〔非公開〕

TR－C－0017

通信ソフトウェアの仕様記述法について

芝本　尚樹　　西園　敏弘
NAOKI SHIBAMOTO　TOSHIHIRO NISHIZONO

1988. 9. 30.

ＡＴＲ通信システム研究所

通信ソフトウェアの
仕様記述法について

芝本　尚樹　　西園　敏弘
Naoki Shibamoto　Toshihiro Nishizono

１９８８．０９
ＡＴＲ通信システム研究所

概要

本報告書では，通信ソフトウェア要求仕様の記述法に関する研究結果を取りまとめる．先ず，要求仕様の記述法とその処理系を提案する．本記述法は，有限状態機械モデルを基本として記述項目を抽出し，知識処理で用いられるフレームモデルをその整理のために適用することにより，通信ソフトウェアの特徴を反映した形式的仕様記述を可能とする．この記述法を構築するために，ＣＣＩＴＴ勧告X.25を要求仕様の例として，その特徴の抽出と記述項目間の構造関係を明確化している．更に，知識処理言語ＡＲＴを用いて，上述の記述法の処理系の構成を示す．また，本報告書では，多重処理における複数プロセス間の相互作用仕様の抽象化手法を提案する．具体的には，ＣＣＩＴＴ勧告X.25における論理チャネル選択機能の２種の実現方式をＣＳＰ（Communicating Sequential Processes）で記述し，プロセス間での状態の制約に着目して新たなメッセージパッシング機構を用いることにより，両仕様を抽象化する．尚，以上の研究は，通信ソフトウェア開発の知的支援を目的とするＫＡＮＴ（Knowledge-based Automatic software generator for Neo-Telecommunications）システム構築のための基礎技術確立の一環として行ったものである．

通信ソフトウェアの仕様記述法について

目　次

１．はじめに

通信サービスは，ますます高度化・多様化しつつある．これを迅速かつ経済的に提供していくためには，通信ソフトウェアの生産性向上が重要な問題である．この解決のためには，通信に関する広範な知識を体系化し，通信ソフトウェアのライフサイクル全般に亙って，その知識を再利用することが求められる．現在，研究を進めているＫＡＮＴ（Knowl-ege-based Automatic softwre generator for Neo-Telecommunications）〔１〕は，通信ソフトウェアの要求分析，仕様化，設計，作成，保守の各段階に必要な知識を用いて，それぞれの段階における人間の知的活動を代替・支援することを目的とする．本報告書では，そのうちの仕様化段階における要求仕様の記述方法に関し，これまで研究を行ってきた結果をとりまとめる．

仕様化段階の知的活動の支援に必要な知識としては，既存システムの仕様自体に関する知識と，仕様化を効率的に進めるための知識とがある．これらの知識は，様々なシステムの開発を通して得られた経験の集大成ともいえる．これらの経験的知識を体系化するためには，通信ソフトウェアの仕様化において適切な情報を効果的に表現できる枠組み（モデル）を構築することが極めて重要である．このモデルに従って，仕様を記述することにより，その仕様を広範な分野で再利用することが可能となる．また，仕様化を効率的に進めるための既知の思考方法に関する知識を用いた支援をそのモデルに基づいて，整然と展開することができる．

このモデルへの一般的要求条件として，先ず，形式性があげられる．この形式性により，仕様を計算機上で取り扱うことが可能となり，プログラムへの自動変換，数学的検証，再利用のための統合的支援への道がひらける．また，仕様の再利用範囲の拡大と仕様化における思考支援のためには，通信ソフトウェアの特徴的問題を反映して，仕様情報を記述する能力と記述にあたっての抽象性も重要な要求条件である．このモデルの構築のためには，これらの要求条件をより具体化していく必要がある．このためには，従来，ソフトウェアの仕様記述に用いられてきた汎用モデルにおける形式性の上に，通信ソフトウェア仕様の記述を行い，その抽象化のための要求条件を抽出し，通信ソフトウェアの記述モデルの構成要素を定めることが有効である．

本報告書では，従来，汎用モデルとして，広く用いられてきた有限状態機械モデルを用いて，通信ソフトウェアの具体的仕様記述を行い，その問題点の指摘と抽象化を試みる．具体的には，先ず，ＣＣＩＴＴ勧告Ｘ．２５〔３〕およびＫＡＮＴプロジェクトにおけるソフトウェア作成実験の題材とした知的電話機の仕様を用いて，通信ソフトウェアの要求仕様の特徴を抽出する．次に，その仕様を有限状態機械モデルを基本として記述するとともに，その仕様に関する知識の記述項目を整理するために，知識処理で用いられているフレームモデルの適用方法を提案する．また，その方法を用いた処理系の基本構成を検討する．更に，有限状態機械モデルを基本とした上述の方法では，十分な記述が困難な例として，多重処理における複数プロセス間の相互作用を採り上げ，その抽象化手法を提案するとともに，その記述モデルへの要求条件を考察する．最後に，以上の結果をとりまとめ，今後の課題についてふれる．

２．要求記述の枠組み

要求の形式的記述法を得るために実際の通信ソフトウェア要求を分析してその特徴を抽出した〔２〕．まずその特徴について述べ，次にそれらの特徴を活かして要求をモデル化する方針を説明する．そしてその方針に沿って設定した具体的な要求記述項目を示す．

２．１　ソフトウェアの要求例

分析に使用した仕様は，ＣＣＩＴＴの勧告Ｘ．２５（『公衆データ網に専用線で接続されたパケットモードで動作する端末のためのデータ端末装置（ＤＴＥ）とデータ回線終端装置（ＤＣＥ）間のインタフェース』）〔３〕およびＫＡＮＴプロジェクトにおけるソフトウェア作成実験の生産物である『知的電話機』の要求仕様書である．これらの要求には次のような特徴がある．

①　『知的電話機』やＸ．２５の要求文書は，自然言語や図・表を用いて書かれた複合テキストである．

自然言語で書かれている要求の特徴は曖昧さを持っていることである．曖昧さとは，ある表現が複数の意味に解釈できるという多義性や，解釈に充分な情報が欠落しているという不完全性である．一方，人間の理解容易性を高めるような抽象性を記述内容に持たせようとすると，適切な形式言語がないために自然言語を用いざるを得ない場合もある．また，要求仕様書は自然言語で書かれている場合にもある程度の形式性を持っている．

通信ソフトウェアの要求仕様書の中では図・表は，ソフトウェア／ハードウェア構成・信号のやりとりのシーケンス・レコードフォーマット等の定義・説明に用いられている．これらの図・表による表現のうちの一部分は等価な言語表現に変換可能である．

②　仕様書として書くべき事柄の項目が部分的に決まっている．これは要求仕様書の中に見出せる形式性の１つである．

③　要求仕様書中の記述は，定義する対象物とその属性として捕らえることができる．ここでいう対象物とは，要求の中で用いる概念，システムが持つべき機能，システムが扱うデータ，システムの制御方法等である．これらの対象は個々に複数の属性項目と属性値を持っていて，その属性によってその対象を他の対象物と識別することができる．

④　対象物の属性がどのような値を持つことができるかにはある種の制約がある．つまり属性値に型の制約がある．

２．２　モデル化の方針

前節で述べた要求の特徴をから，その形式的表現法を検討する．

特徴①で述べたように自然言語による要求は曖昧性を持っている．要求の形式化には曖昧性を除去する必要がある．このためには自然言語による要求中に現れる対象を限定し，

個々の対象の特徴を明確にし，ある対象と他の対象を明確に識別することである．

対象は，その性質，構成要素の項目とその具体的内容によって識別される（特徴③）．対象を表現するための方法として，人工知能の分野で広く用いられているフレームモデルを用いることにする．即ち，要求の中の対象をフレーム，対象の性質や構成要素の項目をスロット，項目の具体的内容をスロット値に対応付ける．この形式化によって，入力のうち冗長な情報を取り除ける，入力に不足している情報が何かがわかるという２つのメリットがある．さらにソフトウェア要求の具体例を蓄積することにより通信分野に共通な項目を抽出することができるものと思われる．

さらに④の特徴を活かして，対象の属性項目に対して予め値の制約を設定しておくこととする．このような制約を利用して属性値をチェックすることによって，曖昧な要求が入力された際の多義性を除去したり，要求の中の矛盾を発見したりすることができ，要求の理解・検証に役立つ．

また要求の記述には最小性が望まれるが，これを満足するためには複数の対象が持つ似たような性質を１つに集約して，集約された１ヶ所だけで定義するとよい．このような集約は人間が入力する要求の中で既に行われている場合もあればそうでない場合もあり，後者の場合には入力要求の中の冗長な内容を見つけてそれを集約するという操作が必要である．これらの集約を形式的に表すためには，複数の対象を１つのクラスとしてまとめて共通の性質（属性項目および値）を１ヶ所で記述する．個々の対象はそのクラスから共通性質の継承を受けるとともに，対象毎に異なる特徴の差分だけを対象固有の記述をできるようにすればよい．クラスもまた対象であるので多重の階層化を行うことができる．モデル化に際してはこのような観点で具体的なクラス設計を行う．

本研究では，まず有限状態機械モデルに基づいて記述項目を設定してみて，その後有限状態機械モデルだけでは表し切れない要求情報の記述法を考察することにする．

２．３　通信ソフトウェア要求の記述項目

前節で述べたような方針に従って，通信ソフトウェア要求の記述項目を整理するためには，まず要求文書の中で何がどのような書かれ方をされているかを分類する必要がある．その観点としては，以下のものが考えられる．尚，その具体的内容を付録１に示す．

①文書構造

文書構造は，文書内の各文が何について記述しているかの分類である．要求中に現れる対象について記述されているものは対象記述，要求の対象の背景となる概念・知識について記述されているものは背景記述，メタ記述は記述方法そのものについて記述されている文で，文書の記述の目的や，文書内における相互参照関係等の記述の構造を説明する．

②定義対象

ソフトウェアの要求の中心となる記述で，何を定義するかの分類である．具体的には，機能・データ・システム構成要素等である．機能は入力データと出力データの値の関係やシステムの動作の時系列を表すが，要求文書の書き方によっては，このような機能が他の

対象と切り離されて明確に定義されているわけではない．データ定義はデータの構造・値域・存在場所等が定義される．

③性質および関係（スロット）

対象がどのような性質を持つか，およびある対象と別の対象がどのような関係にあるかについての記述である．対象１と対象２の間にある関係ｒは，対象１の属性ｒの値が対象２であると考えることができるので，ここでは性質と関係をまとめてスロットとして分類している．

④スロット値の性質

スロットの値の構造に関する分類である．スロット値の単純な要素として，記号・数値・真偽値等を持ち得る．また数値を値として持つ場合には，より細かい分類として，数値の次元・範囲（１点だけを表すか，離散的な複数点を表すか，連続領域を表すか）・精度（正確な値か曖昧さを含む値か）がある．スロット値は上のような単純型を値としてとることもあれば，上の要素を組合せた構造をとる場合もある．構造の種類としては，２つ以上の要素の『組』，集合，全順序を持つシーケンス，半順序を持つシーケンス，場合分け，ａｎｄやｏｒ等によって結ばれた論理的な組合せ等がある．

⑤その他

①～④以外の要求の性質で，複数の動作が相互に関係しながら同時に実行されるという多重性，複数の動作に公平に計算資源が与えられるという公平性，記述しているソフトウェアの外部に起因する不確実性，機能やデータ等に含まれている拡張性等がある．

以下では，比較的簡単な③・④の観点を用いて，要求文書における対象記述に絞り，実際のソフトウェア要求を分析することによって，要求として記述すべき項目を整理する．その手順は以下の通りである．

A．項目の洗い出しと分類

要求文書の中で使われている言葉を抜き出し，それらをｉ）アプリケーションに固有の定義対象（インスタンス），ii）アプリケーションに独立な定義対象（クラス），およびiii）スロットに分類する．例えば，発呼要求パケット・Ｐ２（ＤＴＥ待ち）状態等はインスタンス，データ・状態・メッセージ・状態遷移等はクラス，またメッセージ送信者・メッセージ受信者はメッセージの，前状態・次状態・伝送メッセージは状態遷移のスロットである．

