link_initialize.num=num          --- デバイスオブジェクトを選択
    do  O.string.msg:="IrLAP layer Online"  --- オンラインメッセージ
    destination NDM
    output {L.ADC_Parameters,
    O.string}
  end

  --- Online State: オンライン状態
  init20 is
    source NDM
    input O.link_shutdown             --- 局ユーザが局を終了したとき
    when O.link_shutdown.num=num            --- デバイスオブジェクトを選択
    do  O.Disconnect_Indication.msg:="aborted";
        O.string.msg:="IrLAP layer Offline" --- オフラインメッセージ
    destination OFFLINE
    output {O.Disconnect_Indication,     --- abort したことを上位層へ通知
    L.close_connection,
            O.string}
  end
  init21 is
    source NDM
    input P.physical_layer_down     --- 物理層が使用不可のとき
    when P.physical_layer_down.num=num      --- デバイスオブジェクトを選択
    do  O.Disconnect_Indication.msg:="aborted";
        O.string.msg:="IrLAP layer Offline" --- オフラインメッセージ
    destination OFFLINE
    output {O.Disconnect_Indication,     --- abort したことを上位層へ通知
    L.close_connection,
            O.string}
  end
  --- リンク初期化とシャットダウン手続き ここまで
```

```
  --- Discovery Procedure
  --- 発見手続き
  --- for NDM
  disc10 is
    source NDM
    input O.Discovery_Request
    when mode=0   --- どのモードも選択されてない
     and O.Discovery_Request.num=num
     and mediaBusy=false
    do  S:=O.Discovery_Request.val;
maxSlot:=(O.Discovery_Request.val-1);
        slotCount:=0;
        P.Send_dCmd.val:=O.Discovery_Request.val-1; --- maxSlot
        P.Send_dCmd.va2:=0;                         --- slotCount
L.slot_timer.num:=num;
L.slot_timer.msg:="start";
mode:=1;  --- 発見手続き中を示す
        O.string.msg:="log is empty"
    destination QUERY
    output {P.Send_dCmd,
            L.slot_timer,
            O.string}
  end
  disc11 is
    source NDM
    input O.Discovery_Request
    when mode=0   --- どのモードも選択されてない
     and O.Discovery_Request.num=num
     and mediaBusy=true
    do  O.Discovery_Indication.msg:="media_busy"
    destination NDM
    output O.Discovery_Indication
  end
  disc12a is
    source NDM
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot=s
    when mode=0   --- どのモードも選択されてない
     and slot=P.Recv_dCmd.va2   --- slot はS と s の値によってランダムに決定される
    do  P.Send_dRsp.val:=NA;
        P.Send_dRsp.msg:="info";
L.query_timer.num:=num;
L.query_timer.msg:="start";
        frameSent:=true;
        mode:=1   --- 発見手続き中を示す
    destination REPLY
    output {P.Send_dRsp,
            L.query_timer}
  end
  disc12b is
    source NDM
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot/=s
    when mode=0   --- どのモードも選択されてない
     and slot/=P.Recv_dCmd.va2   --- slot はS と s の値によってランダムに決定される
    do  L.query_timer.num:=num;
L.query_timer.msg:="start";
```

```
frameSent:=false;
mode:=1    --- 発見手続き中を示す
    destination REPLY
    output L.query_timer
  end

  --- for QUERY
  disc20 is
    source QUERY
    input L.slot_timer
    when mode=1   --- 発見手続き中
     and L.slot_timer.num=num
     and L.slot_timer.msg="expired"
     and slotCount<maxSlot
    do  P.Send_dCmd.val:=maxSlot;
        P.Send_dCmd.va2:=slotCount+1;
L.slot_timer.num:=num;
L.slot_timer.msg:="start";
slotCount:=slotCount+1
    destination QUERY
    output {P.Send_dCmd,
            L.slot_timer}
  end
  disc21 is
    source QUERY
    input L.slot_timer
    when mode=1   --- 発見手続き中
     and L.slot_timer.num=num
     and L.slot_timer.msg="expired"
     and (slotCount>maxSlot or slotCount=maxSlot)
    do  P.Send.msg:="End-Discovery-XID-Cmd";
O.Discovery_Confirm.msg:="log"   --- 発見確認
    destination NDM
    output {O.Discovery_Confirm,
            P.Send}
  end
  disc22 is
    source QUERY
    input L.Discovery_Abort_Condition
    when mode=1   --- 発見手続き中
    do  P.Send.msg:="End_Discovery_XID_Cmd";
O.Discovery_Indication.msg:="aborted";
L.slot_timer.num:=num;
L.slot_timer.msg:="stop"
    destination NDM
    output {O.Discovery_Indication,
            P.Send,
            L.slot_timer}
  end
  disc23 is
    source QUERY
    input P.Recv_dRsp
    when mode=1   --- 発見手続き中
    do  sa:=P.Recv_dRsp.val;   --- 相手局のデバイスアドレスを記憶
O.string.msg:="log:=log or {<sa,info>}"
    destination QUERY
    output O.string
  end
  disc24 is
    source QUERY
    input O.Response_Collision
    when mode=1   --- 発見手続き中
     and O.Response_Collision.num=num
```

```
    do  O.string.msg:="log:=log or {<Empty, Empty>}"
    destination QUERY
    output O.string
  end
  disc25 is
    source QUERY
    input P.Recv
    when mode=1   --- 発見手続き中
     and P.Recv.msg="x:x:x:x"
    destination QUERY
  end

  --- for REPLY
  disc30 is
    source REPLY
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and (s>slot or s=slot)
    when mode=1   --- 発見手続き中
     and (slot<P.Recv_dCmd.va2 or slot=P.Recv_dCmd.va2)
     and frameSent=false
    do  P.Send_dRsp.val:=NA;
        P.Send_dRsp.msg:="info";
        frameSent:=true
    destination REPLY
    output P.Send_dRsp
  end
  disc31 is
    source REPLY
    input P.Recv
    when mode=1   --- 発見手続き中
     and P.Recv.msg="End-Discovery-XID-Cmd"
    do  O.Discovery_Indication.msg:="remote";
L.query_timer.num:=num;
L.query_timer.msg:="stop"
    destination NDM
    output {O.Discovery_Indication,
    L.query_timer}
  end
  disc32 is
    source REPLY
    input L.query_timer
    when mode=1   --- 発見手続き中
     and L.query_timer.num=num
     and L.query_timer.msg="expired"
    do  mode:=0
    destination NDM
  end
  disc33 is
    source REPLY
    input P.Recv
    when mode=1   --- 発見手続き中
     and P.Recv.msg="x:x:x:x"
    destination REPLY
  end
  --- 発見手続き ここまで

  --- Sniff Open 手続き Sniffing
  --- for NDM
  snif10a is
    source NDM
    input O.Sniff_Request
```

```
    when mode=0   --- どのモードも選択されてない
     and O.Sniff_Request.num=num
    do  L.sense_timer.num:=num;
L.sense_timer.msg:="start";
mediaBusy:=false;
mode:=2   --- Sniffing 中
    destination POUT
    output L.sense_timer
  end

  --- for POUT
  snif20 is
    source POUT
    input L.sense_timer
    when mode=2   --- Sniffing 中
     and L.sense_timer.num=num
     and L.sense_timer.msg="expired"
     and mediaBusy=false
    do  P.Send_sRsp.val:=NA;
        P.Send_sRsp.msg:="info";
L.sniff_timer.num:=num;
L.sniff_timer.msg:="start"
    destination SNIFF
    output {P.Send_sRsp,
            L.sniff_timer}
  end
  snif21 is
    source POUT
    input L.sense_timer
    when mode=2   --- Sniffing 中
     and L.sense_timer.num=num
     and L.sense_timer.msg="expired"
     and mediaBusy=true
    do  L.sleep_timer.num:=num;
L.sleep_timer.msg:="start"
    destination SLEEP
    output {O.Disable_receiver_etc,
            L.sleep_timer}
  end
  snif22 is
    source POUT
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot=s
    when mode=2   --- Sniffing 中
     and slot=P.Recv_dCmd.va2   --- slot は S と s の値によってランダムに決定される
    do  P.Send_dRsp.val:=NA;
        P.Send_dRsp.msg:="info";
L.query_timer.num:=num;
L.query_timer.msg:="start";
        frameSent:=true
    destination REPLY
    output {P.Send_dRsp,
            L.query_timer}
  end
  snif23 is
    source POUT
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot/=s
    when mode=2   --- Sniffing 中
```

```
     and slot/=P.Recv_dCmd.va2  --- slot は S と s の値によってランダムに決定される
    do  L.query_timer.num:=num;
L.query_timer.msg:="start";
frameSent:=false
    destination REPLY
    output L.query_timer
  end
  snif24 is
    source POUT
    input P.Recv
    when mode=2   --- Sniffing 中
     and P.Recv.msg="x:x:x:x"
    destination POUT
  end

