ソフトウェア工学 89－1
（1992. 12. 14）

Zを用いたプロトコルの仕様記述

菅原一伸† 高橋薫†† 野口正一† 栗田陽子††† 安藤敏彦††† 加藤靖†††

† 東北大学応用情報学研究センター
†† 東北大学電気通信研究所
††† 仙台電波工業高等専門学校

一般のソフトウエアシステムの仕様記述用にZが提案されている。また、通信分散系への適用の試みも行なわれてきている。そこで、本論文では、Zを用いてアブラカダブラサービスの仕様を記述し、プロトコル仕様へのZの適用可能性および有効性について述べる。アブラカダブラサービスは状態遷移モデルを用いてモデル化し、サービスとユーザの間のサービスプリミティブの交換の観点から仕様を記述する。サービスプリミティブの交換はZのオペレーションスキーマに対応させて記述を行なっている。記述した結果については他のFDTとの比較も交えて、評価を行なう。

Protocol Specification Using Z

Kazunobu Sugawara† Kaoru Takahashi†† Shoichi Noguchi†
Yoko Kurita††† Toshihiko Ando††† Yasushi Kato†††

† Research Center for Applied Information Sciences, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, 980 JAPAN

†† Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, 980 JAPAN

††† Sendai National College of Technology
1, Kitahara, Kamiayashi, Aoba-ku, Sendai, 989-31 JAPAN

It has been proposed specification language Z for the specification of general software systems. And also it has been attempted to apply Z for the specification of destributed communication system. In this paper we describe Abracadabra Service specification in Z and show applicability and availability of using Z for protocol specification. We model Abracadabra Service using state transition model and describe the specification in terms of the exchange of the service primitives between the user and the service. This exchange is represented by operation schemas of Z. Furthermore, we evaluate our specification in comparison with Estelle and LOTOS.

1 はじめに

プロトコル、分散システムなどを円滑に開発するためには、その仕様を曖昧なく厳密に表現する必要がある。また、検証の目的のため、適切な数学モデルに基づいた言語を用いることが肝要である。

そのため、これまで、それらのシステムの厳密な仕様化のため、種々の形式記述技法（FDT, Formal Description Technique）が開発されてきている。代表的なものとしては、ISO の LOTOS[1]、Estelle[2]、CCITT の SDL[3] があり、OSI アーキテクチャや ISDN システムなどの仕様記述に広く使われてきている。

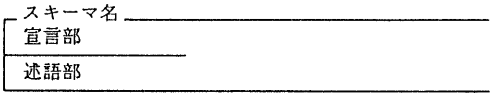
一方、一般のソフトウエアシステムの仕様記述用に、Z[4] [5] が提案されている。そして、Z 及びそのオブジェクト指向版の Object-Z[6] は、通常のソフトウエアシステムだけでなく、通信分散系への適用の試みも近年活発に行なわれている [6] [7] [8]。

本論文では、これらの動きと並行して、プロトコル仕様記述への Z の適用可能性および有効性を調べることを目的とし、よく知られたアブラカダブラサービス（Abracadabra Service）[9] の Z による仕様記述を行なう。同時に、この適用例を通して、他の FDT の仕様記述との比較評価をおこなう。

本論文の構成は次の通りである。第 2 節では、Z について簡単に説明する。第 3 節では、本論文の記述対象であるアブラカダブラサービスの概要を述べる。第 4 節は本論であり、アブラカダブラサービスの Z による仕様記述上のモデルおよび、それに基づいた Z 仕様を構成する。そして第 5 節では、その評価を行なう。第 6 節は結びである。

2 Z

Z はオックスフォード大学で開発された汎用的な仕様記述言語である。Z では、仕様化対象のシステムを、その状態およびシステムに作用するオペレーションを述語論理と集合論に基づいて定義することで、仕様化を行なう。状態とオペレーションはスキーマと呼ばれる構造を用いて記述される。スキーマには状態スキーマ、初期状態スキーマ、オペレーションスキーマがある。状態スキーマはシステムの可能な状態空間を表す。初期状態スキーマはシステムの初期状態を表し、状態スキーマで定義された状態空間の部分集合となる。オペレーションスキーマは対応するオペレーションの実行と、それによるシステムの状態の変化を表すのに使われる。これらのスキーマを用いてシステムをモデル化することが Z の特徴となっている。具体的にスキーマは次のように記述される。