B．属性値のタイプ抽出

Ａのiii）のスロットがとる値の型を制限する．例えば，状態遷移のスロットである前状態・次状態の値は状態クラスに属するものであり，伝送メッセージの値はメッセージクラスに属するものである，といったものである．

C．継承関係の抽出

定義対象間のクラスーインスタンス関係とクラスーサブクラス関係を見出す．例えば，Ｘ．２５パケットレベルにおける衝突状態およびタイムアウト状態は共に例外状態のインスタンスであり，例外状態は状態のサブクラスである．但し，ここではクラスーインスタンス関係，クラスーサブクラス関係の２つを特に区別せず，両方とも単なる継承関係とみなすことにする．

以上による記述項目（定義対象，スロット，およびスロット値の制約）を付録２に示す．これは通信ソフトウェアの動作の表現を有限状態機械モデルに基づいて状態・事象・状態遷移によって表している．Ｏｂｊｅｃｔは定義対象，型はそのＯｂｊｅｃｔ自身の型，，Ｓｌｏｔ値の制約は値として最低限必要な条件で，主に値として可能な型を書いている．

プリミティブな型として，<Boolean>，<Number>，<Symbol>，<Primitive-Action> およびそれらのプリミティブ型を要素とする構造，pair-of，list-of，set-of，partial-ordered-set-of，total-ordered-set-ofを使用可能とする．また既に述べたように，関係もスロットの１種であるとするので，関係の型を持つＯｂｊｅｃｔは他のＯｂｊｅｃｔのＳｌｏｔになり得る．

尚，以上の記述項目の抽出にあたって，無視した観点のうち，⑤の多重性の記述方法については，第４節で検討する．また，①の文書構造を利用した要求文書の理解，ある程度のソフトウェア構成を前提とした要求文書の文脈の解釈，およびそれを用いた②の機能の抽出や⑤の適用は将来の課題である．このためには，通信分野に関する膨大な知識を要し，具体的な内容を解明する仮定でその方法が明らかになると考えられる．

３．要求仕様の処理方法

ここでは，前節で述べた要求の記述項目を用いて記述した要求仕様に関する質問回答や検証等を目的とした処理系を説明する．

３．１　有限状態機械モデルに基づいた要求記述の処理系

２．３節で述べた記述項目をもとに，有限状態機械モデルに基づいた記述法の処理系について述べる．この処理系は，記述した要求の検証，記述内容に関する質問に対する回答，記述内容のシミュレーション実行等を行うことを目標とし，ＡＲＴ（Automated Reasoning Tool）のスキーマシステムをベースとした．プログラムはＡＲＴのルールとして記述する．

３．１．１　記述法の概要

２．３節で述べた要求の必要情報項目からさらに必要なものだけに絞ってクラスの設計を行った．その結果，下の１４種類のクラスを定義した．

クラスの種類　－

Object，Message，System-Parameter，Timer-Value，Data，Status，
Status-Transition，Event，Message-Transfer，Action，Procedure，
Exception-Event，Time-Out，Collision

これらのクラスは，各アプリケーションに共通のいわば通信の要求記述の核となる部分である．これらのクラスのインスタンスを記述することが，特定のアプリケーションに関する要求を記述することになる．これらのクラスが持つスロットおよび具体的なX．２５

の要求記述を，付録３のプログラムリストとして示す．

３．１．２　処理系機能の概要

処理系が持つ機能として質問回答，提示，検証，入力チェック，シミュレーション，設計の５つを考える．これら５つの全てを実現したわけではないが，それぞれの基本的枠組みを説明する．

◆質問回答は要求として定義された内容に関する利用者からの質問に答える機能である．

要求モデルの記述を，

$$\left.\begin{array}{l} p_1\ (c_{1\,1},\ c_{1\,2},\ \cdots\cdots,\ c_{1\,l_1}) \\ p_2\ (c_{2\,1},\ c_{2\,2},\ \cdots\cdots,\ c_{2\,l_2}) \\ \qquad\qquad \vdots \\ p_m\ (c_{m\,1},\ c_{m\,2},\ \cdots\cdots,\ c_{m\,l_m}) \end{array}\right\} \cdots (3.1.2.1)$$

とする．$p_i(1 \leqq i \leqq m)$ は述語，$c_{ij}(1 \leqq j \leqq l_i)$ は定数項である．

質問は以下の形に変換可能な種類のものだけに限定して考える．

$$\left.\begin{array}{l} p_{m+1}\ (v_{m+1\,1},\ v_{m+1\,2},\ \cdots\cdots,\ v_{m+1\,l_{m+1}}) \\ p_{m+2}\ (v_{m+2\,1},\ v_{m+2\,2},\ \cdots\cdots,\ v_{m+2\,l_{m+2}}) \\ \qquad\qquad \vdots \\ p_{m+n}\ (v_{m+n\,1},\ v_{m+n\,2},\ \cdots\cdots,\ v_{m+n\,l_{m+n}}) \end{array}\right\} \cdots (3.1.2.2)$$

$p_i(m+1 \leqq i \leqq m+n)$ は述語，$v_{ij}(1 \leqq j \leqq l_i)$ は定数項または変数項である．ここで，変数項に含まれる変数には，質問の回答となる変数と質問の回答を計算する過程の途中で値が束縛される変数とがある．質問回答とは，$p_i(1 \leqq i \leqq m+n)$から回答となる変数の値を求めること，またはその変数に対する制約を出来る限り簡素化（具体化）することである．

与えられる質問は，例えば『リスタート手順における正常シーケンスで“ＤＣＥリスタート確認”の前に伝送されるメッセージは何か？』，『メッセージ“接続完了”が伝送される事象によって遷移する先の状態の副状態は何か？』といったようなものである．

自然言語で与えられる質問を式 3.1.2.2のような述語式に変換する方法は自然言語処理の問題である．また，式 3.1.2.2のような形に変換できないような種類の質問としてどのようなものがあるかはまだ明らかにしていないが，そのような質問にも回答する必要があるならば，その方法を別途考えなければならない．

◆提示は，要求モデルに書かれている内容をわかりやすい形（図，表，自然言語，等）で人間に見せる機能である．

例えば，定義されている状態遷移をグラフの形で表したり，テキストとして定義されているメッセージデータのフォーマットを図の形で表す，等の視覚化を行う．

提示する内容によってそのわかりやすい形というのは様々であるので，個々の内容について表現の方法を考える必要がある．

◆検証は，要求モデルの正しさを確かめる（形式的にできることが望ましい）機能である．

『正しさ』といっても何が満足されれば『正しい』のかは自明ではないため，どのような検証を行うのかをまず明らかにしなければならない．考えられるものとして，状態の到達可能性，起こり得るデッドロックの検出，世界モデルとの整合性，要求モデルの記述内容の制約等がある．ここでは記述内容の制約の検証について検討する．

◆入力チェックは，要求モデル自体の記述の誤りをなくすために，記述項目の内容に関する制約をチェックする機能である．要求モデルの記述（式 3.1.2.1）の中には，①要求内容の記述，②項の型の宣言，③要求内容の記述の項の値に関する制約の３種類が含まれるものとする．制約のチェックとは式 3.1.2.1から生じる矛盾の原因を①の要求内容の記述の中で同定することである．

◆シミュレーションは，定義されている要求を実際に実行させてみる機能である．ただし要求の内容が全て実現されるわけではなく，ある動作を行う代わりにその動作内容を表示するだけだったり，外界から与えられる条件に代わるものを内部発生させたり人間が入力したりもする．また，シミュレーションの結果の情報は人間にわかりやすい形式で出力する．例えば，事象を人間が入力すると定義された状態遷移の中から正しいものを選択して遷移時の動作の一部を行う，ということである．

◆設計は，記述された完全な要求モデルから，実現が保証される設計モデルへの変換を行う機能である．この機能にはインタフェースの状態遷移から両側の通信主体の状態遷移を導く，論理的なモデルから具体的な実現方法を導く，必要な部分を逐次型の計算モデルへ変換する，必須な制御データの構造を導く，必要な計算コストからリアルタイム性の見積りをする，等がある．

３．１．３　ＡＲＴ言語の概要

ＡＲＴ（Automated Reasoning Tool）は，Inference Corporation（Los Angels, California）が提供する知識処理用言語およびその処理系である．ＡＲＴは知識表現用言語，推論エンジン，プログラミング環境からなる．ＡＲＴの基本的な３要素はファクト，ルール，ビューポイントである〔４〕，〔５〕．

① ファクト（Ｆａｃｔ）

ファクトは事実の宣言で，（項$_1$　項$_2$　…… 項$_n$）の形をしている．ファクトはファクトデータベースの中に存在していて，挿入／削除を行うことが可能である．このファクトを用いてスキーマ（一般にフレームと呼ばれているものに相当）を表現することも可能である．スキーマは一般に

（Ｓｃｈｅｍａ名$_1$
　（Ｓｌｏｔ名$_1$　Ｓｌｏｔ値$_1$）
　（Ｓｌｏｔ名$_2$　Ｓｌｏｔ値$_2$）　　… (3.1.3.1)
　　……
　（Ｓｌｏｔ名$_m$　Ｓｌｏｔ値$_m$））

の形で定義できるが，これは，

(Ｓｌｏｔ名$_1$　Ｓｃｈｅｍａ名$_1$　Ｓｌｏｔ値$_1$)
(Ｓｌｏｔ名$_2$　Ｓｃｈｅｍａ名$_1$　Ｓｌｏｔ値$_2$)　… (3.1.3.2)
⋮
(Ｓｌｏｔ名$_m$　Ｓｃｈｅｍａ名$_1$　Ｓｌｏｔ値$_m$)

と等価である．定義されたスキーマは式 3.1.3.2のような形でファクトデータベースの中に存在する．

スキーマがスロット値として他のスキーマを持つことによりスキーマ間の関係を表せる．関係の特殊なもののクラスとして，継承関係がある．個々の具体的なスロットも"Slot"という名前のスキーマから継承を受けているスキーマの１つである点が，ＡＲＴのスキーマ実現方法の特徴である．

② ルール（Ｒｕｌｅ）

ルールは，（ファクトのパターン　=>　操作）の形をした推論規則である．ここでファクトのパターンとは変数項を含むファクト表現の集合で，変数は１つのルールの中だけ（より厳密にいうと１回のパターンマッチングの中だけ）をスコープとするローカルな束縛をされる．また，操作とは，ファクトの挿入／削除／変更，ビューポイントの生成／削除，Ｌｉｓｐ関数の呼び出し，等である．例えば，あるＲｕｌｅは，

```
(DEFRULE Create-New-Relation "This is a very simple rule."
  (child-of ?a ?b)    (child-of ?b ?c)
  =>
  (assert (Grandchild-of ?a ?c)) )  ;  assertはFactの挿入
```

と書かれる．

③ ビューポイント（Ｖｉｅｗｐｏｉｎｔ）

ビューポイントは事実を内包する１つの世界である．ビューポイントによって，仮説，時刻，状態，等を表すことができる．ビューポイントのイメージを図に表すと図 3.1.3.1のようになる．ビューポイント間ではファクトを継承するが，あるビューポイントで新しいファクトを挿入したり，既にあるファクトを削除したりすることもできる（但し，単調推論を行う仮説型ビューポイントではファクトの削除はできない）．また図 3.1.3.2のように多重のビューポイントを定義することも可能である．

ＡＲＴの実行は基本的に，１回のパターンマッチに対して必ず（それ以上でもそれ以下でもなく）１回だけ発火するルールによって操作を行うという特徴がある．しかも，予め存在するファクトだけではなく動的に追加されるファクトに対してもパターンマッチングが行われ，また逆に動的に追加されるルールも以前から存在するファクトに対してパターンマッチングを行う．このような特徴から，要求モデルを変更してもルールが正しく適用