  --- for SNIFF
  snif30a is
    source SNIFF
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot=s
    when slot=P.Recv_dCmd.va2
    do  P.Send_dRsp.val:=NA;
        P.Send_dRsp.msg:="info";
L.query_timer.num:=num;
L.query_timer.msg:="start";
        frameSent:=true
    destination REPLY
    output {P.Send_dRsp,
            L.query_timer}
  end
  snif30b is
    source SNIFF
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and slot/=s
    when slot/=P.Recv_dCmd.va2
    do  L.query_timer.num:=num;
L.query_timer.msg:="start";
frameSent:=false
    destination REPLY
    output L.query_timer
  end
  snif31 is
    source SNIFF
    input P.Recv_Con
    when P.Recv_Con.msg="u:snrm:cmd:P"
    do  mode:=4;
ca:=P.Recv_Con.val;
dest:=P.Recv_Con.va2
    destination CONN
    output O.Connect_Indication
  end
  snif32 is
    source SNIFF
    input L.sniff_timer
    when L.sniff_timer.num=num
     and L.sniff_timer.msg="expired"
    do  L.sleep_timer.num:=num;
L.sleep_timer.msg:="start"
    destination SLEEP
```

```
    output {O.Disable_receiver_etc,
            L.sleep_timer}
  end
  snif33 is
    source SNIFF
    input P.Recv
    when P.Recv.msg="x:x:x:x"
    destination SLEEP
  end

  --- for SLEEP
  snif40 is
    source SLEEP
    input L.sleep_timer
    when L.sleep_timer.num=num
     and L.sleep_timer.msg="expired"
    do  L.sense_timer.num:=num;
L.sense_timer.msg:="start";
mediaBusy:=false
    destination POUT
    output {O.Enable_receiver_etc,
            L.sense_timer}
  end

  --- for REPLY
  snif50 is
    source REPLY
    input P.Recv_dCmd
    ---when P.Recv_dCmd.val=S
    --- and P.Recv_dCmd.va2=s
    --- and (s>slot or s=slot)
    when mode=2   --- sniffing 手続き中
     and (slot<P.Recv_dCmd.va2 or slot=P.Recv_dCmd.va2)
     and frameSent=false
    do  P.Send_dRsp.val:=NA;
        P.Send_dRsp.msg:="info";
        frameSent:=true
    destination REPLY
    output P.Send_dRsp
  end
  sniff51 is
    source REPLY
    input P.Recv
    when mode=2   --- sniffing 手続き中
     and P.Recv.msg="End-Discovery-XID-Cmd"
    do  O.Discovery_Indication.msg:="remote";
L.query_timer.num:=num;
L.query_timer.msg:="stop";
L.sense_timer.num:=num;
L.sense_timer.msg:="start"
    destination POUT
    output {O.Discovery_Indication,
    L.query_timer,
    L.sense_timer}
  end
  sniff52 is
    source REPLY
    input L.query_timer
    when mode=2   --- sniffing 手続き中
     and L.query_timer.num=num
     and L.query_timer.msg="expired"
    do  L.sense_timer.num:=num;
L.sense_timer.msg:="start"
```

```
    destination POUT
    output L.sense_timer
  end
  sniff53 is
    source REPLY
    input P.Recv
    when mode=2   --- sniffing 手続き中
     and P.Recv.msg="x:x:x:x"
    destination REPLY
  end

  --- Sniff Open 手続き Connect to Sniffer
  --- for NDM
  snif10b is
    source NDM
    input P.Recv_sRsp
    when mode=0   --- どのモードも選択されてない
    do  O.Discovery_Indication.msg:="sniff";
sa:=P.Recv_sRsp.val
    destination NDM
    output O.Discovery_Indication
  end
  snif10c is
    source NDM
    input O.Connect_Request
    when mode=0   --- どのモードも選択されてない
     and O.Connect_Request.num=num
     and O.Connect_Request.msg="sniff"
    do  mode:=3   --- Connect to Sniffer 中
    destination SCONN
  end

  --- for SCONN
  snif60 is
    source SCONN
    input P.Recv_sRsp
    when mode=3   --- Connect to Sniffer 中
    do  ca:=NA+1;
        P.Send_Con.msg:="u:snrm:cmd:P";
        P.Send_Con.val:=NA+1;
        P.Send_Con.va2:=NA;
L.P_timer.num:=num;
L.P_timer.msg:="start"
    destination SSETUP
    output {P.Send_Con,
            L.P_timer}
  end
  snif61 is
    source SCONN
    input P.Recv
    when mode=3   --- Connect to Sniffer 中
     and P.Recv.msg="x:x:x:x"
    destination SCONN
  end

  --- for SSETUP
  snif70 is
    source SSETUP
    input L.P_timer
    when L.P_timer.num=num
     and L.P_timer.msg="expired"
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
```

```
    destination NDM
    output O.Disconnect_Indication
  end
  snif71 is
    source SSETUP
    input P.Recv
    when P.Recv.msg="u:ua:rsp:F"
    do  P.Send.msg:="s:rr:cmd:P";
L.P_timer.num:=num;
L.P_timer.msg:="restart";
mode:=10;   --- 一次局モード
window:=windowSize;
Vr:=0;
Vs:=0;
retryCount:=0;
remoteBusy:=false
    destination XMIT
    output {O.Connect_Confirm,
    P.Send,
    L.P_timer,
            L.AC_Parameters,
            L.NC_Parameters}
  end
  snif72 is
    source SSETUP
    input P.Recv
    when P.Recv.msg="u:dm:rsp:F"
    do  O.Disconnect_Indication.msg:="disconnect";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.P_timer}
  end
  snif73 is
    source SSETUP
    input P.Recv
    when P.Recv.msg="x:x:x:x"
    destination SSETUP
  end
  --- Sniff Open 手続き ここまで


  --- コネクション設定の手続き
  --- for NDM
  conn10 is
    source NDM
    input O.Connect_Request
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and O.Connect_Request.num=num
     and O.Connect_Request.msg="sa"   --- この sa は相手局のデバイスアドレスを示す
     and mediaBusy=false
    do  ca:=NA+1;
dest:=sa;
P.Send_Con.msg:="u:snrm:cmd:P";
P.Send_Con.val:=NA+1;   --- ca
P.Send_Con.va2:=sa;     --- dest
L.F_timer.num:=num;
L.F_timer.msg:="start";
mode:=4;   --- Connection 中
retryCount:=0
    destination SETUP
```

```
    output {P.Send_Con,
    L.F_timer}
  end
  ---conn11 is
  ---  source NDM
  ---  input O.Connect_Request
  ---  when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
  ---   and O.Connect_Request.num=num
  ---   and O.Connect_Request.msg="sa"
  ---   and mediaBusy=false
  ---  do  O.Disconnect_Indication.msg:="disconnect";
  ---      mode:=0   --- モードを初期状態にする
  ---  destination NDM
  ---  output O.Disconnect_Indication
  ---end
  conn12 is
    source NDM
    input O.Connect_Request
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and O.Connect_Request.num=num
     and O.Connect_Request.msg="sa"   --- この sa は相手局のデバイスアドレスを示す
     and mediaBusy=true
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0      --- モードを初期状態にする
    destination NDM
    output O.Disconnect_Indication
  end
  conn13 is
    source NDM
    input P.Recv_Con
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and P.Recv_Con.msg="u:snrm:cmd:P"
    do  mode:=4;   --- Connection 中
ca:=P.Recv_Con.val;
dest:=P.Recv_Con.va2
    destination CONN
    output O.Connect_Indication
  end
  conn14 is
    source NDM
    input P.Recv
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and P.Recv.msg="u:test:cmd:P"
    do  P.Send.msg:="u:test:rsp:F"
    destination NDM
    output P.Send
  end
  conn15 is
    source NDM
    input P.Recv
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and P.Recv.msg="x:x:cmd:P"
    do  P.Send.msg:="u:dm:rsp:F"
    destination NDM
    output P.Send
  end
  conn16 is
    source NDM
    input P.Recv
    when (mode=1 or mode=2) --- アドレス発見か Sniffing が終わっているとき
     and P.Recv.msg="x:x:x:x"
    destination NDM
  end
```

```
  --- for CONN
  conn20 is
    source CONN
    input O.Connect_Response
    when mode=4   --- Connection 中
     and O.Connect_Response.num=num
    do  P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
mode:=20;   --- 二次局モード
window:=windowSize;
Vr:=0;
Vs:=0;
retryCount:=0;
remoteBusy:=false
    destination XMIT
    output {P.Send,
    L.WD_timer,
    L.NC_Parameters,
    L.AC_Parameters}
  end
  conn21 is
    source CONN
    input O.Disconnect_Request
    when mode=4   --- Connection 中
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:dm:rsp:F";
mode:=0   --- モードを初期状態にする
    destination NDM
    output P.Send
  end
  conn22 is
    source CONN
    input P.Recv
    when mode=4   --- Connection 中
     and P.Recv.msg="x:x:x:x"
    destination CONN
  end

  --- for SETUP
  conn30 is
    source SETUP
    input L.F_timer
    when mode=4   --- Connection 中
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and retryCount<N3
    do  P.Send_Con.msg:="u:snrm:cmd:P";
P.Send_Con.val:=ca;
P.Send_Con.va2:=dest;
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination SETUP
    output {P.Send_Con,
    L.F_timer,
    L.Perform_Random_Backoff}
  end
  conn31 is
    source SETUP
    input L.F_timer
    when mode=4   --- Connection 中
     and L.F_timer.num=num
```