宣言部には変数とその変数の型を “変数 : 型” のように記述する。述語部には宣言部で宣言された変数間の関係を述語論理を用いて記述する。変数は通常、状態を表す。変数は、?、!、′で装飾されることがあり、? は入力を表し、! は出力を表す。′ はオペレーションの実行によって、状態変化した後の状態を表すのに使われる。宣言部にはスキーマ名も書くことができる。この場合宣言部はマージされて、述語部はこのスキーマの述語部と宣言されたスキーマの述語部の合接となる。慣例として頭に Δ をつけてスキーマ名を宣言すると、そのスキーマで記述された状態の状態変化を表す。頭に Ξ をつけた場合には、状態変化がないことを表す。

また Z では同じ構造だが名前だけが異なるスキーマを作るのに名前換え（renaming）の機能が用意されている。

3 アブラカダブラサービス

本論文の記述対象であるアブラカダブラサービスは一対のユーザ間で作用する信頼のある双方向のコネクション型のサービスを提供するものである（図 1）。サポートされるサービスプリミティブは次の通りである。

ConReq	コネクション確立要求
ConInd	コネクション確立指示
ConResp	コネクション確立応答
ConConf	コネクション確立確認
DatReq	データ転送要求
DatInd	データ転送指示
DisReq	コネクション解放要求
DisInd	コネクション解放指示

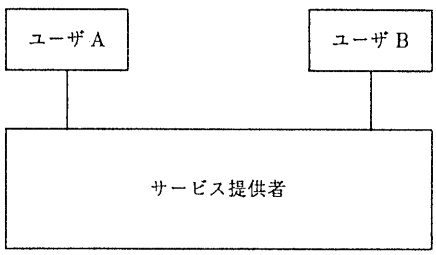


図 1: アブラカダブラサービス

コネクションを確立する時には、ユーザがサービス提供者に ConReq を発する。この場合、通常、相手側のユーザがサービス提供者から ConInd を受けとり、その応答と

してConRespを発する。そして、ConReqを発したユーザがConConfを受けとり、コネクションが確立される。しかし、双方が同時にコネクションの確立を行なうなら、ユーザにConIndは渡されずにConConfが渡される。なお、コネクション確立はDisReqを発することによって拒否または取り消しをすることができる。

コネクションが確立されると、各ユーザは相手側にDatIndとして渡されるであろうDatReqをサービス提供者に発することができる。

データメッセージはコネクションの解放が発生する時を除いて順序と内容が保存される。コネクションの解放が発生する時は、サービス提供者中に残っているデータメッセージは失われる可能性がある。

各ユーザは確立されたコネクションをDisReqを発することによって解放することができる。これは通常はDisIndとして相手側に渡される。しかし、同時にDisReqが発せられた場合には、DisIndは渡されずにすぐにコネクションは解放される。サービス提供者自身がコネクションを解放することができる。各ユーザはこのことをDisIndによって知らされる。しかしDisIndを受けとる前にDisReqを発するならDisIndは渡されない。