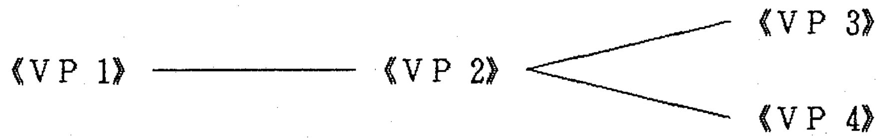

図 3.1.3.1　ビューポイント

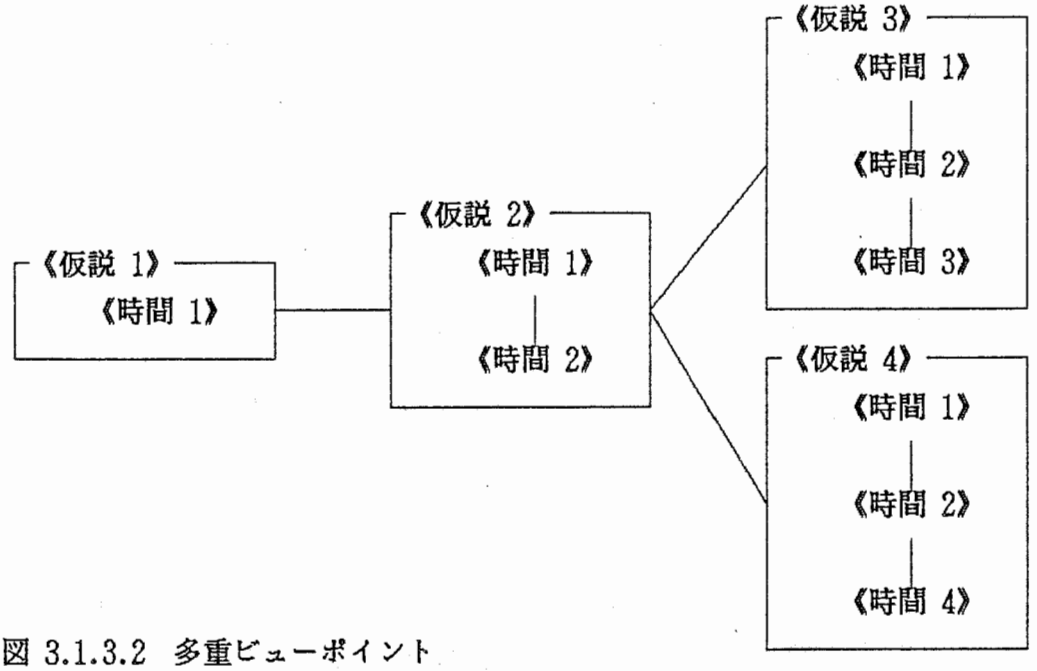

図 3.1.3.2　多重ビューポイント

されるという利点がある．その反面，ある局面ではルールを適用したくない場合もあり得るが，その場合は発火の可否を制御する機構の導入が必要になる．

3．2　実現方法の概要

3．1．2節で述べた各機能は，1つの形式的要求記述を操作対象とし，図 3.2.1のように解釈実行系と入出力系の2群として構成する．ＡＲＴ言語を用いる場合，形式的要求記述はファクトデータベースの中にファクトの集合として存在するので，これらの機能はルールとして記述し実現する．現在までに，スロット値の制約のチェック，シミュレーションの基本部分，簡単なユーザインタフェースを実現している．ユーザインタフェース以外の部分について次に説明する．

① スロット値に関する制約のチェック（入力チェック）

ここでは，スロットの値が，ｉ）ある特定のスキーマのインスタンスでなければならない（スキーマの継承型のチェック），ii）ある特定の型のＬｉｓｐオブジェクトでなければならない（Ｌｉｓｐオブジェクトの型チェック），iii）ある集合の要素でなければならない（列挙型のチェック），および，iv）スロットが持つ値の個数がある定められた範囲になければならない（スロット値の個数のチェック），という制約のチェックを行う．複数のスロットに及ぶ制約のチェックはまだ実現していない．詳細については，付録３のプログラムリストを参照．

ｉ）スキーマの継承型のチェック

図 3.2.2において，Ａのインスタンス（ここではＢ）のスロットＳ１の値が，Ｔ１のインスタンスでなければならないという制約を，

(C-Slot-Value-Type　A　S1　T1)

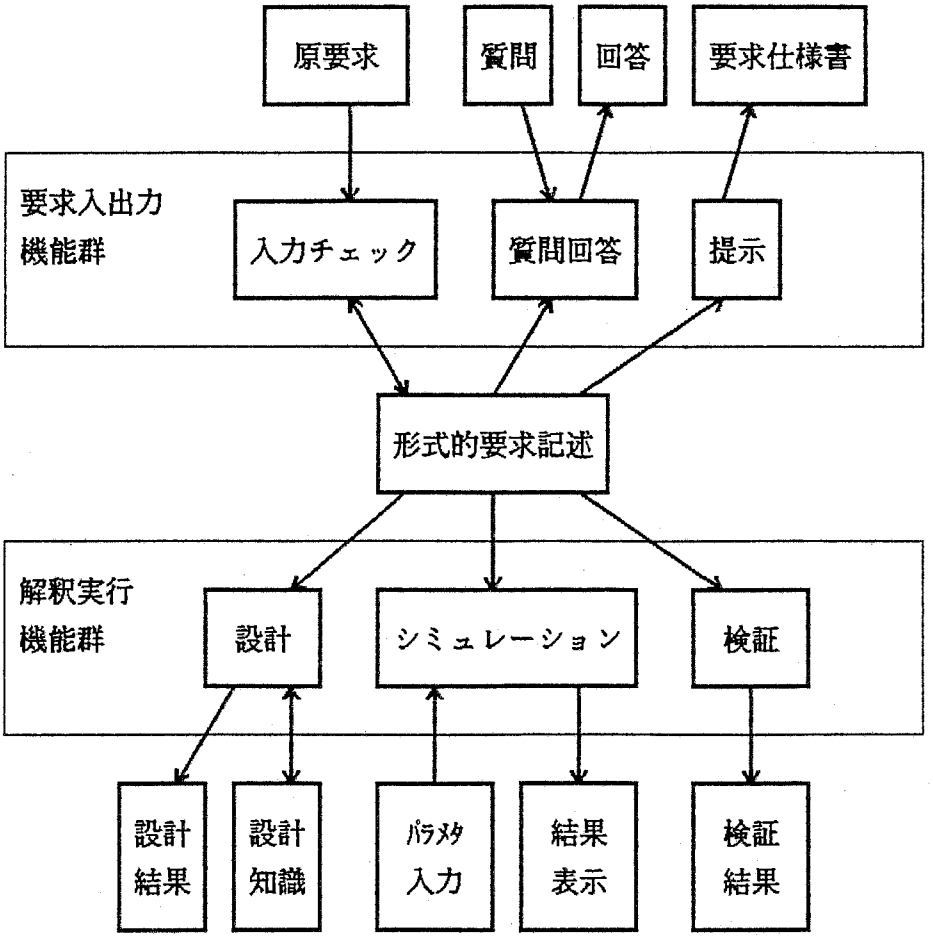

図 3.2.1　構成イメージ

というファクトで表現する．このとき，Ｂが持つＳ１の値Ｖ１について，

(is-a　V1　T1)

かどうかを判定する．（ルール名：　Check-Slot-Value-Type)

ii）Ｌｉｓｐオブジェクトの型チェック

図 3.2.3において，ＡのインスタンスのスロットＳ１の値が，（例えば）偶数でなければならないという制約を，

(C-Slot-Value-Predicate　A　S1　evenp)

```
(Schema A (S1))
(Schema B (is-a A) (S1 V1))
(Schema T1)
(Schema V1)
```

図 3.2.2　スキーマの継承型チェック

```
(Schema A (S1))
(Schema B (is-a A) (S1 100))
```

図 3.2.3　LISPｵﾌﾞｼﾞｪｸﾄの型チェック

```
(Schema A (S1))
(Schema B (is-a A) (S1 V1))
```

図 3.2.4　列挙型のチェック

```
(Schema A (S1))
(Schema B (is-a A) (S1 V1 V2 … V3))
```

図 3.2.5　値の個数のチェック

というファクトで表現する．このとき，Bが持つＳ１の値１００について，

(evenp 100)

を評価して判定する．ｅｖｅｎｐのかわりに適当なＬｉｓｐ関数を書くことによって，様々な型のチェックが可能である．（ルール名： Check-Slot-Value-Type)

iii）列挙型のチェック

図 3.2.4において，AのインスタンスのスロットＳ１の値が，（例えば）集合｛Ｖ１，Ｖ２，Ｖ３｝の要素でなければならないという制約を，

(C-Slot-Value-Member A S1 V1)
(C-Slot-Value-Member A S1 V2)
(C-Slot-Value-Member A S1 V3)

というファクトで表現し，具体的なインスタンスのスロット値についてその正当性を判定する．（ルール名： Check-Slot-Value-Member)

iv）スロット値の個数のチェック

図 3.2.5において，AのインスタンススキーマのスロットＳ１が持てる値の個数の範囲の制約を，

(C-Slot-How-Many A S1 n_1 n_2)

というファクトで表現する．ただし， は個数の最小値（０以上の整数），n_2は個数の最大値（n_1以上の整数，または，ＩＮＦ）である．このとき，

(S1 B ?)

というファクトが何個あるかを数え上げて範囲内にあるかどうかを判定する．

（ルール名： Create-Slot-Value-Counter, Count-up-Slot-Value, Check-Slot-How-Many)

個数のチェックは複数のルールが副作用を残しながら互いに影響しあうので，１度チェックを行った後要求モデルを変更すると，２度目のチェックは正しく働かない．この場合２度目のチェックも正しく行うには，関連するルールを削除してから新たに同じルールを定義しなおすとよい．

② シミュレーション

状態遷移の基本的な動きのシミュレーションを行う．１つのビューポイントがある時点での状態を表し，１回の状態遷移毎に新しいビューポイントを下に生成する．１つのビューポイントの下に複数の状態遷移を生成することも可能である．

あるビューポイントにおいて，

(Sim-Present-Status ?Status) … (3.2.1)
(Sim-Prev-Status ?) … (3.2.2)
(Sim-Prev-Transaction ?) … (3.2.3)
(Sim-Prev-Event ?) … (3.2.4)

とマッチするファクト（‘?’で始まる項は変数）があるとき，そのビューポイントに，

(Sim-Event ?Event) … (3.2.5)

とマッチするファクトを挿入すると，新しいビューポイントを生成して，ファクト

(Sim-Present-Status ?Next-Status) … (3.2.6)
(Sim-Prev-Status ?Status) … (3.2.7)

(Sim-Prev-Transaction ?Trans) … (3.2.8)
(Sim-Prev-Event ?Event) … (3.2.9)

を新しいビューポイントの中に作る．?Next-Statusと?Transは，具体的な状態遷移の定義から導かれる．また，式 (3.2.5)は元のビューポイントにおいて，式 (3.2.1)，(3.2.2)，(3.2.3)，(3.2.4)は新しいビューポイントにおいて，それぞれ削除される．

3．3 部品合成手法

ＫＡＮＴでは，中間および最終生産物を部品から合成する手法を用いようとしている．部品合成の際，要求仕様と部品仕様の記述の枠組みが必要である．また複数の部品を合成したときの生成物（合成された部品）と元の部品との関係（生成規則）もその枠組みの上で定められる．

まず部品仕様と要求仕様の記述のしかたを定義する．機能の要素を変換処理と時間的要因とに分類し，それぞれを静的機能，動的機能と呼ぶことにする．静的な機能は副作用がなく，入力データから出力データへの写像関係だけが常に問題となるような機能である．静的な機能の部品仕様は関数・論理式等で入出力データ間の関係を定義すればよい．また動的な機能は，副作用を伴う動作の時間的順序としての入力系列および出力系列が問題になる機能である．動的な機能の仕様は半順序関係を持つ動作の集合を定義すればよい．より一般的には要求および部品は静的および動的両方の機能を持つので，仕様は上の２つの組として定義される．