```
     and L.F_timer.msg="expired"
     and (retryCount>N3 or retryCount=N3)
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0      --- モードを初期状態にする
    destination NDM
    output O.Disconnect_Indication
  end
  conn32 is
    source SETUP
    input P.Recv_Con
    when mode=4   --- Connection中
     and P.Recv_Con.msg="u:snrm:cmd:P"
     and P.Recv_Con.va2>NA
    do  P.Send.msg:="u:ua:rsp:f";
        L.F_timer.num:=num;
L.F_timer.msg:="stop";
        L.WD_timer.num:=num;
L.WD_timer.msg:="start";
mode:=20;   --- 二次局モード
window:=windowSize;
Vr:=0;
Vs:=0;
retryCount:=0;
remoteBusy:=false
    destination XMIT
    output {O.Connect_Confirm,
    P.Send,
    L.F_timer,
    L.WD_timer,
    L.NC_Parameters,
    L.AC_Parameters}
  end
  ---conn33 is
  ---  source SETUP
  ---  input P.Recv_Con
  ---  when mode=4   --- Connection中
  ---   and P.Recv_Con.msg="u:snrm:cmd:P"
  ---   and P.Recv_Con.val>NA
  ---  destination SETUP
  ---end
  conn34 is
    source SETUP
    input P.Recv_Con
    when mode=4   --- Connection中
     and P.Recv_Con.msg="u:snrm:cmd:P"
     and P.Recv_Con.va2<NA
    destination SETUP
  end
  conn35 is
    source SETUP
    input P.Recv
    when mode=4   --- Connection中
     and P.Recv.msg="u:ua:rsp:F"
    do  P.Send.msg:="s:rr:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="restart";
mode:=10;   --- 一次局モード
window:=windowSize;
Vr:=0;
Vs:=0;
retryCount:=0;
remoteBusy:=false
    destination XMIT
```

```
    output {O.Connect_Confirm,
    P.Send,
    L.F_timer,
    L.NC_Parameters,
    L.AC_Parameters}
  end
  conn36 is
    source SETUP
    input P.Recv
    when mode=4   --- Connection中
     and P.Recv.msg="u:dm:rsp:x"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer}
  end
  conn37 is
    source SETUP
    input P.Recv
    when mode=4   --- Connection中
     and P.Recv.msg="u:disc:cmd:x"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer}
  end
  conn38 is
    source SETUP
    input P.Recv
    when mode=4   --- Connection中
     and P.Recv.msg="x:x:x:x"
    destination SETUP
  end
  --- コネクション設定の手続き ここまで


  --- 一次局の状態遷移
  --- for XMIT
  npa10 is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="i:cmd:Nr:Ns:P";
--- P.Send.msg:="i:cmd:Nr:Ns:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=windowSize
```

```
    destination RECV
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
  ---npa11a is
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=10   --- 一次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window>1
  ---   and AckRequired=true
  ---  do  P.Send.msg:="u:ui:cmd: :not P";
  ---   P.Send.msg:="u:ui:cmd: :not P:data";
  ---   L.P_timer.num:=num;
  ---   L.P_timer.msg:="stop";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination RECV
  ---  output {P.Send,
  ---     L.P_timer,
  ---     L.F_timer}
  ---end
  npa11b is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
     and AckRequired=true
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
   AckRequired:=false;
   window:=windowSize
    destination RECV
    output {P.Send,
       L.P_timer,
       L.F_timer}
  end
  npa12 is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
     and AckRequired=false
    do  P.Send.msg:="u:ui:cmd:P";
--- P.Send.msg:="u:ui:cmd:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
```

```
window:=windowSize
    destination RECV
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
  ---npa13 is
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=10   --- 一次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="DATA"
  ---   and remoteBusy=false
  ---   and window>1
  ---  do  P.Send.msg:="i:cmd:Nr:Ns:not P"
  ---   P.Send.msg:="i:cmd:Nr:Ns:not P:data";
  --- Store_Vs:="data";
  --- Ack_Vs:=false;
  --- Vs:=(Vs+1) mod 8;
  --- AckRequired:=false;
  --- window:=window-1
  ---  destination XMIT
  ---  output P.Send
  ---end
  ---npa14 is
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=10   --- 一次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window>1
  ---  do  P.Send.msg:="u:ui:cmd:not P";
  ---   P.Send.msg:="u:ui:cmd:not P:data";
  --- window:=window-1
  ---  destination XMIT
  ---  output P.Send
  ---end
  npa15 is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window=1
    do  P.Send.msg:="i:cmd:Nr:Ns:P";
--- P.Send.msg:="i:cmd:Nr:Ns:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=windowSize
    destination RECV
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
  ---npa16a is
```

```
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=10   --- 一次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window=1
  ---   and AckRequired=true
  ---  do  P.Send.msg:="u:ui:cmd: :not P";
  ---   P.Send.msg:="u:ui:cmd: :not P:data";
  ---   L.P_timer.num:=num;
  ---   L.P_timer.msg:="stop";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination RECV
  ---  output {P.Send,
  ---          L.P_timer,
  ---     L.F_timer}
  ---end
  npa16b is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=true
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
        L.F_timer.num:=num;
L.F_timer.msg:="start";
   AckRequired:=false;
   window:=windowSize
    destination RECV
    output {P.Send,
            L.P_timer,
       L.F_timer}
  end
  npa17 is
    source XMIT
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=false
    do  P.Send.msg:="u:ui:cmd:P";
--- P.Send.msg:="u:ui:cmd:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
window:=windowSize
    destination RECV
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
```

```
  npa18 is
    source XMIT
    input O.Reset_Request
    when mode=10   --- 一次局モード
     and O.Reset_Request.num=num
    do  P.Send.msg:="u:snrm:cmd:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination RESET
    output {P.Send,
    L.P_timer,
    L.F_timer,
    L.WMT_Delay}
  end
  npa19 is
    source XMIT
    input O.Disconnect_Request
    when mode=10   --- 一次局モード
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:disc:cmd:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.P_timer,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  npa20 is
    source XMIT
    input L.Local_Busy_Detected
    when mode=10   --- 一次局モード
    destination BUSY
  end
  npa21 is
    source XMIT
    input L.P_timer
    when mode=10   --- 一次局モード
     and L.P_timer.num=num
     and L.P_timer.msg="expired"
    do  P.Send.msg:="s:rr:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.F_timer}
  end

  --- for RECV
  npb10 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="i:rsp:Ns:Nr:not F"
    do  Vr:=(Vr+1) mod 8;
AckRequired:=true
```

```
    destination RECV
    output {O.Data_Indication,
    L.Update_Nr_Received}
  end
  npb11 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="i:rsp:Ns:Nr:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start";
Vr:=(Vr+1) mod 8;
AckRequired:=true
    destination XMIT
    output {O.Data_Indication,
    L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  npb12 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:ui:rsp:not F"
    destination RECV
    output O.Unidata_Indication
  end
  npb13 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:ui:rsp:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start"
    destination XMIT
    output {O.Unidata_Indication,
    L.F_timer,
    L.P_timer}
  end
  npb14 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:xid:rsp:F"
    do  P.Send.msg:="s:rr:cmd:P:Nr";
L.F_timer.num:=num;
L.F_timer.msg:="start";
AckRequired:=false
    destination RECV
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npb15 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="i:rsp:uNs:Nr:not F"
    destination RECV
```

```
    output L.Update_Nr_Received
  end
  npb16 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="i:rsp:uNs:Nr:F"
    do  P.Send.msg:="s:rr:cmd:P:Nr";
L.F_timer.num:=num;
L.F_timer.msg:="start";
AckRequired:=false
    destination RECV
    output {P.Send,
            L.F_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  ---npb17 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and P.Recv.msg="i:rsp:uNs:Nr:F"
  ---  do  P.Send.msg:="s:rej:cmd:P:Nr";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- AckRequired:=false
  ---  destination RECV
  ---  output {P.Send,
  ---          L.F_timer,
  ---          L.Update_Nr_Received,
  ---          L.WMT_Delay}
  ---end
  npb18 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="i:rsp:Ns:uNr:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="start";
Vr:=(Vr+1) mod 8;
   AckRequired:=false
    destination RECV
    output {O.Data_Indication,
       L.F_timer,
       L.Update_Nr_Received,
       L.resend_rejected_frames}
  end
  npb19 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rr:rsp:uNr:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="start";
remoteBusy:=false
    destination RECV
    output {L.F_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  npb20 is
    source RECV
    input P.Recv
```

```
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rej:rsp:Nr:F"
     and remoteBusy=false
    do  L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {L.F_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  npb21 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rej:rsp:Nr:F"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.F_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  npb22 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="i:rsp:iNs:Nr:not F"
      or P.Recv.msg="i:rsp:Ns:iNr:not F"
      or P.Recv.msg="s:x:rsp:iNr:not F")
    do  O.Reset_Indication.msg:="local";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
xmitFlag:=false
    destination RESET_WAIT
    output {O.Reset_Indication,
    L.F_timer}
  end
  ---npb23 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and (P.Recv.msg="i:rsp:iNs:Nr:not F"
  ---    or P.Recv.msg="i:rsp:Ns:iNr:not F"
  ---    or P.Recv.msg="s:x:rsp:iNr:not F")
  ---  destination PCLOSE_WAIT
  ---end
  npb24 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="i:rsp:iNs:Nr:F"
      or P.Recv.msg="i:rsp:Ns:iNr:F"
      or P.Recv.msg="s:x:rsp:iNr:F")
    do  O.Reset_Indication.msg:="local";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
xmitFlag:=true
    destination RESET_WAIT
    output {O.Reset_Indication,
    L.F_timer}
```