一度コネクションが解放されると各ユーザはConReqを発することによって新たにコネクションの確立を行なうことができる。

通常のサービスプリミティブの順番を図2に示す。

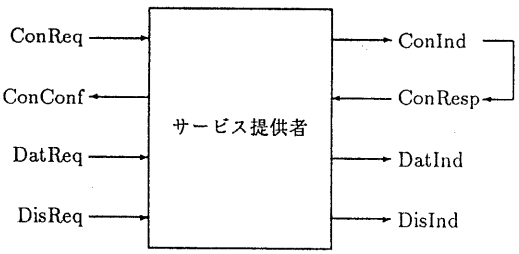


図 2: 通常のサービスプリミティブの順番

4 アブラカダブラサービスの仕様記述

4.1 サービスのモデル化

4.1.1 基本的な考え方

本節では、前節で述べたアブラカダブラサービスをZで記述する際の記述モデルについて議論する。

アブラカダブラサービスの仕様は図1のサービス提供者を記述することで与える。サービス提供者は互いに独立なユーザAとユーザBへのサービスを提供するものであるから、各ユーザに対して、サービス提供者をProviderAとProviderBに分けてモデル化する。また、ユーザ間のデータメッセージ（SDU, Service Data Unit）を表すのにProviderAとProviderBの間にキューを構成してモデル化する。このキューはSDUの系列であり、ProviderAからProviderB、そしてProviderBからProviderAへの2つからなると考える。前節で述べたことからこの内容はコネクションが解放される時を除いてSDUの順序と内容を保存する。ユーザとサービス提供者の間のサービスプリミティブについては、Zのオペレーションと対応づける。以上のモデル化より、Z仕様におけるアブラカダブラサービスの状態スキーマはProviderAとProviderBの状態、そしてSDU系列からなる2つのキューで構成することができる。（図3）

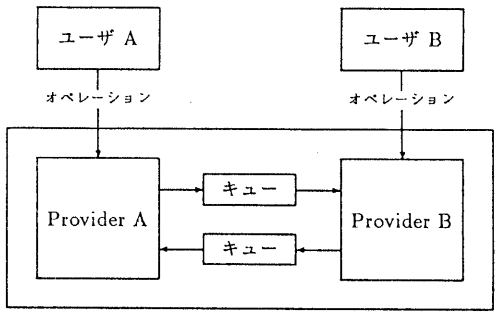


図 3: サービスの記述上のモデル

ProviderAとProviderBの状態および状態変化は従来の状態遷移モデルを用い、サービスプリミティブの発生と連動して、次項のように表す。

4.1.2 ProviderAとBの状態

ProviderAとProviderBの状態を次の5つから構成する。

idle	コネクションを解放した後の状態（初期状態）
calling	ユーザからConReqを受けとった後の状態
called	ユーザにConIndを渡した後の状態
connected	コネクションが確立した状態
disconnecting	コネクションの解放を行なっている状態

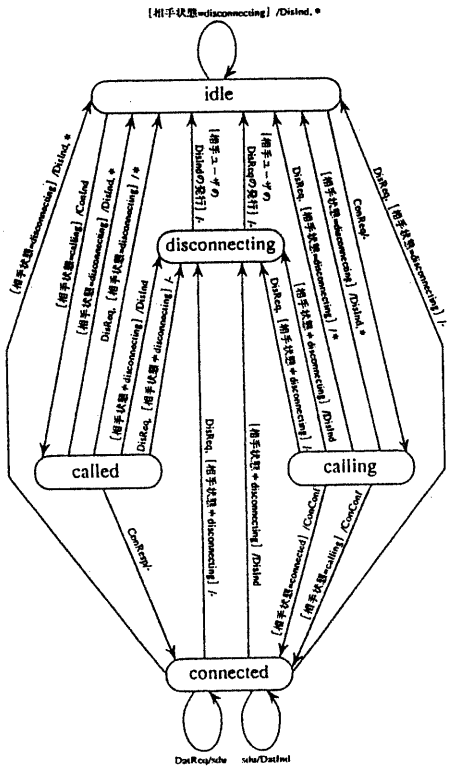


図 4: ProviderA と ProviderB の状態遷移図

これらの状態間の変化は、3節のアブラカダブラサービスの定義を考慮し、図4に示す状態遷移図で表す。

図4の中で、矢印はその方向に状態遷移が可能であることを示しており、矢印についている添字は状態遷移にともなう入力、条件、出力、操作などを示している。"/" の前は入力を、後ろは出力を示している。条件は "[ ]" で示している。"*" は状態遷移にともなって、相手状態を idle にして、SDUs を空にする操作を示している。

例として、図2に示したような通常の場合について考える。最初は ProviderA も ProviderB も idle 状態であるとする。この時にユーザ A が ConReq を発した場合、ProviderA は idle 状態から calling 状態に遷移する。ProviderB は ProviderA が calling 状態の時、ユーザ B に ConInd を渡して idle 状態から called 状態に遷移することができる。さらに、この状態でユーザ B が ConResp を発すると、ProviderB は called 状態から connected 状態に遷移する。ProviderA は ProviderB が connected 状態の時、ユーザ A に ConConf を渡して calling 状態から connected 状態に遷移することができる。connected 状態に遷移した ProviderA と ProviderB は、ユーザが DatReq を発した場合、DatReq のパラメータである SDU をキューの後ろに加える。また、キューが空でない場合、キューの先頭の SDU をパラメータとして DatInd をユーザに渡すことができる。