与えられた要求仕様と（複数の）部品仕様から要求を満足する生産物をつくるということは，要求を満足する部品の組合わせ方を，あらゆる部品組合せの空間の中で探索することである．部品を組み合わせるとき，入出力データのデータ型やデータの意味，動作の半順序列に対してマッチングを行う．但しここではデータの意味をどのように与えるかは問わない．

ある程度の規模の要求と充分な汎用性を持つほどの大きさの部品とを考えた場合，探索空間が非常に広すぎることが予想されるので，部品の組み合わせ方に関する知識を利用する方法が必要である．

部品探索の代案として，完全に機械的な探索によって行うのではなく，人間が自分の知識を用いて部品を組み合わせて生産物を作り，それが要求に合っているかどうかを検証するという方法も考えられる．検証項目は，入出力データの型の整合性，要求仕様を満足する入出力データ関係になっているかどうか，要求仕様を満足する動作の順序が得られているかどうか，冗長な部品がないかどうか，等である．

4．多重通信におけるプロセス間相互作用

通信ソフトウェアの中で有限状態機械モデルで表し切れないものの１つとして複数の通信路を制御する多重処理がある〔６〕．従来の方法では，複数の通信路を制御する通信システムの仕様は，まず有限状態機械モデルに基づき，各通信路毎の呼状態の管理部分と資

源状態管理等の共通な制御を行う部分に階層化し，次に階層間のインタラクションを付加するという形態で記述されてきた．しかし，この記述法は，通信路単独の呼処理の簡明さに対して，排他制御等の複数の通信路にまたがる多重処理が不鮮明であるという問題がある．この解決には，階層間のインタラクションを重視し，それを明確に表現できるモデルが必要となる．

多重処理を形式的に記述するモデルを検討するために，まず実現方法をも含めた詳細な仕様の記述を行ってみる．そこから多重通信に特有の考慮要因をみつけ，その問題を抽象化してモデルを構築するという方向で進める．多重処理の例として，X．２５におけるＤＴＥの論理チャネルの獲得・解放制御のシーケンス（図 4.1）を考えることにする．

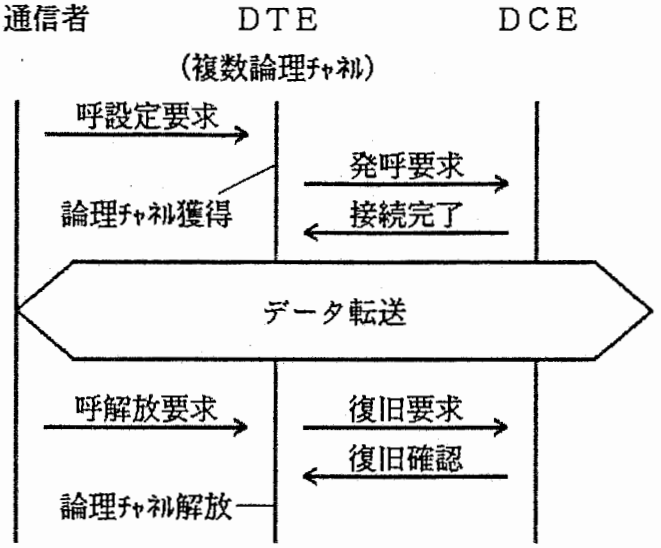

図 4.1
呼設定～呼解放
シーケンス

４．１　多重処理の記述例

プロセスの構成としては，複数の“論理チャネル制御”プロセスが各論理チャネル上で独立に状態遷移定義に基づく１対１プロトコル処理を行い，“管理部”プロセスが呼と論理チャネルの対応を管理する形になる．管理部は通信者からの呼設定要求に対して，空き状態の論理チャネルがあればその最老番号のものを獲得する．尚，各プロセスは並列に動作するものとする．

これは多重処理のうちの資源管理の問題であり，資源（論理チャネル）の空塞情報の管理と，資源の獲得・解放メッセージの処理により種々の実現方法がある．その典型的な２方式について，資源の獲得・解放に直接関係する部分はそれぞれ以下のような制御を行う．

〔方式１〕管理部が空塞情報を持ち，呼設定要求時にはその情報に基づき空き論理チャネルを選択し，該当する論理チャネル制御に呼設定指示メッセージを送る．論理チャネル制御は自らが空きとなったときに呼解放通知メッセージを管理部に送り，それによって管理部は空塞情報を空き状態に戻す．

〔方式２〕管理部は，呼設定要求時に，最老番号から順に獲得・呼設定指示メッセージを送る動作を，設定可応答が返るまで繰り返す．論理チャネル制御は各自の空塞情報に基づき呼設定可否応答メッセージを返す．

両方式を比較すると表 4.1.1のように双方に一長一短があり，一概に優劣を決め得るものではない．

表 4.1.1　2方式の利点と欠点

	利　　点	欠　　点
方式1	・管理部が外に問い合わせる必要がない．	・管理情報が冗長で実状態との不一致が起こり得る．
方式2	・実状態と管理情報との不一致がない．	・問い合わせのための論理チャネルオーバヘッドがある．

4．2　記述における考慮要因

上述の何れの方式においても，空き論理チャネルがある限り，呼設定要求を認める対処が必要であり，基本的にはメッセージの処理順序が重要となる．即ち，呼設定要求と呼の解放メッセージが同時に存在する場合には，後者を優先して処理する必要がある．

これを実現するためには，まず方式1では，管理部への呼設定要求メッセージと論理チャネル制御への復旧確認パケットのうち後者の処理を優先するという複数オブジェクト間にまたがる大域的同期制御が必要である．更に，管理部単独でも呼設定要求と呼解放通知のメッセージ待ち行列のうち後者の優先処理が必要となる．また方式2では，論理チャネル制御において，獲得・呼設定指示メッセージよりも復旧確認パケットを優先する必要があり，資源管理に係る順序的制約により，本来のプロトコル上の状態管理処理が影響を受ける．

即ち，多重処理を記述するための考慮要因としては，資源管理情報の他に，上述のような複数オブジェクトにまたがるメッセージ処理の順序関係，それらの大域的同期，プロトコル処理の記述への影響などがある．従って，記述の基本となるモデルの構成要素は，これらを統合し抽象化して示し得るものを選定する必要がある．また，この他にもリアルタイム性やサービスの公平性等も重要な考慮要因であると考えられる．

4．3　ＣＳＰによる記述

前節で説明した2方式の仕様をＣＳＰ（Communicating Sequential Processes）〔7〕を用いて厳密に記述する．

まずＣＳＰの記述法について，ＢＮＦ記法用いながら簡単に説明する．ＣＳＰの特徴は，並列する複数のプロセス間での同期や変数値のやりとりを簡単に記述できるコマンドが用

意されていることである．各プロセス内部の実行は逐次的である．各プロセスの実行内容はコマンド(command) によって表され，コマンドには単純コマンド(simple command)，選択コマンド(alternative command)，繰り返しコマンド (repetitive command)，並列コマンド (parallel command) がある．

<command> ::= <simple command> | <structured command>
<simple command> ::= <null command> | <assignment command>
| <input command> | <output command>
<structured command> ::= <alternative command> | <repetitive command>
| <parallel command>
<null command> ::= skip
<command list> ::= {<declaration>; | <command>;} <command>

並列コマンドは並行する複数のプロセスを表現する．

<parallel command> ::= [<process> {|| <process> }]
<process> ::= <process label> <command list>
<process label> ::= <empty> | <identifier>::
| <identifier>(<label subscript> {,<label subscript>})::

プロセス間の同期および値のやりとりは入出力コマンドによって行う．

<input command> ::= <source>?<target variable>
<output command> ::= <destination>!<expression>
<source> ::= <process name>
<destination> ::= <process name>
<process name> ::= <identifier> | <identifier>(<subscripts>)
<subscripts> ::= <integer expression> {, <integer expression>}

例えば，ｐ１？ｖ１という入力コマンドはｐ１という名前で識別されるプロセスから受け取った値を変数ｖ１に代入することを表し，ｐ２！ｖ２という出力コマンドは，変数ｖ２のあたいをプロセスｐ２に渡すことを意味する．これらの入出力コマンドはすべて同期的に動作する．即ち，相手プロセスが対応する入出力コマンドに達するまで，自プロセスの入出力コマンドのところで待ち合わせをする．

ガード付きコマンド(guarded command) はガード部の条件が満たされた場合のみ後続するコマンドリストを実行する．選択コマンドは複数のガード部を持ち，条件が満たされるガードのうちの１個を選んで（複数のガード部の条件が合うときどのガードが選ばれるかは非決定的である），後続のコマンドリストを実行する．繰り返しコマンドは，中のガード条件が全く満たされなくなるまで実行を繰り返す．

<repetitive command> ::= *<alternative command>
<alternative command> ::= [<guarded command> {[]<guarded command> }]
<guarded command> ::= <guard> → <command list> |
(<range> {,<range>})<guard> → <command list>
<guard> ::= <guard list> | <guard list>;<input command> | <input command>
<guard list> ::= <guard element> {;<guard element>}
<guard element> ::= <boolean expression> | <declaration>

先の２方式をこのＣＳＰを用いて記述すると次のようになる．

```
SYSTEM = [ sel::SEL ‖ lc(i:1..n)::LC ]

/*  方式１　選択部空塞管理方式 */
  SEL =  BI(1..n) := busy;
         * [(i:1..n)lc(i)?BI → BI(i) := idle []
            mng?cr()  → [ i:=1; * [ i≦n, BI(i) = busy → i:=i+1;  ] ;
                                [ i≦n → BI(i) := busy; lc(i)!cs; SETUPOK []
                                  i>n → SETUPNG  ] ] ]
  LC =   bi := idle; sel!idle;
         * [ sel?cs → bi := busy; SETUPACTION;
                      * [bi = busy → mng?pkt;
                                     [ pkt≠CF  → PROTOCOLACTION []
                                       pkt=CF → bi := idle ] ] ;
                      sel!idle ]
/*  方式２　空塞問い合わせ方式 */
  SEL =  BI := busy;
         * [ mng?cr → i:=1;
                      * [ i≦n, BI = busy  → lc(i)!cs; lc(i)?BI; i:=i+1 ] ;
                      [ i≦n → BI := busy; SETUPOK  []
                        i>n → SETUPNG ] ]
  LC =   bi := idle;
         * [ sel?cs → sel!idle; bi := busy; SETUPACTION;
                      * [ bi = busy  →
                             [ mng?pkt  → [ pkt≠CF  → PROTOCOLACTION []
                                            pkt=CF → bi := idle ] []
                               sel?cs → sel!busy ] ] ]
```

これら２方式の違いは，ｌｃ（ｉ）（１≦ｉ≦ｎ）の各プロセスが管理する論理チャネルがｂｕｓｙ状態からｉｄｌｅ状態に遷移したという事象を，どの時点でｓｅｌプロセスに知らせるかという点の違いである．方式１では呼が解放された時点でＬＣの定義の最下行のｓｅｌ！ｉｄｌｅによってｓｅｌプロセスに知らせているのに対し，方式２では呼が解放されてその次にｓｅｌプロセスがｌｃ（ｉ）！ｃｓによって問い合わせてきた時点で２行目のｓｅｌ！ｉｄｌｅで知らせている．（呼がまだ解放されていない場合は最下行のｓｅｌ！ｂｕｓｙが実行される）．

ここで新しい入力コマンドを導入してＣＳＰを拡張し，このタイミングの違いを抽象化して２方式を統一した仕様を記述する．新しいコマンドは‘#’で表わし，‘?’と同様に前に相手プロセス名，後ろに入力変数を記述する．‘?’は対応する出力コマンドの実行を必ず待ちあわせるのに対して，‘#’は対応する出力コマンドがない場合はその入力をスキップする．つまり‘#’が実際に値を入力する条件は，対応するプロセスがそれ以