```
  end
  ---npb25 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and (P.Recv.msg="i:rsp:iNs:Nr:F"
  ---    or P.Recv.msg="i:rsp:Ns:iNr:F"
  ---    or P.Recv.msg="s:x:rsp:iNr:F")
  ---  do  P.Send.msg:="u:disc:cmd:P";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- retryCount:=0
  ---    destination PCLOSE
  ---  output {P.Send,
  ---     L.F_timer,
  ---     L.WMT_Delay,
  ---     L.RB_Data}
  ---end
  npb26 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rr:rsp:Nr:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start";
remoteBusy:=false
    destination XMIT
    output {L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  npb27 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:srej:rsp:Nr:F"
     and remoteBusy=false
    do  L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {L.F_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frame}
  end
  npb28 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:srej:rsp:Nr:F"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.F_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  npb29 is
    source RECV
```

```
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rnr:rsp:Nr:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start";
remoteBusy:=true
    destination XMIT
    output {L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  npb30 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:frmr:rsp:F"
    do  O.Reset_Indication.msg:="local";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
xmitFlag:=true
    destination RESET_WAIT
    output {O.Reset_Indication,
    L.F_timer}
  end
  ---npb31 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and P.Recv.msg="u:frmr:rsp:F"
  ---  do  L.F_timer.num:=num;
  ---      L.F_timer.msg:="stop";
  ---      L.P_timer.num:=num;
  ---      L.P_timer.msg:="start"
  ---  destination XMIT
  ---  output {L.F_timer,
  ---     L.P_timer}
  ---end
  ---npb32 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and P.Recv.msg="u:frmr:rsp:F"
  ---  do  P.Send.msg:="u:disc:cmd:P";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- retryCount:=0
  ---  destination PCLOSE
  ---  output {P.Send,
  ---     L.F_timer,
  ---     L.WMT_Delay,
  ---     L.RB_Data}
  ---end
  npb33 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:rd:rsp:F"
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
```

```
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  npb34 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:rnrm:rsp:F"
    do  O.Reset_Indication.msg:="remote";
L.F_timer.num:=num;
L.F_timer.msg:="stop"
    destination RESET_CHECK
    output {O.Reset_Indication,
    L.F_timer}
  end
  npb35 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:rnrm:rsp:F"
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  npb36 is
    source RECV
    input L.Local_Busy_Detected
    when mode=10   --- 一次局モード
    destination BUSY_WAIT
  end
  npb37 is
    source RECV
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and (retryCount<N2) and (retryCount/=N1)
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination RECV
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npb38 is
    source RECV
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and retryCount=N1
    do  P.Send.msg:="s:rr:cmd:Nr:P";
```

```
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination RECV
    output {O.Status_Indication,
    P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npb39 is
    source RECV
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and (retryCount>N2
      or  retryCount=N2)
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.ADC_Parameters}
  end
  npb40 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:x:not F"
    destination RECV
  end
  npb41 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:x:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start"
    destination XMIT
    output {L.F_timer,
    L.P_timer}
  end
  npb42 is
    source RECV
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="s:x:cmd:x"
       or  P.Recv.msg="i:cmd:x:x:x")
    do  O.Disconnect_Indication.msg:="PrimaryConflict";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end

  --- for PCLOSE_WAIT
  npc10 is
    source PCLOSE_WAIT
    input L.F_timer or P.Recv
```

```
    when P.Recv.msg="x:x:x:F"
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  npc11 is
    source PCLOSE_WAIT
    input P.Recv
    when P.Recv.msg="x:x:x:not F"
    destination PCLOSE_WAIT
  end
  npc12 is
    source PCLOSE_WAIT
    input P.Recv
    when P.Recv.msg="s:x:cmd:x"
      or P.Recv.msg="i:cmd:x:x:x"
    do  O.Disconnect_Indication.msg:="PrimaryConflict";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end

  --- for RESET_WAIT
  npd10 is
    source RESET_WAIT
    input O.Reset_Request
    when O.Reset_Request.num=num
     and xmitFlag=true
    do  P.Send.msg:="u:snrm:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RESET
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npd11 is
    source RESET_WAIT
    input O.Reset_Request
    when O.Reset_Request.num=num
     and xmitFlag=false
    do  L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RESET
    output L.F_timer
  end
  npd12 is
    source RESET_WAIT
    input O.Disconnect_Request
    when O.Disconnect_Request.num=num
    do  P.Send.msg:="u:disc:cmd:P";
```

```
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end

  --- for RESET_CHECK
  npe10 is
    source RESET_CHECK
    input O.Reset_Response
    when mode=10   --- 一次局モード
     and O.Reset_Response.num=num
    do  P.Send.msg:="u:snrm:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
Vr:=0; Vs:=0;
window:=windowSize;
remoteBusy:=false;
retryCount:=0
    destination RESET
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npe11 is
    source RESET_CHECK
    input O.Disconnect_Request
    when mode=10   --- 一次局モード
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end

  --- for RESET
  npf10 is
    source RESET
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:ua:rsp:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start";
Vr:=0; Vs:=0;
window:=windowSize;
remoteBusy:=false;
retryCount:=0
    destination XMIT
    output {O.Reset_Confirm,
    L.F_timer,
    L.P_timer}
  end
```

```
  npf11 is
    source RESET
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:dm:rsp:F"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end
  npf12 is
    source RESET
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:x:x"
    destination RESET
  end
  npf13 is
    source RESET
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:rsp:F"
    do  P.Send.msg:="u:snrm:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RESET
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npf14 is
    source RESET
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and retryCount<N3
    do  P.Send.msg:="u:snrm:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination RESET
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npf15 is
    source RESET
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and (retryCount>N3
      or  retryCount=N3)
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.ADC_Parameters}
  end
```

```
  --- for BUSY
  npg10 is
    source BUSY
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="i:cmd:Nr:Ns:not P";
--- P.Send.msg:="i:cmd:Nr:Ns:not P:data";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=window-1
    destination BUSY
    output P.Send
  end
  npg11 is
    source BUSY
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="u:ui:cmd:not P";
--- P.Send.msg:="u:ui:cmd:not P:data";
window:=window-1
    destination BUSY
    output P.Send
  end
  npg12 is
    source BUSY
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window=1
    do  P.Send.msg:="i:cmd:Nr:Ns:P";
--- P.Send.msg:="i:cmd:Nr:Ns:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
window:=windowSize;
AckRequired:=false
    destination BUSY_WAIT
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
  ---npg13a is
  ---  source BUSY
  ---  input O.Data_Request
  ---  when mode=10   --- 一次局モード
  ---   and O.Data_Request.num=num
```

```
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window=1
  ---   and AckRequired=true
  ---  do  P.Send.msg:="u:ui:cmd: :not P";
  ---   P.Send.msg:="u:ui:cmd: :not P:data";
  ---   L.P_timer.num:=num;
  ---   L.P_timer.msg:="stop";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination BUSY_WAIT
  ---  output {P.Send,
  ---     L.P_timer,
  ---     L.F_timer}
  ---end
  npg13b is
    source BUSY
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=true
    do  P.Send.msg:="s:rr:cmd:Nr:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
   AckRequired:=false;
   window:=windowSize
    destination BUSY_WAIT
    output {P.Send,
       L.P_timer,
       L.F_timer}
  end
  npg14 is
    source BUSY
    input O.Data_Request
    when mode=10   --- 一次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=false
    do  P.Send.msg:="u:ui:cmd:P";
--- P.Send.msg:="u:ui:cmd:P:data";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
window:=windowSize
    destination BUSY_WAIT
    output {P.Send,
    L.P_timer,
    L.F_timer}
  end
  npg15 is
    source BUSY
    input O.Disconnect_Request
    when mode=10   --- 一次局モード
```

```
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:disc:cmd:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.P_timer,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  npg16 is
    source BUSY
    input L.Local_Busy_Cleared
    when mode=10   --- 一次局モード
    do  P.Send.msg:="s:rr:cmd:P";
L.P_timer.num:=num;
L.P_timer.msg:="stop";
L.P_timer.num:=num;
L.F_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.P_timer,
    L.F_timer,
    L.WMT_Delay}
  end
  npg17 is
    source BUSY
    input L.P_timer
    when mode=10   --- 一次局モード
     and L.P_timer.num=num
     and L.P_timer.msg="expired"
    do  P.Send.msg:="s:rnr:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end

  --- for BUSY_WAIT
  nph10 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="i:rsp:Ns:Nr:not F"
       or  P.Recv.msg="i:rsp:uNs:Nr:not F")
    destination BUSY_WAIT
    output L.Update_Nr_Received
  end
  nph11 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="i:rsp:Ns:Nr:F"
       or  P.Recv.msg="i:rsp:uNs:Nr:F")
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
```

```
L.P_timer.msg:="start"
    destination BUSY
    output {L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  nph12 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:ui:rsp:not F"
    destination BUSY_WAIT
  end
  nph13 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and (P.Recv.msg="u:ui:rsp:F"
      or  P.Recv.msg="u:xid:rsp:F")
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start"
    destination BUSY
    output {L.F_timer,
    L.P_timer}
  end
  nph14 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rr:rsp:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start";
remoteBusy:=false
    destination BUSY
    output {L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  nph15 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rnr:rsp:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="strat";
remoteBusy:=true
    destination BUSY
    output {L.F_timer,
    L.P_timer,
    L.Update_Nr_Received}
  end
  nph16 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rej:rsp:F"
     and remoteBusy=false
```