ユーザ A が DisReq を発すると、ProviderA は disconnecting 状態に遷移する。ProviderB は ProviderA が disconnecting 状態の時、ユーザ B に DisInd を渡し、idle 状態に遷移することができる。この時 ProviderA を idle 状態に遷移させ、キューを空にするという操作が行なわれる。

4.2 アブラカダブラサービスの Z 仕様

3節で述べたアブラカダブラサービスを、4.1 のサービスのモデル化に基づいて、以下のように Z で仕様化する。

4.2.1 SDU および ProviderA と B の状態とサービスプリミティブ

全ての Service Data Unit からなる集合 SDU を Z の基本型 (Basic Type) を用いて導入する。

$[SDU]$

基本型とは、ある集合をそれらがすでに知られているものとして定義する型のことである。

また、4.1.2 で示したようなアブラカダブラサービスの *ProviderA* と *ProviderB* の状態を $STATE$ 型、3節に示したサービスプリミティブを SP 型として以下のように定義する。

$$\begin{aligned} STATE ::= & \mathit{idle} \mid \mathit{calling} \mid \mathit{called} \\ & \mid \mathit{connected} \mid \mathit{disconnecting} \\ SP ::= & \mathit{ConReq} \mid \mathit{ConInd} \mid \mathit{ConResp} \\ & \mid \mathit{ConConf} \mid \mathit{DatReq} \mid \mathit{DatInd} \\ & \mid \mathit{DisReq} \mid \mathit{DisInd} \end{aligned}$$

ここで、$STATE$ 型、SP 型は Z の自由型 (Free Type) を用いて定義している。自由型とは、ある型をそれに相当するすべての要素を列挙することで定義するものである。

4.2.2 状態空間

4.1 のサービスのモデル化に従いアブラカダブラサービスの状態空間を、スキーマ *ProviderA*、*ProviderB*、*AtoB_SDUs*、*BtoA_SDUs* を用いて以下のようにスキーマ *AbraService* として構成する。

AbraService

$ProviderA$
$ProviderB$
$AtoB\_SDUs$
$BtoA\_SDUs$

ProviderA

$stateA : STATE$

AtoB_SDUs

$AtoB\_SDUs : \text{seq}\ SDU$

ProviderB

$ProviderA[stateB/stateA]$

BtoA_SDUs

$AtoB\_SDUs[BtoA\_SDUs/AtoB\_SDUs]$

ここで、名前換え (renaming) を利用している。名前換えは、同じ構造を持つスキーマが存在し、変数の名前のみが異なる場合に、"スキーマ名 [新しい変数名 / もとの変数名]" のように表記して、変数の名前を置き換える機能を持つ。また、seq SDU は、SDU 型の要素の系列を表す。

4.2.3 初期状態

アブラカダブラサービスの初期状態は、*ProviderA* と *ProviderB* の状態がともに *idle* 状態で、*ProviderA* から *ProviderB*、*ProviderB* から *ProviderA* のそれぞれのキューが空の時であるので、次のように定義できる。

InitAbraService

$AbraService$

$stateA = stateB = idle$
$AtoB\_SDUs = BtoA\_SDUs = \langle\rangle$

ここで$\langle\rangle$は、空系列を意味する。

4.2.4 オペレーション

4.1 節で説明した状態遷移図をモデルとして、アブラカダブラサービスのオペレーションを Z のオペレーションを用いて以下のように記述する。その際、*ProviderA* と *ProviderB* は同じ構造になるので、ここでは主として *ProviderA* について説明を加えていく。

ProviderA

コネクションを確立するためには、ユーザ A は *ProviderA* に *ConReq* を発する。ユーザ A から *ConReq* を受けとった *ProviderA* は *idle* 状態から *calling* 状態に遷移する。

ConReqA

$\Delta ProviderA$
$sp? : SP$

$stateA = idle$
$sp? = ConReq$
$stateA' = calling$

ユーザ B が *ConReq* を発すると、*ProviderA* はユーザ A に *ConInd* を渡すことができる。そして、*ProviderA* は *idle* 状態から *called* 状態に遷移する。