前に必ず出力コマンドに達していることである．またもう１つの拡張として，ガード部にｅｌｓｅを記述できるようにする．選択コマンド（alternative command）において他のガードが解かれないときに，ｅｌｓｅガードが解かれて後続のコマンドが実行される．拡張したＣＳＰで記述した２方式の統一仕様は次のようになる．

```
/*  統一仕様  */
SEL =  * [ mng?cr → [ (i:1..n)lcn(i)#ask → lcn(i)!cs; SETUPOK  |
                      else → SETUPNG ] ]
LC  =  sel!idle;
       * [ sel?cs → * [ mng?pkt, pkt ≠CF  → PROTOCOLACTION ] ;
           sel!idle ]
```

４．４　考察

前節で一見違うように見える２方式の仕様が何故１つに統一できたのか，また２つの仕様が共通して持つ本質的に必要なプロセス間の相互作用の性質について考察する．

２つの方式は，ｌｃ（ｉ）プロセスにおける事象をいつｓｅｌプロセスが受け取るかという実現上の違いによるものだった．ｓｅｌプロセスは必ずしも論理チャネルの解放を即時に知る必要はなく，次に通信主体から接続要求が来てｉｄｌｅ状態にある論理チャネルを探すときまでに知ればよい．統一仕様はこの実現方法の違いに独立で，ｌｃ（ｉ）プロセスが事象をｓｅｌプロセスに知らせようとしてから，ｓｅｌプロセスが実際にその事象を必要とするまでのタイミングのずれを吸収して抽象化したものである．別の言い方をすると，事象はｓｅｌとｌｃ（ｉ）の両プロセス間の入出力チャネル内に必要とされるまで蓄積されていることになる．

次に各プロセスの状態変数間の制約に着目して，この概念の一般化を考察する．元の問題は，ｍ個の通信主体からの通信要求をｎ個の論理チャネルに割り当てるというものである．従って全体として必要な状態変数は，各論理チャネルが，ｂｕｓｙかｉｄｌｅかと，ｂｕｓｙならばその論理チャネルが割り当てられている通信主体の番号である．各プロセスが必要な状態変数を持つとして，データ構造をＰａｓｃａｌ風に表現すると下のようになる．

```
const  n, m;
type   LCnumber = 1..n ;  Tnumber = 1..m ;
       LCstate = record
         BI : (busy, idle) ; Talloc : Tnumber   end;

                プロセスｓｅｌの状態データ
                var    LCstateVector : array [ LCnumber ] of LCstate ;
制約
                プロセスｌｃ（ｉ）の状態データ
                var    BI : ( busy, idle ) ;  Talloc : Tnumber ;
                       ProtocolStatus : (Ready, DTEwaiting, DCEwaiting, …… ) ;
```

ｓｅｌプロセスとｌｃ（ｉ）プロセスとの間で変数の値の制約を持っている．即ちｓｅｌのBI(i) とｌｃ（ｉ）のBIは本来の問題としては同じ値を持っていなければならないという，値の一致の制約である．

ここで簡単化のためにｐ１，ｐ２という２つのプロセスがそれぞれｖ１，ｖ２という変数を持ち，両者間に一致の制約があると考える．それぞれのプロセスは自分の変数の値を読んだり書き換えたりすることができるが，相手プロセスの変数を直接読んだり書き換えたりすることはできない．２プロセス間には通信手段があり，それを使って同期を取ったり値を相互に伝えあったりすることができるものとする（図 4.4.1）．

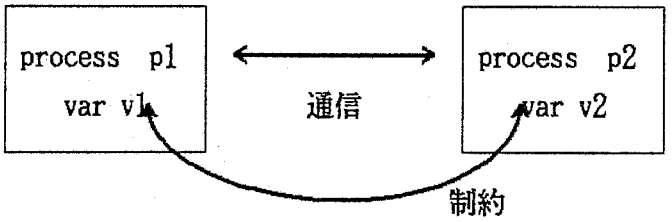

図 4.4.1 プロセス間の変数の制約

両プロセスに重複して状態データを持たずに片方だけで状態を管理し，その状態へのアクセス（参照および定義）はプロセスへのメッセージとして記述するという考え方もできが，そのようなメッセージによって情報を書き換える場合のメッセージパラメタとしての新変数値，または情報を読み出す場合のメッセージの返値の利用は，メッセージを送る側のプロセスにもそのような変数を持っていると考えるのと何ら変わりない．従って，両方に状態データを持ち相互作用によって状態を参照・定義しあうというより一般的なモデルで考える．

変数間の制約を保つために，一方の変数が再定義（更新）されたときにその変更を他方の変数に伝えるプロセス間相互作用が必要となる．しかし，一方の変更が必ずしも即座に相手に伝わる必要はなく，相手プロセスがその変数を参照する直前までに伝播していさえすればよい．

変数間の制約は明示的に定義することにし，伝播のための相互作用を次のように定義する．ｐ１またはｐ２どちらかのプロセスがコマンドを起動したときに，最後に更新された側の変数の値が，まだ更新されてない側の変数（古い値を持っている）に代入される．前の通信以来どちらの変数も値が変わっていない場合は，実質的には何も起こらない．この相互作用を使うためのコマンドを記号‘@’で表すことにする．記号‘@’の前に相手プロセス名，後ろに自プロセス内の変数名を記述し（例．ｐ１@ｖ２），それ全体を１つの変数のように扱い代入や参照を行う（例．ｐ１@ｖ２：＝ｂｕｓｙ，ｘ：＝ｐ２@ｖ１）．

‘@’は変数の定義（Definition）として使われる場合と変数の参照（Reference)として使われる場合とがある．定義とは，その変数が代入式の左辺に現れることである．参照とは，その変数が代入式の右辺やガード部のブール表現中等に現れることである．制約の維持のためには変数の定義・参照およびプロセス間の相互作用が正しい順序で行われる必要がある．個々のプロセスは独立に動作するため，この正しい順序は図 4.4.2のような半順序として与えられる．ここで，Ｒ（ｖ）は変数ｖの値が参照されること，Ｄ（ｖ）は変

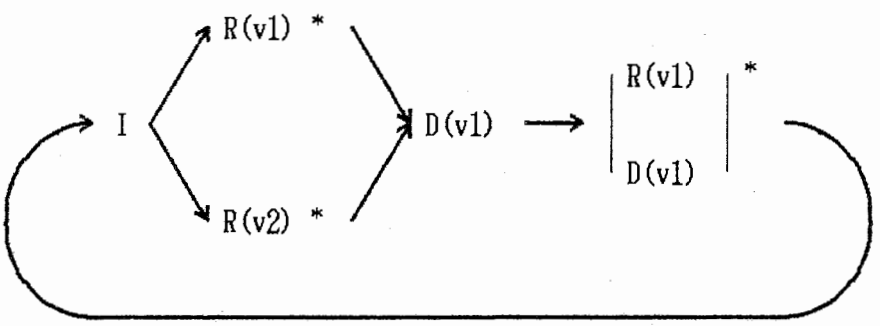

図 4.4.2　状態変数の正しさを保証するアクセス順序

数ｖの値が定義されること，Ｉは上述の相互作用を表す．また右肩のアスタリスクは０回以上の繰り返しを示す．この図はｖ１だけが定義される場合を表しているが，ｖ２が定義される場合も，変数名が互いに入れ代わるだけで同様である．この半順序関係をＣＳＰ風の記法を用いて書くと次のようになる．

```
*  [ I;  [ *[R(v1) ] ‖ *[R(v2) ] ] ;
     [ D(v1);  *[ R(v1) [] D(v1) ] []
       D(v2);  *[ R(v2) [] D(v2) ]     ] ]
```

一般に，｛D( )，R( )，I｝の順序列が与えられたとき，その列が上述の半順序関係を満足するかどうかを判定することができる．また，満足しない順序列に適切なＩを挿入することにより満足な列を作ることも可能である．

上では‘@’コマンドを実現方法を意識しない抽象的な通信として説明した．次にその実現上の問題を考える．‘@’コマンドを実現するためには，どちらのプロセスが最後に値を書き換えたかを知っている必要がある．１つの簡単な方法は変数アクセスの順序をグローバルに制御し，変数の参照時に逐一インタラクションが必要かどうかを判定することである．

図 4.4.3において２つのプロセスの間にある斜線部は，①最後の通信以来値が書き換えられたかどうか，②最後に書き換えたプロセスのＩＤ，の２項目の情報と，その情報にアクセスするためのセマフォを持っている．これらの情報はプロセスに共通で，‘@’コマンドを通してだけアクセス可能であるとする．

‘@’コマンドによってｖ１，ｖ２が定義されるときには，まずセマフォによってそのグローバル情報をロックし，実際の変数値の更新を行い，項目①にtrueを，項目②に今書き換えたプロセスＩＤを記録し，ロックを解くという手順をとる．また，ｖ１，ｖ２が参照されるときには，まずセマフォによってロックし，項目①がtrueで項目②に自プロセス

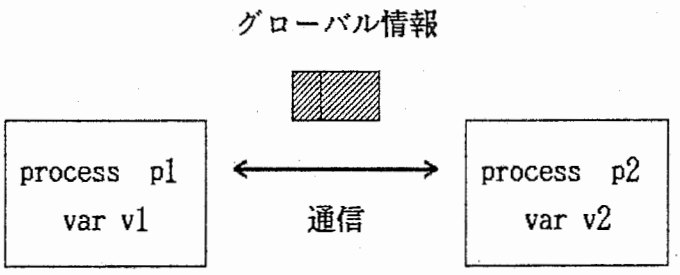

図 4.4.3　実現方法の一例

ＩＤが記録されているならば，インタラクションを起動して（このとき相手プロセスは必ずインタラクションに応じるものとする），相手側の変数から自分側へ値をコピーし，項目①にfalseを記録して，ロックを解くという手順になる．但し，これはある１つの実現方法だけを述べたもので，‘@’コマンドの計算が本質的にこのようなグローバル情報を持つべきというわけではない．

３プロセス以上にまたがる制約の解消のためのインタラクション，値の一致以外の制約の解消も基本的に上の方法の応用として解決できる．

上述の合成・検証を論理を用いて扱う場合には，時間的な順序関係を表すために様相論理〔８〕，〔９〕のうちの時制論理（tense logic）を用いることができる．

５．まとめおよび今後の課題

通信ソフトウェアの要求仕様記述法について検討し，特に多重処理におけるプロセス間の相互作用とプロセス内の状態変数の関係を詳しく考察した．具体的には，先ず，要求文書を分析しその中の形式性を整理した．これは主に有限状態機械モデルに基づいて，記述内容を対象とその性質および対象間の関係という形で取り出したものであり，この特徴を利用して，フレーム形式の記述法を用いることを試みた．またこのとき，スロット値に型の制約があることがわかった．上記の要求文書の分析の結果，有限状態機械モデルだけではうまく表しにくい内容の要求もあることがわかり，他のモデルも必要であることを指摘した．そのような内容の１つとして多重処理における資源管理の問題をとりあげ，仕様をＣＳＰを用いて記述し，次にＣＳＰを拡張することによってその仕様を抽象化する例を示した．更にこの問題を一般化し，複数プロセスにおける状態変数の定義・参照とプロセス間で変数の再定義を伝達するための相互作用との順序関係を考察し，正しい半順序列を与えた．これによって，このような順序を機械的に検証または訂正できる可能性が得られた．尚，フレーム形式仕様の処理系のプロトタイプを作成したが，それは上述のスロット値のチェックと，有限状態機械モデルに基づいたプロトコル処理の簡単なシミュレーションの機能を持つ．

異機種間のコンピュータ接続のために標準の通信規約が定められつつあるが，その全部が形式的に記述されていないために，同じ標準に則っていながらその解釈の違いが原因となって相互に接続できないといった問題が既に起こってきている．このような解釈の違いは，従来モデルで表現しにくく適切な形式的記述法がない部分で特に生じると考えられ，そのような部分の適切なモデルを構築することが重要である．

本報告書で考察した多重処理における資源管理はそのような部分のごく一部で，他にも複数プロセスにまたがるメッセージ処理の優先順序，大域的同期，それらのプロトコル処理への影響，またリアルタイム性，サービスの公平性等の解決すべき問題が残っている．これらについても，今後具体的仕様を分析して考慮要因を明確化し，モデルを構築していく必要がある．