```
    do  L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination BUSY_WAIT
    output {L.F_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nph17 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:rej:rsp:F"
     and remoteBusy/=false
    do  P.Send.msg:="s:rnr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.F_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nph18 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:srej:rsp:F"
     and remoteBusy=false
    do  L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination BUSY_WAIT
    output {L.F_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nph19 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="s:srej:rsp:F"
     and remoteBusy/=false
    do  P.Send.msg:="s:rnr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.F_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nph20 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:rd:rsp:F"
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
```

```
    L.RB_Data}
  end
  nph21 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:frmr:rsp:F"
    do  O.Reset_Indication.msg:="local";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
xmitFlag:=true
    destination RESET_WAIT
    output {O.Reset_Indication,
    L.F_timer}
  end
  ---nph22 is
  ---  source BUSY_WAIT
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and P.Recv.msg="u:frmr:rsp:F"
  ---  do  L.F_timer.num:=num;
  ---      L.F_timer.msg:="stop";
  ---      L.P_timer.num:=num;
  ---      L.P_timer.msg:="start"
  ---  destination BUSY
  ---  output {L.F_timer,
  ---     L.P_timer}
  ---end
  ---nph23 is
  ---  source BUSY_WAIT
  ---  input P.Recv
  ---  when mode=10   --- 一次局モード
  ---   and P.Recv.msg="u:frmr:rsp:F"
  ---  do  P.Send.msg:="u:disc:cmd:P";
  ---   L.F_timer.num:=num;
  ---   L.F_timer.msg:="start";
  --- retryCount:=0
  ---  destination PCLOSE
  ---  output {P.Send,
  ---     L.F_timer,
  ---     L.WMT_Delay,
  ---     L.RB_Data}
  ---end
  nph24 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:rnrm:rsp:F"
    do  O.Reset_Indication.msg:="remote";
L.F_timer.num:=num;
L.F_timer.msg:="stop"
    destination RESET_CHECK
    output {O.Reset_Indication,
    L.F_timer}
  end
  nph25 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="u:frmr:rsp:F"
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
```

```
retryCount:=0
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay,
    L.RB_Data}
  end
  nph26 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:x:not F"
    destination BUSY_WAIT
  end
  nph27 is
    source BUSY_WAIT
    input P.Recv
    when mode=10   --- 一次局モード
     and P.Recv.msg="x:x:x:F"
    do  L.F_timer.num:=num;
L.F_timer.msg:="stop";
L.P_timer.num:=num;
L.P_timer.msg:="start"
    destination BUSY
    output {L.F_timer,
    L.P_timer}
  end
  nph28 is
    source BUSY_WAIT
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and retryCount<N2
     and retryCount/=N1
    do  P.Send.msg:="s:rnr:cmd:Nr:P";
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination BUSY_WAIT
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  nph29 is
    source BUSY_WAIT
    input L.F_timer
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and retryCount=N1
    do  P.Send.msg:="s:rnr:cmd:Nr:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination BUSY_WAIT
    output {O.Status_Indication,
    P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  nph30 is
    source BUSY_WAIT
    input L.F_timer
```

```
    when mode=10   --- 一次局モード
     and L.F_timer.num=num
     and L.F_timer.msg="expired"
     and (retryCount>N2
      or  retryCount=N2)
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.ADC_Parameters}
  end

  --- for PCLOSE
  npi10 is
    source PCLOSE
    input P.Recv
    when P.Recv.msg="u:ua:rsp:F"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end
  npi11 is
    source PCLOSE
    input P.Recv
    when P.Recv.msg="u:dm:rsp:F"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end
  ---npi12 is
  ---  source PCLOSE
  ---  input P.Recv
  ---  when P.Recv.msg="u:dm:rsp:F"
  ---  destination PCLOSE
  ---end
  npi13 is
    source PCLOSE
    input P.Recv
    when P.Recv.msg="s:x:cmd:x"
      or P.Recv.msg="i:cmd:x:x:x"
    do  O.Disconnect_Indication.msg:="disconnect";
L.F_timer.num:=num;
L.F_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.F_timer,
    L.ADC_Parameters}
  end
  npi14 is
    source PCLOSE
    input L.F_timer
    when L.F_timer.num=num
```

```
     and L.F_timer.msg="expired"
     and retryCount<N3
    do  P.Send.msg:="u:disc:cmd:P";
L.F_timer.num:=num;
L.F_timer.msg:="start";
retryCount:=retryCount+1
    destination PCLOSE
    output {P.Send,
    L.F_timer,
    L.WMT_Delay}
  end
  npi15 is
    source PCLOSE
    input L.F_timer
    when L.F_timer.num=num
     and L.F_timer.msg="expired"
     and (retryCount>N3 or retryCount=N3)
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.ADC_Parameters}
  end
  --- 一次局の状態遷移 ここまで


  --- 二次局の状態遷移
  --- for XMIT
  nsa10 is
    source XMIT
    input O.Data_Request and O.Pending_Data_Requests
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Pending_Data_Requests.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="i:rsp:Nr:Ns:not F";
--- P.Send.msg:="i:rsp:Nr:Ns:not F:data";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=window-1
    destination XMIT
    output P.Send
  end
  nsa11 is
    source XMIT
    input O.Data_Request and O.Pending_Data_Requests
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Pending_Data_Requests.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="u:ui:rsp:not F";
--- P.Send.msg:="u:ui:rsp:not F:data";
window:=window-1
    destination XMIT
    output P.Send
  end
  nsa12 is
```

```
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="i:rsp:Nr:Ns:F";
--- P.Send.msg:="i:rsp:Nr:Ns:F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=windowSize
    destination RECV
    output {P.Send,
    L.WD_timer}
  end
  ---nsa13a is
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=20   --- 二次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window>1
  ---   and AckRequired=true
  ---  do  P.Send.msg:="u:ui:rsp: :not F";
  ---   P.Send.msg:="u:ui:rsp: :not F:data";
  ---   L.WD_timer.num:=num;
  ---   L.WD_timer.msg:="start";
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination RECV
  ---  output {P.Send,
  ---     L.WD_timer}
  ---end
  nsa13b is
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
     and AckRequired=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
   AckRequired:=false;
   window:=windowSize
    destination RECV
    output {P.Send,
       L.WD_timer}
  end
  nsa14 is
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
```

```
     and remoteBusy=false
     and window>1
     and AckRequired=false
    do P.Send.msg:="u:ui:rsp:F";
--- P.Send.msg:="u:ui:rsp:F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
window:=windowSize
    destination RECV
    output {P.Send,
    L.WD_timer}
  end
  nsa15 is
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window=1
    do  P.Send.msg:="i:rsp:Nr:Ns:F";
--- P.Send.msg:="i:rsp:Nr:Ns:F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=windowSize
    destination RECV
    output {P.Send,
    L.WD_timer}
  end
  ---nsa16a is
  ---  source XMIT
  ---  input O.Data_Request
  ---  when mode=20   --- 二次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window=1
  ---   and AckRequired=true
  ---  do  P.Send.msg:="u:ui:rsp: :not F";
  ---   P.Send.msg:="u:ui:rsp: :not F:data";
  ---   L.WD_timer.num:=num;
  ---   L.WD_timer.msg:="start";
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination RECV
  ---  output {P.Send,
  ---     L.WD_timer}
  ---end
  nsa16b is
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
```

```
L.WD_timer.msg:="start";
   AckRequired:=false;
   window:=windowSize
    destination RECV
    output {P.Send,
       L.WD_timer}
  end
  nsa17 is
    source XMIT
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
     and AckRequired=false
    do  P.Send.msg:="u:ui:rsp:F";
--- P.Send.msg:="u:ui:rsp:F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
window:=windowSize
    destination RECV
    output {P.Send,
    L.WD_timer}
  end
  nsa18 is
    source XMIT
    input O.Disconnect_Request
    when mode=20   --- 二次局モード
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:rd:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output {P.Send,
    L.WD_timer,
    L.RB_Data}
  end
  nsa19 is
    source XMIT
    input O.Reset_Request
    when mode=20   --- 二次局モード
     and O.Reset_Request.num=num
    do  P.Send.msg:="u:rnrm:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
retryCount:=0
    destination RESET
    output {P.Send,
    L.WD_timer}
  end
  nsa20 is
    source XMIT
    input L.Local_Busy_Detected
    when mode=20   --- 二次局モード
    destination BUSY
  end

  --- for RECV
  nsb10 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
```

```
     and P.Recv.msg="i:cmd:Ns:Nr:not P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=(Vr+1) mod 8;
AckRequired:=true
    destination RECV
    output {O.Data_Indication,
    L.WD_timer,
    L.Update_Nr_Received}
  end
  nsb11 is
    source RECV
    input O.Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Data_Requests.num=num
     and P.Recv.msg="i:cmd:Ns:Nr:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
Vr:=(Vr+1) mod 8;
AckRequired:=false
    destination XMIT
    output {O.Data_Indication,
    L.WD_timer,
    L.Update_Nr_Received}
  end
  nsb12a is
    source RECV
    input O.Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Data_Requests.num=num
     and P.Recv.msg="i:cmd:Ns:Nr:P"
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=(Vr+1) mod 8
    destination RECV
    output {O.Data_Indication,
    P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb12b is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="i:cmd:Ns:Nr:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=(Vr+1) mod 8
    destination RECV
    output {O.Data_Indication,
    P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb13 is
    source RECV
    input P.Recv
```

```
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:ui:cmd:not P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {O.Unitdata_Indication,
    L.WD_timer}
  end
  nsb14 is
    source RECV
    input O.Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Data_Requests.num=num
     and P.Recv.msg="u:ui:cmd:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop"
    destination XMIT
    output {O.Unitdata_Indication,
    L.WD_timer}
  end
  nsb15a is
    source RECV
    input O.No_Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.No_Pending_Data_Requests.num=num
     and P.Recv.msg="u:ui:cmd:P"
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
AckRequired:=false
    destination RECV
    output {O.Unitdata_Indication,
    P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsb15b is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:ui:cmd:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
AckRequired:=false
    destination RECV
    output {O.Unitdata_Indication,
    P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsb16 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:xid:cmd:P"
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
AckRequired:=false
    destination RECV
```

```
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsb17 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:test:cmd:P"
    do  P.Send.msg:="u:test:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  ---nsb18 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=20   --- 二次局モード
  ---   and P.Recv.msg="u:test:cmd:P"
  ---  do  P.Send.msg:="s:rr:rsp:Nr:F";
  ---      L.WD_timer.num:=num;
  ---      L.WD_timer.msg:="start"
  ---  destination RECV
  ---  output {P.Send,
  ---     L.WD_timer,
  ---     L.WMT_Delay}
  ---end
  nsb19 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="i:cmd:uNs:Nr:not P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {L.WD_timer,
            L.Update_Nr_Received}
  end
  nsb20 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="i:cmd:uNs:Nr:P"
    do  P.Send.msg:="s:rr:rsp:F:Nr";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  ---nsb21 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=20   --- 二次局モード
  ---   and P.Recv.msg="i:cmd:uNs:Nr:P"
  ---  do  P.Send.msg:="s:rej:rsp:F:Nr";
  ---      L.WD_timer.num:=num;
  ---      L.WD_timer.msg:="start"
```

```
  ---  destination RECV
  ---  output {P.Send,
  ---     L.WD_timer,
  ---     L.WMT_Delay,
  ---     L.Update_Nr_Received}
  ---end
  nsb22 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="i:cmd:Ns:uNr:P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=(Vr+1) mod 8
    destination RECV
    output {0.Data_Indication,
    L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nsb23 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rr:cmd:Ns:uNr:P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=false
    destination RECV
    output {L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nsb24 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:Nr:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nsb25 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:Nr:P"
     and remoteBusy/=false
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb26 is
    source RECV
```

```
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="i:cmd:iNs:Nr:not P"
      or  P.Recv.msg="i:cmd:Ns:iNr:not P"
      or  P.Recv.msg="s:x:cmd:iNr:not P")
    destination ERROR
    output O.Prepare_FRMR_Response
  end
  nsb27 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="i:cmd:iNs:Nr:P"
      or  P.Recv.msg="i:cmd:Ns:iNr:P"
      or  P.Recv.msg="s:x:cmd:iNr:P")
    do  P.Send.msg:="u:frmr:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsb28 is
    source RECV
    input O.Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Data_Requests.num=num
     and P.Recv.msg="s:rr:cmd:Nr:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
remoteBusy:=false
    destination XMIT
    output {L.WD_timer,
    L.Update_Nr_Received}
  end
  nsb29a is
    source RECV
    input O.No_Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.No_Pending_Data_Requests.num=num
     and P.Recv.msg="s:rr:cmd:Nr:P"
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=false
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb29b is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rr:cmd:Nr:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=false
```

```
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb30 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:Nr:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frame}
  end
  nsb31 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:Nr:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
            L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb32 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rnr:cmd:Nr:P"
    do  P.Send.msg:="s:rr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=true
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsb33 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:disc:cmd:P"
    do  O.Disconnect_Indication.msg:="disconnect";
P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    P.Send,
    L.WD_timer,
```

```
    L.WMT_Delay,
    L.RB_Data,
    L.ADC_Parameters}
  end
  nsb34 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="Unknown Frame"
    do  P.Send.msg:="u:frmr:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsb35 is
    source RECV
    input L.Local_Busy_Detected
    when mode=20   --- 二次局モード
    destination BUSY_WAIT
  end
  ---nsb36 is
  ---  source RECV
  ---  input P.Recv
  ---  when mode=20   --- 二次局モード
  ---   and P.Recv.msg="U:snrm:cmd:P"
  ---  do  O.Reset_Indication.msg:="remote";
  ---      L.WD_timer.num:=num;
  ---      L.WD_timer.msg:="stop"
  ---  destination RESET_CHECK
  ---  output {O.Reset_Indication,
  ---     L.WD_timer}
  ---end
  nsb37 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="U:snrm:cmd:P"
    do  P.Send.msg:="u:rd:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output {P.Send,
       L.WD_timer,
       L.WMT_Delay}
  end
  nsb38 is
    source RECV
    input L.WD_timer
    when mode=20   --- 二次局モード
     and L.WD_timer.num=num
     and L.WD_timer.msg="expired"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {O.Status_Indication,
    L.WD_timer}
  end
  ---nsb39 is
  ---  source RECV
  ---  input L.WD_timer
```

```
  ---  when mode=20   --- 二次局モード
  ---   and L.WD_timer.num=num
  ---   and L.WD_timer.msg="expired"
  ---  do  O.Disconnect_Indication.msg:="NoRespose";
  ---      mode:=0   --- モードを初期状態にする
  ---  destination NDM
  ---  output {O.Disconnect_Indication,
  ---     L.ADC_Parameters,
  ---     L.RB_Data}
  ---end
  nsb40 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="s:x:rsp:x"
      or  P.Recv.msg="i:rsp:x:x:x")
    do  O.Disconnect_Indication.msg:="PrimaryConflict";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.ADC_Parameters}
  end
  nsb41 is
    source RECV
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="x:x:cmd:not P"
    destination RECV
  end

  --- for ERROR
  nsc10 is
    source ERROR
    input P.Recv
    when P.Recv.msg="x:x:x:P"
    do  P.Send.msg:="u:frmr:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsc11 is
    source ERROR
    input P.Recv
    when P.Recv.msg="u:disc:cmd:P"
    do  O.Disconnect_Indication.msg:="disconnect";
P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    P.Send,
    L.WD_timer,
    L.RB_Data,
    L.ADC_Parameters}
  end
  nsc12 is
```

```
    source ERROR
    input P.Recv
    when P.Recv.msg="u:dm:rsp:P"
    do  O.Disconnect_Indication.msg:="disconnect";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.RB_Data,
    L.ADC_Parameters}
  end
  nsc13 is
    source ERROR
    input P.Recv
    when P.Recv.msg="x:x:x:not P"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination ERROR
    output L.WD_timer
  end

  --- for RESET_CHECK
  nsd10 is
    source RESET_CHECK
    input O.Reset_Response
    when mode=20   --- 二次局モード
     and O.Reset_Response.num=num
    do  P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=0;
Vs:=0;
window:=windowSize;
remoteBusy:=false;
retryCount:=0
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.RB_Data}
  end
  nsd11 is
    source RESET_CHECK
    input O.Disconnect_Request
    when mode=20   --- 二次局モード
     and O.Disconnect_Request.num=num
    do  P.Send.msg:="u:rd:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end

  --- for RESET
  nse10 is
    source RESET
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:snrm:cmd:P"
    do  P.Send.msg:="u:ua:rsp:F";
```

```
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Vr:=0;
Vs:=0;
window:=windowSize;
remoteBusy:=false;
retryCount:=0
    destination RECV
    output {O.Reset_Confirm,
    P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nse11 is
    source RESET
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:dm:x:P"
    do  O.Disconnect_Indication.msg:="disconnect";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.RB_Data,
    L.ADC_Parameters}
  end
  nse12 is
    source RESET
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="x:x:x:x"
    destination RESET
  end
  nse13 is
    source RESET
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="x:x:cmd:P"
    do  P.Send.msg:="u:rnrm:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RESET
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nse14 is
    source RESET
    input L.WD_timer
    when mode=20   --- 二次局モード
     and L.WD_timer.num=num
     and L.WD_timer.msg="expired"
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.RB_Data,
    L.ADC_Parameters}
  end

  --- for BUSY
```

```
  nsf10 is
    source BUSY
    input O.Data_Request and O.Pending_Requests
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Pending_Requests.num=num
     and O.Data_Request.msg="DATA"
     and window>1
     and remoteBusy=false
    do  P.Send.msg:="i:rsp:Nr:Ns:not F";
--- P.Send.msg:="i:rsp:Nr:Ns:not F:data";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=window-1
    destination BUSY
    output P.Send
  end
  nsf11 is
    source BUSY
    input O.Data_Request and O.Pending_Requests
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Pending_Requests.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window>1
    do  P.Send.msg:="i:rsp:Nr:Ns:not F";
--- P.Send.msg:="i:rsp:Nr:Ns:not F:data";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=window-1
    destination BUSY
    output P.Send
  end
  nsf12a is
    source BUSY
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="DATA"
     and remoteBusy=false
     and window=1
    do  P.Send.msg:="i:rsp:Nr:Ns:not F";
--- P.Send.msg:="i:rsp:Nr:Ns:not F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
Store_Vs:="data";
Ack_Vs:=false;
Vs:=(Vs+1) mod 8;
AckRequired:=false;
window:=windowSize
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer}
  end
  ---nsf12b is
  ---  source BUSY
  ---  input O.Data_Request
  ---  when mode=20   --- 二次局モード
```