謝辞

本研究を進めるにあたって適切な御指導を頂いたＡＴＲ通信システム研究所山下紘一社長，門田充弘通信ソフトウェア研究室長に感謝します．また，討論・助言して頂いた同研究所各位に感謝します．

参考文献

1. 島他:“次世代を目指したソフトウェア自動作成システムＫＡＮＴの構想”，情報処理学会第35回全国大会，7M-3，(1987)．
2. 芝本，林他:“設計仕様から見た要求仕様記述法”，情報処理学会第36回全国大会，1L-5，(1988)．
3. ＣＣＩＴＴ: データ通信網　インタフェース　勧告X.20～X.32，RED BOOK，Volume Ⅷ，Fascicle Ⅷ.3，日本ＩＴＵ協会，(1986)．
4. Bruce D. Clayton: ＡＲＴ　Ｐｒｏｇｒａｍｍｉｎｇ　Ｔｕｔｏｒｉａｌ，Vol. 1-4，Inference Corp.，(1987)．
5. ＡＲＴ　Ｒｅｆｅｒｅｎｃｅ　Ｍａｎｕａｌ，Vol. 1-2，Inference Corporation，(1987)．
6. 芝本，西園他:“通信ソフトウェアにおける多重処理仕様記述法の検討”，情報処理学会第37回全国大会，5L-6，(1988)．
7. C. A. R. Hoare: “Communicating Sequential Processes”，Communications of the ACM，Vol. 21，No. 8，pp. 666-677．
8. 堂下，西田，三浦:“様相論理とその情報処理への応用（Ⅰ）様相論理”，情報処理，Vol. 29，No. 1，pp. 2-10，(Jan. 1988)．
9. 堂下，西田，島田:“様相論理とその情報処理への応用（Ⅱ）ﾊｰﾄﾞｳｪｱ・ｿﾌﾄｳｪｱへの応用”，情報処理，Vol. 29，No. 2，pp. 129-135，(Feb. 1988)．

付録 1

記述レベル

◇対象記述	メタレベルでない，要求の対象に関する記述.
◇背景記述	
◇メタ記述	要求対象ではなく，記述そのものに関する記述.
◆文書目的	当該文書又はセクションの一般的説明・記述目的の記述.
◆記述場所	何が何処に（どの文書に，どのセクションに）書かれているかに関する記述．他文書・他ブロックの参照．図・表の参照.
◆定義内容	文書の定義内容に関する記述.
◆定義範囲	文書がどの範囲の事柄を定義しているか，又はどの範囲を定義していないかに関する記述.
◆記述法・定義法	文書がどのような記法に従って記述されているかに関する記述.
◆用語・略語の意味	文書中に現れる用語・略語の意味に関する記述.

定義対象

◇概念定義	要求の中で使われる概念の定義に関する記述.
◇機能定義	機能の定義
◆静的機能	
◆動的機能	
＋制御定義	
－経時的順序	時間的に前後関係があるものの順序.
－制御フロー	機能間の制御の流れに関する記述.
－タイミング	機能の実行のタイミングに関する記述．インタバル時間等.
－状態定義	有限状態機械モデルにおける状態とその遷移の定義.
－事象定義	
▷能動	状態を変化させる.
▷受動	状態の変化に応じた処理を行う.
＋動作定義	
◇データ定義	使われるデータの定義に関する記述.
◆値域	
▷連続領域データ	値域が連続的な領域となるようなデータ．数値・文字列等.
▷選択データ	値域が離散的な領域となるようなデータ．選択パラメタ・論理値・数ビットのビット列等
◆存在場所	
▷入出力データ	ある機能単位の入出力となるようなデータ.
▷外部データ	機能の外部でデータ自体が独立に存在するもの.
▷内部データ	機能の内部的なデータ.

対象が持つ属性・対象間の関係

◇属性定義

◇関係定義

◆表出－意図の対応関係　　表面に表れているもの・概念・名前とその裏にある意図との関係に関する記述.

＋名前－内容の対応
＋状態－意味の対応
＋データ－意味の対応
＋動作－目的の対応
＋データ－目的の対応
＋手段－目的の対応

◆階層間対応関係　　階層性のあるものにおける，階層間対応関係
＋機能の仕様と実現の関係
＋階層的データ

◆具体－抽象関係

◆独立関係
◆従属関係

属性値の性質

◇記号

◇真偽値

◇数値

◆次元
＋個数，＋時間，＋時刻，＋データ長

◆範囲
＋１点，＋複数点，＋連続領域

◆精度
＋正確，＋曖昧・近傍

◇構造

◆組

◆集合

◆シーケンス
＋全順序シーケンス，＋半順序シーケンス

◆場合分け　　条件による場合分け

◆論理的組合せ
＋Ａｎｄ，＋Ｏｒ，＋全称，＋存在

その他

◇多重性	複数の処理の同時実行と相互作用
◇公平性	複数の処理に公平に計算資源が与えられること
◇不確定性	ソフトウェアにとって不確実な事柄に関する記述．例えば物理的な制約によるタイミングのずれ等．
◇拡張性	機能やデータ等が将来拡張される含みを持っていること．

付録 2

Ｏｂｊｅｃｔ名	型
Ｓｌｏｔ名	Ｓｌｏｔ値の制約

もの (Object)	<Object>
名前 (Name) 型 (is-a) 別名 (Alias)	<Symbol> <Object> <Object>
関係 (Relation)	<Object>
対称関係 (Symmetric-Relation)	<Relation>
接続関係 (Connction)	<Symmetric-Relation>
接続先	<Object>
全体－部分関係 (Part-of)	<Relation>
全体	<Object>
クラス－サブクラス関係 (is-a)	<Relation>
上位 (Superclass)	<Object>
代表名	<Object>
要素 (Element)	set-of <Object>
パラメタ (Parameter)	<Object>
システムパラメタ (System-Parameter)	<Parameter>
タイマ値 (Timer-Value)	<System-Parameter>
データ長最大値 (Maximum-Data-Length)	<System-Parameter>
データ (Data)	<Object>
フォーマット (Data-Format)	

メッセージ (Message)		<Data>
	制御用メッセージ (For-Control) 情報用メッセージ (For-Information) 送信するものの型 (Sender-Type) 受信するものの型 (Receiver-Type) メッセージが転送される状態・状況 (Situation) メッセージパラメタ (Message-Parameters) メッセージデータ (Message-Data)	<Boolean>> <Boolean>> <Object> <Object> <Status> | <Comment> list-of <Parameter> <Data>
シーケンス番号 (Sequence-Number)		<Data>
	番号体系	
状態 (Status)		<Object>
	オーナー (Owner) タイムリミット (Time-Limit) 副状態 (Sub-Status)	<Object> <Timer-Value> set-of <Status>
状態遷移 (Transition)		<Object>
	前状態 (Previous-Status) 次状態 (Next-Status) 事象 (T-Event) 遷移動作 (T-Action)	<Status> <Status> <Event> <Action>
事象 (Event)		<Object>
メッセージ伝送 (Message-Transfer)		<Event>
	伝送メッセージ (Transferred-Message)	<Message>
動作 (Action)		<Object>
	動作状態・状況 (Situatuin) 動作主体 (Agent) 様式 (Pattern) 動作内容	<Status> <Object> {must,may} partial-ordered-set-of (<Action>,<Primitive-Action>)
手順 (Procedure)		<Object>
	手順内容	partial-ordered-set-of (<Procedure>, <Action>, <Transition>)
例外状態 (Exception-Status)		<Status>
	例外条件 (Condition) 通知処置 回復動作 (Recovery-Action)	 <Action> <Action>

衝突状態 (Collision-Status)		<Exception-Status>
	衝突メッセージ (Colliding-Messages)	pair-of <Message>
タイムアウト状態 (Timeout-Status)		<Exception-Status>
	前状態	<Status>