```
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="DATA"
  ---   and remoteBusy=false
  ---   and window=1
  ---  do  P.Send.msg:="s:rnr:Nr:F";
  ---   L.WD_timer.num:=num;
  ---   L.WD_timer.msg:="start";
  --- Store_Vs:="data";
  --- Ack_Vs:=false;
  --- Vs:=(Vs+1) mod 8;
  --- AckRequired:=false;
  --- window:=windowSize
  ---  destination BUSY_WAIT
  ---  output {P.Send,
  ---     L.WD_timer}
  ---end
  nsf13a is
    source BUSY
    input O.Data_Request
    when mode=20   --- 二次局モード
     and O.Data_Request.num=num
     and O.Data_Request.msg="UDATA"
     and remoteBusy=false
     and window=1
    do  P.Send.msg:="u:ui:rsp:not F";
--- P.Send.msg:="u:ui:rsp:not F:data";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
window:=windowSize
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer}
  end
  ---nsf13b is
  ---  source BUSY
  ---  input O.Data_Request
  ---  when mode=20   --- 二次局モード
  ---   and O.Data_Request.num=num
  ---   and O.Data_Request.msg="UDATA"
  ---   and remoteBusy=false
  ---   and window=1
  ---  do  P.Send.msg:="s:rnr:Nr:F";
  ---   L.WD_timer.num:=num;
  ---   L.WD_timer.msg:="start";
  --- window:=windowSize
  ---  destination BUSY_WAIT
  ---  output {P.Send,
  ---     L.WD_timer}
  ---end
  nsf14 is
    source BUSY
    input L.Local_Busy_Cleared
    when mode=20   --- 二次局モード
    do  P.Send.msg:="s:rr:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination RECV
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end

  --- BUSY_WAIT
```

```
  nsg10 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="i:cmd:Ns:Nr:not P"
      or  P.Recv.msg="i:cmd:uNs:Nr:not P")
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {L.WD_timer,
    L.Update_Nr_Received}
  end
  nsg11 is
    source BUSY_WAIT
    input O.Pending_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Requests.num=num
     and (P.Recv.msg="i:cmd:Ns:Nr:P"
      or  P.Recv.msg="i:cmd:uNs:Nr:P")
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop"
    destination BUSY
    output {L.WD_timer,
    L.Update_Nr_Received}
  end
  nsg12a is
    source BUSY_WAIT
    input O.No_Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.No_Pending_Data_Requests.num=num
     and (P.Recv.msg="i:cmd:Ns:Nr:P"
      or  P.Recv.msg="i:cmd:uNs:Nr:P")
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nsg12b is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="i:cmd:Ns:Nr:P"
      or  P.Recv.msg="i:cmd:uNs:Nr:P")
     and remoteBusy=true
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nsg13 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:ui:cmd:not P"
```

```
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output L.WD_timer
  end
  nsg14 is
    source BUSY_WAIT
    input O.Pending_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Requests.num=num
     and P.Recv.msg="u:ui:cmd:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop"
    destination BUSY
    output L.WD_timer
  end
  nsg15a is
    source BUSY_WAIT
    input O.No_Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.No_Pending_Data_Requests.num=num
     and P.Recv.msg="u:ui:cmd:P"
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsg15b is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:ui:cmd:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsg16 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:xid:cmd:P"
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsg17 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:test:cmd:P"
```

```
    do  P.Send.msg:="u:test:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  ---nsg18 is
  ---  source BUSY
  ---  input P.Recv
  ---  when mode=20   --- 二次局モード
  ---   and P.Recv.msg="u:test:cmd:P"
  ---  do  P.Send.msg:="u:rnr:rsp:F";
  ---      L.WD_timer.num:=num;
  ---      L.WD_timer.msg:="start"
  ---  destination BUSY_WAIT
  ---  output {P.Send,
  ---     WD_timer,
  ---     L.WMT_Delay}
  ---end
  nsg19 is
    source BUSY_WAIT
    input O.Pending_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.Pending_Requests.num=num
     and P.Recv.msg="s:rr:cmd:P"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
remoteBusy:=false
    destination BUSY
    output {L.WD_timer,
    L.Update_Nr_Received}
  end
  nsg20a is
    source BUSY_WAIT
    input O.No_Pending_Data_Requests and P.Recv
    when mode=20   --- 二次局モード
     and O.No_Pending_Data_Requests.num=num
     and P.Recv.msg="s:rr:cmd:P"
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=false
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsg20b is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rr:cmd:P"
     and remoteBusy=true
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=false
    destination BUSY_WAIT
    output {P.Send,
```

```
    L.WD_timer,
    L.WMT_Delay,
    L.Update_Nr_Received}
  end
  nsg21 is
    source BUSY_WAIT
    input P.Recv and L.Pending_Busy_Cleared
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rnr:cmd:P"
     and L.Pending_Busy_Cleared.bool=false
    do  P.Send.msg:="S:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
remoteBusy:=true
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsg22 is
    source BUSY_WAIT
    input P.Recv and L.Pending_Busy_Cleared
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rnr:cmd:P"
     and L.Pending_Busy_Cleared.bool=true
    do  remoteBusy:=true
    destination BUSY
  end
  nsg23 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:F"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nsg24 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:rej:cmd:F"
     and remoteBusy=true
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nsg25 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:srej:cmd:F"
     and remoteBusy=false
    do  L.WD_timer.num:=num;
```

```
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {L.WD_timer,
    L.Update_Nr_Received,
    L.resend_rejected_frames}
  end
  nsg26 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="s:srej:cmd:F"
     and remoteBusy=true
    do  P.Send.msg:="s:rnr:rsp:Nr:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.Update_Nr_Received,
    L.WMT_Delay}
  end
  nsg27 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:disc:cmd:P"
    do  O.Disconnect_Indication.msg:="disconnect";
P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    P.Send,
    L.WD_timer,
    L.WMT_Delay,
    L.RB_Data,
    L.ADC_Parameters}
  end
  nsg28 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:snrm:cmd:P"
    do  O.Reset_Indication.msg:="remote";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop"
    destination RESET_CHECK
    output {O.Reset_Indication,
    L.WD_timer}
  end
  nsg29 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="u:snrm:cmd:P"
    do  P.Send.msg:="u:rd:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
```

```
  end
  nsg30 is
    source BUSY_WAIT
    input L.WD_timer
    when mode=20   --- 二次局モード
     and L.WD_timer.num=num
     and L.WD_timer.msg="expired"
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {O.Status_Indication,
    L.WD_timer}
  end
  ---nsg31 is
  ---  source BUSY_WAIT
  ---  input L.WD_timer
  ---  when mode=20   --- 二次局モード
  ---   and L.WD_timer.num=num
  ---   and L.WD_timer.msg="expired"
  ---  do  O.Disconnect_Indication.msg:="disconnect";
  ---      mode:=0   --- モードを初期状態にする
  ---  destination NDM
  ---  output {O.Disconnect_Indication,
  ---     L.ADC_Parameters}
  ---end
  nsg32 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and (P.Recv.msg="s:x:rsp:x"
      or  P.Recv.msg="i:rsp:x:x:x")
    do  O.Disconnect_Indication.msg:="PrimaryConflict";
L.WD_timer.num:=num;
L.WD_timer.msg:="start";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.ADC_Parameters}
  end
  nsg33 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="x:x:cmd:not P"
    destination BUSY_WAIT
  end
  nsg34 is
    source BUSY_WAIT
    input P.Recv
    when mode=20   --- 二次局モード
     and P.Recv.msg="Unknown-Frame"
    do  P.Send.msg:="u:frmr:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination BUSY_WAIT
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end

  --- for SCLOSE
  nsh10 is
```

```
    source SCLOSE
    input P.Recv
    when P.Recv.msg="u:disc:cmd:P"
    do  O.Disconnect_Indication.msg:="disconnect";
P.Send.msg:="u:ua:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    P.Send,
    L.WD_timer,
    L.ADC_Parameters}
  end
  nsh11 is
    source SCLOSE
    input P.Recv
    when P.Recv.msg="u:dm:rsp:F"
    do  O.Disconnect_Indication.msg:="disconnect";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.ADC_Parameters}
  end
  ---nsh12 is
  ---  source SCLOSE
  ---  input P.Recv
  ---  when P.Recv.msg="u:dm:rsp:F"
  ---  destination SCLOSE
  ---end
  nsh13 is
    source SCLOSE
    input P.Recv
    when P.Recv.msg="s:x:rsp:x"
      or P.Recv.msg="i:rsp:x:x:x"
    do  O.Disconnect_Indication.msg:="disconnect";
L.WD_timer.num:=num;
L.WD_timer.msg:="stop";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
    L.WD_timer,
    L.ADC_Delay}
  end
  nsh14 is
    source SCLOSE
    input P.Recv
    when P.Recv.msg="x:x:x:P"
    do  P.Send.msg:="u:rd:rsp:F";
L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output {P.Send,
    L.WD_timer,
    L.WMT_Delay}
  end
  nsh15 is
    source SCLOSE
    input P.Recv
    when P.Recv.msg="x:x:x:x"
```

```
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="start"
    destination SCLOSE
    output L.WD_timer
  end
  nsh16 is
    source SCLOSE
    input L.WD_timer
    when L.WD_timer.num=num
     and L.WD_timer.msg="expired"
    do  O.Disconnect_Indication.msg:="disconnect";
mode:=0   --- モードを初期状態にする
    destination NDM
    output {O.Disconnect_Indication,
            L.ADC_Parameters}
  end
  --- 二次局の状態遷移 ここまで
end