付録3　プログラムリスト

（プログラム名）	（内容）
RM-Protocol-Kernel.art	状態・メッセージ・状態遷移等のプロトコル処理の基本部分のスキーマ定義と，入力チェック・シミュレーション用ルールの定義
X25-Packet-Level.art	X．２５固有部分の具体的スキーマ（インスタンス）の定義
RM-Own-Predicates.lisp	補助的なＬＩＳＰのPredicateの定義
RM-Own-Utilities.lisp	動的なルール生成のための関数の定義

```
;;; -*- Mode: ART; Base: 10.; Package: ART-USER -*-

;|
;|
;|   Kernel for Protocol Requirement Model  written in ART (Version 3)
;|
;|      Written by   Naoki SHIBAMOTO
;|             from 04-APR-1988
;|
;|
;|   Table of Contents
;|
;|     + Definition of Viewpoint Levels for Protocol Simulation
;|
;|     + Definition of Root Object (named "Object")
;|
;|     + Definition of Inheritance Relation (named "is-a+")
;|
;|     + Definition of Object Classes
;|
;|     + Definition of Relation (named "Connection")
;|
;|     + Definition of Relations for Slot Value Constraint
;|
;|     + Definition of Rules for Slot Value Constraint
;|
;|     + Definition of Constraints on Slot Values
;|
;|     + Definition of Relations for Protocol Simulation
;|
;|     + Definition of Rules for Protocol Simulation
;|

;;;
;;; Definition of Viewpoint Levels for Protocol Simulation
;;;

(def-viewpoint-levels ; 4/12/88 10:28:21
  (Ch merging nil semantics state))


;;;
;;; Definition of Root Object
;;;

(defschema Object "もの" ; 4/05/88 19:31:05
  (Name)
  (Alias (slot-how-many multiple-values))
  (Intension)
  (in-Section (slot-how-many multiple-values))
  ; Sections which define this object
  )


;;;
;;; Definition of Inheritance Relation
;;;

(defschema is-a+ "Works as like 'is-a' relation" ; 4/07/88 17:08:18
  ; 'Is-a+' is for convenience in using Relational Network of ART Studio.
  ; But, the effect hasn't yet fully tested.
  (instance-of inh-Relation)
  (element-of inh-Relations)
  (inverse kinds+)
  (transitivity (repeat (step is-a+ $) 1 inf))
  )
```

```
;;;
;;; Definition of Object Classes
;;;

(defschema Message "メッセージ" ; 4/06/88 18:33:48
  (is-a+ Object)
  (for-Control) (for-Information)
  (Sender-Type) (Receiver-Type) (Situation)) ; Situation ?

(defschema System-Parameter "システムパラメタ" ; 4/07/88 16:22:25
  (is-a+ Object)
  (Parameterized-Slot (slot-how-many multiple-values)))

(defschema Timer-Value "タイマ値" ; 4/11/88 19:28:56
  (is-a+ System-Parameter)
  (Seconds)
  (Parameterized-Slot Seconds))

(defschema Data "データ" ; 4/06/88 18:41:59
  (is-a+ Object))

(defschema Status "状態" ; 4/06/88 18:42:24
  (is-a+ Object)
  (Owner) (Time-Limit)
  (Substatus (slot-how-many multiple-values)))

(defschema Status-Transition "状態遷移" ; 4/12/88 10:40:47
  (is-a+ Object)
  (Pre-Status (slot-how-many multiple-values)) (Next-Status)
  (T-Event (slot-how-many multiple-values)) (T-Action))

(defschema Event "事象" ; 4/06/88 18:47:42
  (is-a+ Object))

(defschema Message-Transfer "メッセージ転送" ; 4/06/88 18:49:00
  (is-a+ Event)
  (Transferred-Message))

(defschema Action "動作" ; 4/06/88 18:52:17
  (is-a+ Object)
  (Action-Pettern) (Agent) (Acts))

(defschema Procedure "手続き" ; 4/06/88 18:58:09
  (is-a+ Object)
  (Proc-Acts))

(defschema Exception-Event "例外事象" ; 4/07/88 15:39:06
  (is-a+ Event)
  (Recovery-Method))

(defschema Time-Out "タイムアウト" ; 4/07/88 15:40:31
  (is-a+ Exception-Event)
  (Status-B4-Timeout))

(defschema Collision "衝突" ; 4/07/88 15:41:45
  (is-a+ Exception-Event)
  (Colliding-Messages (slot-how-many multiple-values)))



;;;
;;; Definition of Relation "Connection"
;;;

(defrelation Connection (?primary ?secondary ?channel) ;4/07/88 17:33:41
  "接続関係")
```

```
;;;
;;; Definition of Relations for Slot Value Constraint
;;;

;-- Relation for Slot Value Violation Fact

(defrelation Slot-Value-Violation (?v-code ?schema ?slot ?value ?correct-type)
  ; 4/06/88 19:40:10
  "Constraint on slot value is not satisfied")


;-- Relations for Constraint on Slot Value Type

(defrelation C-Slot-Value-Type (?schema ?slot ?value-type) ; 4/06/88 19:18:44
  "Constraint on slot value type. (What type should the value be)")

(defrelation C-Slot-Value-Predicate (?schema ?slot ?predicate) ; 4/07/88 11:17:08
  "Constraint on slot value type. (Whether the value satisfies the LISP predicate)")

(defrelation C-Slot-Value-Member (?schema ?slot ?member) ; 4/06/88 19:22:12
  "Constraint on slot value. (What value can the value is)")


;-- Relation for Constraint on Slot Value Number

(defrelation C-Slot-How-Many (?schema ?slot ?lower ?upper) ; 4/11/88 11:41:36
  "Constraints on how many values the slot has")
(defrelation Temp-Slot-Value-Counter (?schema ?slot ?lower ?upper ?how-many)
  ; 4/11/88 11:44:56
  "Fact for value counter")
(defrelation Temp-Slot-Value-Exists (?schema ?slot ?Value)
  ; 4/14/88 11:28:15
  "Fact that the value exists")
```

```
;;;
;;; Definition of Rules for Slot Value Constraint
;;;

;-- Check by Object Class

(defrule Check-Slot-Value-Type "Checking the constraint on slot value type."
  ; 4/07/88 10:15:01
  (C-Slot-Value-Type ?class-schema ?slot ?value-type)
  (schema ?schema
          (is-a+ ?class-schema)
          (?slot ?slot-value))
  (not (is-a+ ?slot-value ?value-type))
  =>
  (assert (Slot-Value-Violation value-type ; Violation Code
                                ?schema ?slot ?slot-value ?value-type)))

;-- Check by LISP Predicate

(defrule Check-Slot-Value-Predicate "Checking the constraint on slot value type."
  ; 4/07/88 11:19:32
  (C-Slot-Value-Predicate ?class-schema ?slot ?pred)
  (schema ?schema
          (is-a+ ?class-schema)
          (?slot ?slot-value))
  (test (null (apply ?pred (list ?slot-value))))
  =>
  (assert (Slot-Value-Violation predicate ; Violation Code
                                ?schema ?slot ?slot-value ?pred)))

;-- Check by Member of Finite Value Set

(defrule Check-Slot-Value-Member "Checking the constraint on slot value type."
  ; 4/07/88 20:36:01
  (C-Slot-Value-Member ?class-schema ?slot ?)
  (schema ?schema
          (is-a+ ?class-schema)
          (?slot ?slot-value))
  (not (C-Slot-Value-Member ?class-schema ?slot ?slot-value))
  =>
  (assert (Slot-Value-Violation member ; Violation Code
                                ?schema ?slot ?slot-value Nil)))

;-- Check How Many Values the Slot Has

(defrule Create-Slot-Value-Counter "Create counter fact." ; 4/14/88 11:29:13
  (declare (salience (+ 2 *default-salience*)))
  (C-Slot-How-Many ?class-schema ?slot ?lower ?upper)
  (schema ?schema
          (is-a+ ?class-schema)
          (?slot ?v))
  =>
  (assert (Temp-Slot-Value-Exists ?schema ?slot ?v)
          (Temp-Slot-Value-Counter ?schema ?slot ?lower ?upper 0)))

(defrule Count-up-Slot-Value "Count up a counter fact by one." ; 4/14/88 11:31:22
  (declare (salience (1+ *default-salience*)))
  ?t1 <- (Temp-Slot-Value-Exists ?schema ?slot ?v)
  ?t2 <- (Temp-Slot-Value-Counter ?schema ?slot ?lower ?upper ?i)
  =>
  (retract ?t1 ?t2)
  (assert (Temp-Slot-Value-Counter ?schema ?slot ?lower ?upper =(1+ ?i))))

(defrule Check-Slot-How-Many "Check the counted value." ; 4/11/88 13:47:09
  (declare (salience *default-salience*))
  ?t <- (Temp-Slot-Value-Counter ?schema ?slot ?lower ?upper ?i)
  (=> (retract ?t))
  (split ((test (RM-out-bound-p ?lower ?upper ?i)) ; Violation ditected
          =>
          (assert (Slot-Value-Violation How-Many ; Violation Code
                                        ?schema ?slot ?i Nil)))
         ((test (RM-in-bound-p ?lower ?upper ?i)) => ) ))
```

```
;;;
;;; Definition of Constrants on Slot Values
;;;   (using relations defined previously)
;;;

(deffacts C-S-Value-Object "" ; 4/07/88 21:03:17
  (C-Slot-Value-Predicate Object in-Section stringp))

(deffacts C-S-Value-Message "" ; 4/07/88 21:09:38
  (C-Slot-Value-Member Message for-Control T)
  (C-Slot-Value-Member Message for-Control Nil)
  (C-Slot-Value-Member Message for-Information T)
  (C-Slot-Value-Member Message for-Information Nil))

(deffacts C-S-Value-Timer-Value "" ; 4/07/88 21:11:44
  (C-Slot-Value-Predicate Timer-Value Seconds numberp))

(deffacts C-S-Status "" ; 4/07/88 21:13:35
  (C-Slot-Value-Type Status Time-Limit Timer-Value))

(deffacts C-S-Status-Transition "" ; 4/07/88 21:19:06
  (C-Slot-Value-Type Status-Transition Pre-Status Status)
  (C-Slot-Value-Type Status-Transition Next-Status Status)
  (C-Slot-Value-Type Status-Transition T-Event Event))

(deffacts C-S-Message-Transfer "" ; 4/07/88 21:20:32
  (C-Slot-Value-Type Message-Transfer Transferred-Message Message))

(deffacts C-S-Action "" ; 4/11/88 14:55:02
  (C-Slot-Value-Member Action Action-Pettern May)
  (C-Slot-Value-Member Action Action-Pettern Must)
  (C-Slot-Value-Type Action Agent Object))

(deffacts C-S-Time-out "" ; 4/11/88 14:58:16
  (C-Slot-Value-Type Time-Out Status-B4-Timeout Status))

(deffacts C-S-Collision "" ; 4/11/88 14:59:03
  (C-Slot-Value-Type Collision Colliding-Messages Message)
  (C-Slot-How-Many Collision Colliding-Messages 2 Inf))
```

```
;;;
;;; Definition of Relations for Protocol Simulation
;;;

(defrelation Sim-Present-Status (?Status) ; 4/12/88 09:56:43
  "Status after the previous transition" )

(defrelation Sim-Prev-Status (?Status) ; 4/12/88 09:59:10
  "Status before the previous transition" )

(defrelation Sim-Prev-Transition (?Status-Transition) ; 4/12/88 10:03:10
  "Transition which made the status present" )

(defrelation Sim-Prev-Event (?Event) ; 4/12/88 10:05:48
  "Event which triggered off the privious transition" )

(defrelation Sim-Event (?Event) ; 4/12/88 10:10:51
  "Event which is ocuuring (and will be retracted soon)" )

(defrelation Sim-Initial-Status (?Status) ; 4/12/88 13:08:50
  "Status to start simulation from" )

(defrelation Sim-Start (?switch) ; 4/12/88 12:05:43
  "Control flag for simulation" )

;;;
;;; Definition of Rules for Protocol Simulation
;;;

(defrule Sim-Trigger-Transition "Make transition triggered by an event"
  ; 4/12/88 10:30:59
  ?f1 <- (Sim-Present-Status ?Status)
  ?f2 <- (Sim-Event ?Event)
  ?d1 <- (Sim-Prev-Status ?)
  ?d2 <- (Sim-Prev-Transition ?)
  ?d3 <- (Sim-Prev-Event ?)
  (split (
          (schema ?trans
                  (is-a+ Status-Transition)
                  (Pre-Status ?Status)
                  (T-Event ?Event)
                  (Next-Status ?Next-Status))
          =>
          (retract ?f2)
          (sprout (retract ?f1 ?d1 ?d2 ?d3)
                  (assert (Sim-Present-Status ?Next-Status)
                          (Sim-Prev-Status ?Status)
                          (Sim-Prev-Transition ?trans)
                          (Sim-Prev-Event ?Event))))
         (
          (not (schema ?trans
                       (is-a+ Status-Transition)
                       (Pre-Status ?Status)
                       (T-Event ?Event)))
          =>
          (retract ?f2)
          (sprout (retract ?f1 ?d1 ?d2 ?d3)
                  (assert (Sim-Present-Status undef)
                          (Sim-Prev-Status ?Status)
                          (Sim-Prev-Transition undef)
                          (Sim-Prev-Event ?Event))))))

(defrule Sim-Start-Initially "Create initial viewpoint in simulation"
  ; 4/12/88 13:05:45
  (Sim-Start on)
  (Sim-Initial-Status ?init-Stat)
  =>
  (sprout (assert (Sim-Prev-Status nil)
                  (Sim-Prev-Transition nil)
                  (Sim-Prev-Event nil)
                  (Sim-Present-Status ?init-Stat))))
```

```
;;; -*- Mode: ART; Base: 10.; Package: ART-USER -*-

;|
;|