--- 物理層での情報伝達を示すクラス
class PHLAYER
  field P:PHYSICAL
  state s0
  transition
  start is
    source init
    destination s0
  end
  t10 is
    source s0
    input P.Send_dCmd
    do  P.Recv_dCmd.val:=P.Send_dCmd.val;
P.Recv_dCmd.va2:=P.Send_dCmd.va2
    destination s0
    output P.Recv_dCmd
  end
  t11 is
    source s0
    input P.Send_dRsp
    do  P.Recv_dRsp.val:=P.Send_dRsp.val;
P.Recv_dRsp.msg:=P.Send_dRsp.msg
    destination s0
    output P.Recv_dRsp
  end
  t12 is
    source s0
    input P.Send_sRsp
    do  P.Recv_sRsp.val:=P.Send_sRsp.val;
P.Recv_sRsp.msg:=P.Send_sRsp.msg
    destination s0
    output P.Recv_sRsp
  end
  t13 is
    source s0
    input P.Send_Con
    do  P.Recv_Con.msg:=P.Send_Con.msg;
P.Recv_Con.val:=P.Send_Con.val;
P.Recv_Con.va2:=P.Send_Con.va2
    destination s0
    output P.Recv_Con
  end
```

```
  t14 is
    source s0
    input P.Send
    do  P.Recv.msg:=P.Send.msg
    destination s0
    output P.Recv
  end
end



--- IrLAP 層の内部タイマーを管理するクラス
class TIMER
  field L:LOCAL
  attribute count, num, kind:Int
  state OFFTIMER, ONTIMER
  transition
  start is
    source init
    do  count:=0;   --- タイマーのカウント数を保持する
kind:=0     --- タイマーの種類を示す (0 はどれも選択されてない状態)
    destination OFFTIMER
  end
  t1a is
    source OFFTIMER
    input L.slot_timer
    when L.slot_timer.num=num
     and L.slot_timer.msg="start"
    do  kind:=1
    destination ONTIMER
  end
  t1b is
    source OFFTIMER
    input L.query_timer
    when L.query_timer.num=num
     and L.query_timer.msg="start"
    do  kind:=2
    destination ONTIMER
  end
  t1c is
    source OFFTIMER
    input L.sense_timer
    when L.sense_timer.num=num
     and L.sense_timer.msg="start"
    do  kind:=3
    destination ONTIMER
  end
  t1d is
    source OFFTIMER
    input L.sniff_timer
    when L.sniff_timer.num=num
     and L.sniff_timer.msg="start"
    do  kind:=4
    destination ONTIMER
  end
  t1e is
    source OFFTIMER
    input L.sleep_timer
    when L.sleep_timer.num=num
     and L.sleep_timer.msg="start"
    do  kind:=5
    destination ONTIMER
  end
```

```
  t1f is
    source OFFTIMER
    input L.P_timer
    when L.P_timer.num=num
     and L.P_timer.msg="start"
    do  kind:=6
    destination ONTIMER
  end
  t1g is
    source OFFTIMER
    input L.F_timer
    when L.F_timer.num=num
     and L.F_timer.msg="start"
    do  kind:=7
    destination ONTIMER
  end
  t1h is
    source OFFTIMER
    input L.WD_timer
    when L.WD_timer.num=num
     and L.WD_timer.msg="start"
    do  kind:=8
    destination ONTIMER
  end

  t2 is
    source ONTIMER
    input not L.slot_timer
       or not L.query_timer
       or not L.sense_timer
       or not L.sniff_timer
       or not L.sleep_timer
       or not L.P_timer
       or not L.F_timer
       or not L.WD_timer
    do  count:=count+1
    destination ONTIMER
  end

  t3a is
    source ONTIMER
    input L.slot_timer
    when L.slot_timer.num=num
     and L.slot_timer.msg="stop"
     and kind=1
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3b is
    source ONTIMER
    input L.query_timer
    when L.query_timer.num=num
     and L.query_timer.msg="stop"
     and kind=2
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3c is
    source ONTIMER
    input L.sense_timer
    when L.sense_timer.num=num
```

```
     and L.sense_timer.msg="stop"
     and kind=3
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3d is
    source ONTIMER
    input L.sniff_timer
    when L.sniff_timer.num=num
     and L.sniff_timer.msg="stop"
     and kind=4
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3e is
    source ONTIMER
    input L.sleep_timer
    when L.sleep_timer.num=num
     and L.sleep_timer.msg="stop"
     and kind=5
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3f is
    source ONTIMER
    input L.P_timer
    when L.P_timer.num=num
     and L.P_timer.msg="stop"
     and kind=6
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3g is
    source ONTIMER
    input L.F_timer
    when L.F_timer.num=num
     and L.F_timer.msg="stop"
     and kind=7
    do  count:=0;
kind:=0
    destination OFFTIMER
  end
  t3h is
    source ONTIMER
    input L.WD_timer
    when L.WD_timer.num=num
     and L.WD_timer.msg="stop"
     and kind=8
    do  count:=0;
kind:=0
    destination OFFTIMER
  end

  t4a is
    source ONTIMER
    input L.P_timer
    when L.P_timer.num=num
     and L.P_timer.msg="restart"
     and kind=6
```

```
    do  count:=0
    destination ONTIMER
  end
  t4b is
    source ONTIMER
    input L.F_timer
    when L.F_timer.num=num
     and L.F_timer.msg="restart"
     and kind=7
    do  count:=0
    destination ONTIMER
  end

  t5a is
    source ONTIMER
    input not L.slot_timer
    when count=3 and kind=1
    do  L.slot_timer.num:=num;
L.slot_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.slot_timer
  end
  t5b is
    source ONTIMER
    input not L.query_timer
    when count=15 and kind=2
    do  L.query_timer.num:=num;
L.query_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.query_timer
  end
  t5c is
    source ONTIMER
    input not L.sense_timer
    when count=15 and kind=3
    do  L.sense_timer.num:=num;
L.sense_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.sense_timer
  end
  t5d is
    source ONTIMER
    input not L.sniff_timer
    when count=15 and kind=4
    do  L.sniff_timer.num:=num;
L.sniff_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.sniff_timer
  end
  t5e is
    source ONTIMER
    input not L.sleep_timer
    when count=15 and kind=5
    do  L.sleep_timer.num:=num;
L.sleep_timer.msg:="expired";
```

```
count:=0;
kind:=0
    destination OFFTIMER
    output L.sleep_timer
  end
  t5f is
    source ONTIMER
    input not L.P_timer
    when count=15 and kind=6
    do  L.P_timer.num:=num;
L.P_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.P_timer
  end
  t5g is
    source ONTIMER
    input not L.F_timer
    when count=15 and kind=7
    do  L.F_timer.num:=num;
L.F_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.F_timer
  end
  t5h is
    source ONTIMER
    input not L.WD_timer
    when count=15 and kind=8
    do  L.WD_timer.num:=num;
L.WD_timer.msg:="expired";
count:=0;
kind:=0
    destination OFFTIMER
    output L.WD_timer
  end
end
--- クラス記述部 ここまで


--- システム記述部
system IrLAP
  object irlap1(        --- 一次局になるデバイスオブジェクト
      num:=1;           --- 一次局のデバイスナンバー
      --- S:=3;
      --- s:=2;
      mediaBusy:=false;
      frameSent:=false;

      NA:=4;            --- 自局のデバイスアドレス
      sa:=-1;           --- 相手局のデバイスアドレス
      ca:=-1;           --- コネクションアドレス
      dest:=-1;         --- 自局のデバイスアドレスを格納する

      windowSize:=6;    ---
      Vr:=0;            --- Initialize_Connection_State
      Vs:=0;            --- コネクション状態変数を初期化する
      retryCount:=0;    ---
      remoteBusy:=false;---
```

```
    N1:=2;   --- ユーザが状態指示によって問題を警告されるべき値
    N2:=3;   --- 自然に切断されるのに必要な「送信」再送回数の最大値
    N3:=3;   --- コネクションリセット再送回数の最大値
    Store_Vs:="";
    Ack_Vs:=false;
    AckRequired:=false;
    xmitFlag:=false

    ):IRLAP

  object timer1(
    num:=1
    ):TIMER


  object irlap2(        --- 二次局になるデバイスオブジェクト
    num:=2;             --- 二次局のデバイスナンバー
    --- S:=3;
    --- s:=2;
    mediaBusy:=false;
    frameSent:=false;

    NA:=2;              --- 自局のデバイスアドレス
    sa:=-1;             --- 相手局のデバイスアドレス
    ca:=-1;             --- コネクションアドレス
    dest:=-1;           --- 自局のデバイスアドレスを格納する

    windowSize:=6;      ---
    Vr:=0;              --- Initialize_Connection_State
    Vs:=0;              --- コネクション状態変数を初期化する
    retryCount:=0;      ---
    remoteBusy:=false;---

    N1:=2;   --- ユーザが状態指示によって問題を警告されるべき値
    N2:=3;   --- 自然に切断されるのに必要な「送信」再送回数の最大値
    N3:=3;   --- コネクションリセット再送回数の最大値
    Store_Vs:="";
    Ack_Vs:=false;
    AckRequired:=false;
    xmitFlag:=false

    ):IRLAP

  object timer2(
    num:=2
    ):TIMER


  object bottom :PHLAYER
end
--- システム記述部 ここまで
--- End of IrLAP Procedure
```