;|    Requirement Model for X.25 Packet Level Interface
;|
;|       Written by   Naoki SHIBAMOTO
;|               from 04-APR-1988
;|
;|
;|
;|    Table of Contents
;|
;|      + Definition of Instances in X.25 Packet Level
;|
;|         - Main Agents
;|
;|         - Status, Time Limit, Messages, Message Transfers, and
;|           Status Transitions in Restart Procedure
;|
;|         - Status, Time Limit, Messages, Message Transfers, and
;|           Status Transitions in Call Procedure
;|
;|      + Definition of Initial Status
;|
```

```
;;;
;;; Definition of Instances in X.25 Packet Level
;;;

;;; Definition of Instances  -- Main Agents

(defschema DCE "Data Circuit Terminating Equipment" ; 4/07/88 14:15:55
  (is-a+ Object))
(defschema DTE "Data Terminal Equipment" ; 4/07/88 14:16:35
  (is-a+ Object))
(defschema Logical-Chanel "論理チャネル" ; 4/07/88 14:17:11
  (is-a+ Object))

(deffacts Connection-Definition "DCEとDTEがLogical-Channelで結ばれている"
  ; 4/07/88 17:35:54
  (Connection DCE ; primary
              DTE ; secondary
              Logical-Channel ; channel
              ))

;;; Definition of Instances  -- Status in Restart Procedure

(defschema R1 "" ; 4/07/88 14:19:12
  (is-a+ Status) (alias パケットレベルレディ)
  (Owner Logical-Chanel)
  (substatus P1 P2 P3 P4 P5 P6 P7))
(defschema R2 "" ; 4/07/88 14:20:42
  (is-a+ Status) (alias DTEリスタート要求)
  (Owner Logical-Chanel) (Time-limit T20))
(defschema R3 "" ; 4/07/88 14:24:31
  (is-a+ Status) (alias DCEリスタート指示)
  (Owner Logical-Chanel) (Time-limit T10))


;;; Definition of Instances  -- Time Limit in Restart Procedure

(defschema T20 "Time Limit of R2 Status" ; 4/07/88 14:24:57
  (is-a+ Timer-Value))
(defschema T10 "Time Limit of R3 Status" ; 4/07/88 14:25:52
  (is-a+ Timer-Value))


;;; Definition of Instances  -- Messages in Restart Procedure

(defschema リスタート要求 "" ; 4/07/88 14:28:05
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))
(defschema DCEリスタート確認 "" ; 4/07/88 14:31:32
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema リスタート指示 "" ; 4/07/88 14:32:39
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema DTEリスタート確認 "" ; 4/07/88 14:33:08
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))


;;; Definition of Instances  -- Message Transfers in Restart Procedure

(defschema E-リスタート要求 "" ; 4/07/88 15:43:59
  (is-a+ Message-Transfer) (Transferred-Message リスタート要求))
(defschema E-DCEリスタート確認 "" ; 4/07/88 15:45:28
  (is-a+ Message-Transfer) (Transferred-Message DCEリスタート確認))
(defschema E-リスタート指示 "" ; 4/07/88 15:45:59
  (is-a+ Message-Transfer) (Transferred-Message リスタート指示))
(defschema E-DTEリスタート確認 "" ; 4/07/88 15:46:24
  (is-a+ Message-Transfer) (Transferred-Message DTEリスタート確認))
```

```
;;; Definition of Instances  -- Status Transitions in Restart Procedure

; T-Action slots haven't yet been described.

(defschema ST-R-101 "" ; 4/07/88 14:34:19
  (is-a+ Status-Transition) (Pre-Status R1) (Next-Status R2)
  (T-Event E-リスタート要求))
(defschema ST-R-102 "" ; 4/07/88 14:37:08
  (is-a+ Status-Transition) (Pre-Status R2) (Next-Status R2)
  (T-event E-リスタート要求))
(defschema ST-R-103 "" ; 4/07/88 14:44:38
  (is-a+ Status-Transition) (Pre-Status R3) (Next-Status R1)
  (T-Event E-リスタート要求))                ; This is a collision case.
(defschema ST-R-202 "" ; 4/07/88 14:50:05
  (is-a+ Status-Transition) (Pre-Status R2) (Next-Status R1)
  (T-Event E-DCEリスタート確認))
(defschema ST-R-301 "" ; 4/07/88 14:56:53
  (is-a+ Status-Transition) (Pre-Status R1) (Next-Status R3)
  (T-Event E-リスタート指示))
(defschema ST-R-302 "" ; 4/07/88 14:57:40
  (is-a+ Status-Transition) (Pre-Status R2) (Next-Status R1)
  (T-Event E-リスタート指示))                ; This is a collision case.
(defschema ST-R-403 "" ; 4/07/88 14:59:19
  (is-a+ Status-Transition) (Pre-Status R3) (Next-Status R1)
  (T-Event E-DTEリスタート確認))
(defschema ST-R-503 "" ; 4/07/88 14:59:51
  (is-a+ Status-Transition) (Pre-Status r3) (Next-Status r3)
  (T-Event Transfer-of-any-Message-other-than-in-Restart-Procedure))
```

```
;;; Definition of Instances  -- Status in Call Procedure

(defschema P1 "" ; 4/07/88 16:01:46
  (is-a+ Status) (alias レディ)
  (Owner Logical-Chanel))
(defschema P2 "" ; 4/07/88 16:02:25
  (is-a+ Status) (alias DTE待機)
  (Owner Logical-Chanel) (Time-limit T21))
(defschema P3 "" ; 4/07/88 16:03:05
  (is-a+ Status) (alias DCE待機)
  (Owner Logical-Chanel) (Time-limit T11))
(defschema P4 "" ; 4/07/88 16:03:25
  (is-a+ Status) (alias データ転送)
  (Owner Logical-Chanel)
  (substatus d1 d2 d3))
(defschema P5 "" ; 4/07/88 16:04:10
  (is-a+ Status) (alias 呼衝突)
  (Owner Logical-Chanel))
(defschema P6 "" ; 4/07/88 16:04:44
  (is-a+ Status) (alias DTE復旧要求)
  (Owner Logical-Chanel) (Time-limit T23))
(defschema P7 "" ; 4/07/88 16:05:00
  (is-a+ Status) (alias DCE切断指示)
  (Owner Logical-Chanel) (Time-limit T13))


;;; Definition of Instances  -- Time Limit in Call Procedure

(defschema T21 "Time Limit of P2 Status" ; 4/07/88 16:19:43
  (is-a+ Timer-Value))
(defschema T11 "Time Limit of P3 Status" ; 4/07/88 16:23:10
  (is-a+ Timer-Value))
(defschema T23 "Time Limit of P6 Status" ; 4/07/88 16:36:15
  (is-a+ Timer-Value))
(defschema T13 "Time Limit of P7 Status" ; 4/07/88 16:36:46
  (is-a+ Timer-Value))


;;; Definition of Instances  -- Messages in Call Procedure

(defschema 発呼要求 "" ; 4/07/88 17:28:03
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))
(defschema 着呼 "" ; 4/07/88 17:31:02
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema 着呼受付 "" ; 4/07/88 17:42:14
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))
(defschema 接続完了 "" ; 4/07/88 17:43:11
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema 復旧要求 "" ; 4/07/88 17:43:38
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))
(defschema DCE復旧確認 "" ; 4/07/88 17:44:12
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema 切断指示 "" ; 4/07/88 17:44:50
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DCE) (Receiver-Type DTE))
(defschema DTE切断確認 "" ; 4/07/88 17:45:36
  (is-a+ Message) (for-Control T) (for-Information Nil)
  (Sender-Type DTE) (Receiver-Type DCE))
```

```
;;; Definition of Instances  -- Message Transfers in Call Procedure

(defschema E-発呼要求 "" ; 4/07/88 19:04:00
  (is-a+ Message-Transfer) (Transferred-Message 発呼要求))
(defschema E-着呼 "" ; 4/07/88 19:04:22
  (is-a+ Message-Transfer) (Transferred-Message 着呼))
(defschema E-着呼受付 "" ; 4/07/88 19:04:46
  (is-a+ Message-Transfer) (Transferred-Message 着呼受付))
(defschema E-接続完了 "" ; 4/07/88 19:04:58
  (is-a+ Message-Transfer) (Transferred-Message 接続完了))
(defschema E-復旧要求 "" ; 4/07/88 19:05:14
  (is-a+ Message-Transfer) (Transferred-Message 復旧要求))
(defschema E-DCE復旧確認 "" ; 4/07/88 19:05:32
  (is-a+ Message-Transfer) (Transferred-Message DCE復旧確認))
(defschema E-切断指示 "" ; 4/07/88 19:05:56
  (is-a+ Message-Transfer) (Transferred-Message 切断指示))
(defschema E-DTE切断確認 "" ; 4/07/88 19:06:48
  (is-a+ Message-Transfer) (Transferred-Message DTE切断確認))


;;; Definition of Instances  -- Status Transitions in Call Procedure

; T-Action slots hevn't yet been described.

(defschema ST-P-101 "" ; 4/07/88 17:47:45
  (is-a+ Status-Transition) (Pre-Status P1) (Next-Status P2)
  (T-Event E-発呼要求))
(defschema ST-P-103 "" ; 4/07/88 18:18:06
  (is-a+ Status-Transition) (Pre-Status P3) (Next-Status P5)
  (T-Event E-発呼要求))                          ; This is a collision case.
(defschema ST-P-201 "" ; 4/07/88 18:20:15
  (is-a+ Status-Transition) (Pre-Status P1) (Next-Status P3)
  (T-Event E-着呼))
(defschema ST-P-202 "" ; 4/07/88 18:20:34
  (is-a+ Status-Transition) (Pre-Status P2) (Next-Status P5)
  (T-Event E-着呼))
(defschema ST-P-303 "" ; 4/07/88 18:21:11
  (is-a+ Status-Transition) (Pre-Status P3) (Next-Status P4)
  (T-Event E-着呼受付))
(defschema ST-P-402 "" ; 4/07/88 18:21:52
  (is-a+ Status-Transition) (Pre-Status P2) (Next-Status P4)
  (T-Event E-接続完了))
(defschema ST-P-405 "" ; 4/07/88 18:22:16
  (is-a+ Status-Transition) (Pre-Status P5) (Next-Status P4)
  (T-Event E-接続完了))
(defschema ST-P-501 "" ; 4/07/88 18:22:36
  (is-a+ Status-Transition) (Pre-Status P1 P2 P4 P5 P6) (Next-Status P6)
  (T-Event E-復旧要求))
(defschema ST-P-503 "" ; 4/07/88 18:24:48
  (is-a+ Status-Transition) (Pre-Status P3) (Next-Status P6)
  (T-Event E-復旧要求) (intention 着呼拒否))
(defschema ST-P-507 "" ; 4/07/88 18:25:48
  (is-a+ Status-Transition) (Pre-Status P7) (Next-Status P1)
  (T-Event E-復旧要求))                          ; This is a collision case.
(defschema ST-P-606 "" ; 4/07/88 18:27:46
  (is-a+ Status-Transition) (Pre-Status P6) (Next-Status P1)
  (T-Event E-DCE復旧確認))
(defschema ST-P-701 "" ; 4/07/88 18:28:17
  (is-a+ Status-Transition) (Pre-Status P1 P3 P4 P5 P7) (Next-Status P7)
  (T-Event E-切断指示))
(defschema ST-P-702 "" ; 4/07/88 18:28:50
  (is-a+ Status-Transition) (Pre-Status P2) (Next-Status P7)
  (T-Event E-切断指示) (intention 不完了呼))
(defschema ST-P-706 "" ; 4/07/88 18:29:18
  (is-a+ Status-Transition) (Pre-Status P6) (Next-Status P1)
  (T-Event E-切断指示))                          ; This is a collision case.
(defschema ST-P-807 "" ; 4/07/88 18:30:52
  (is-a+ Status-Transition) (Pre-Status P7) (Next-Status P1)
  (T-Event E-DTE切断確認))

;;; Definition of Initial Status
(deffacts Initial-Status "Initial Status of X.25 Packet Level" ; 4/12/88 13:16:55
  (Sim-Initial-Status R1))
```

```
;;; -*- Mode: LISP; Base: 10.; Syntax: Common-lisp; Package: ACU -*-

;;;
;;; Programed by Naoki SHIBAMOTO
;;;           for Requirement Model prototyping in KANT using ART
;;;
;;;   "RM" leading predicate names stands for "Requirement Model"
;;;
;;;    written 4/07/88 11:30:22
;;;

;-- Predicates for number attributes

(defun RM-evenp (n)
  (and (numberp n) (evenp n)))

(defun RM-oddp (n)
  (and (numberp n) (oddp n)))

(defun RM-minusp (n)
  (and (numberp n) (minusp n)))

(defun RM-plusp (n)
  (and (numberp n) (plusp n)))

(defun RM-zerop (n)
  (and (numberp n) (zerop n)))


;-- Predicates for order of numbers
;     (used in Checking constraints on slot-how-many

(defun RM-in-bound-p (lower upper i)
  (and (numberp i)
       (cond ((numberp lower)
              (<= lower i))
             (t nil))
       (cond ((numberp upper)
              (<= i upper))
             ((equal upper 'inf) t)
             (t nil))))

(defun RM-out-bound-p (lower upper i)
  (not (RM-in-bound-p lower upper i)))
```

```
;;; -*- Mode: LISP; Base: 10.; Syntax: Common-lisp; Package: ACU -*-

(defvar *Dynadefrule-Default-Pathname* "Clm06:>Shibamoto>R-Model>Temp-for-defrule.art")


(defun RM-Dynamic-Defrule
       (Rule-name document Rule-body
        &optional (immediate-load-p t) (pathname *Dynadefrule-Default-Pathname*))
  (with-open-file (stream pathname :direction :output)
    (format stream ";;; -*- Mode: ART; Base: 10.; Package: ART-USER -*-~%")
    (format stream ";;; generated by RM-Dynamic-Defrule ~\\datime\\~%~%")
    (format stream "(DEFRULE ")
    (write Rule-name :stream stream)
    (format stream " \"~a\"~%" document)
    (loop for element in Rule-body do
      (cond
        ((stringp element)
         (format stream "~a" element))
        ((characterp element)
         (write-char element stream))
        ((symbolp element)
         (write-char #\Space stream)
         (write element :stream stream)
         (write-char #\Space stream))))
    (format stream " )~%"))
  (when immediate-load-p
    (art-load pathname)))
```