ConIndA

$\Delta ProviderA$
$\Xi ProviderB$
$sp! : SP$

$stateA = idle$
$stateB = calling$
$stateA' = called$
$sp! = ConInd$

ConInd を受けとったユーザ A は、その応答として *ConResp* を *ProviderA* に発することができる。そして、*ProviderA* は *called* 状態から *connected* 状態に遷移する。

ConRespA

$\Delta ProviderA$
$sp? : SP$

$stateA = called$
$sp? = ConResp$
$stateA' = connected$

ユーザ B が *ConResp* を発して *connected* 状態になった時、または、双方が同時にコネクションの確立を行なったため *ProviderB* が *calling* 状態の時、*ProviderA* はユーザ A に *ConConf* を渡しコネクションが確立する。

ConConfA

$\Delta ProviderA$
$\Xi ProviderB$
$sp! : SP$

$stateA = calling$
$stateB \in \{connected, calling\}$
$stateA' = connected$
$sp! = ConConf$

コネクションが確立されている時、データを転送するためにはユーザ A は $DatReq$ を $ProviderA$ に発して、データメッセージを入力する。入力されたメッセージは $AtoB\_SDUs$ の最後に加えられる。

DatReqA

$\Xi ProviderA$
$\Delta AtoB\_SDUs$
$sp? : SP$
$sdu? : SDU$

$stateA = connected$
$sp? = DatReq$
$AtoB\_SDUs' = AtoB\_SDUs \frown \langle sdu? \rangle$

$\frown$ は $\frown$ の前後に書かれた系列を連接したものを、結果の系列とする関数である。

コネクションが確立されている時、ユーザ B からデータメッセージが送られた場合、ユーザ A は $ProviderA$ から $DatInd$ とユーザ B からのメッセージを受けとることができる。

DatIndA

$\Xi ProviderA$
$\Delta BtoA\_SDUs$
$sp! : SP$
$sdu! : SDU$

$stateA = connected$
$BtoA\_SDUs \neq \langle\rangle$
$sp! = DatInd$
$BtoA\_SDUs = sdu! \frown BtoA\_SDUs'$

$ProviderA$ の状態が $calling$、$called$、$connected$ の時、コネクションを解放するためには、ユーザ A は $ProviderA$ に $DisReq$ を発する。その時、$ProviderB$ が $disconnecting$ 状態以外ならば、$ProviderA$ は $disconnecting$ 状態に遷移する。もし $ProviderB$ が $disconnecting$ 状態ならば、$ProviderA$、$ProviderB$ 共に $idle$ 状態になり、コネクションは解放される。この時キューの中に残っているデータメッセージは失われる。

DisReqA

$\Delta AbraService$
$sp? : SP$

$(stateA \in \{calling, called, connected\} \land$
$sp? = DisReq \land$
$stateB \neq disconnecting \land$
$stateA' = disconnecting \land$
$stateB' = stateB \land$
$AtoB\_SDUs' = AtoB\_SDUs \land$
$BtoA\_SDUs' = BtoA\_SDUs)$
$\lor$
$(stateA \in \{calling, called, connected\} \land$
$sp? = DisReq \land$
$stateB = disconnecting \land$
$stateA' = stateB' = idle \land$
$AtoB\_SDUs' = BtoA\_SDUs' = \langle\rangle)$

ユーザ B が $DisReq$ を発すると、$ProviderA$ はユーザ A に $DisInd$ を渡し、$ProviderA$、$ProviderB$ を $idle$ 状態にしてコネクションを解放する。この時キューの中に残っていたデータメッセージは失われる。$ProviderA$ または $ProviderB$ によってコネクションの解放が行なわれる場合に、$ProviderB$ が $disconnecting$ 状態以外ならば、ユーザ A に $DisInd$ を渡し $ProviderA$ を $disconnecting$ 状態に遷移させる。

DisIndA

$\Delta AbraService$
$sp! : SP$

$(stateA \in \{idle, calling, called, connected\} \land$
$stateB = disconnecting \land$
$stateA' = stateB' = idle \land$
$AtoB\_SDUs' = BtoA\_SDUs' = \langle\rangle \land$
$sp! = DisInd)$
$\lor$
$(stateA \in \{calling, called, connected\} \land$
$stateB \neq disconnecting \land$
$stateA' = disconnecting \land$
$stateB' = stateB \land$
$BtoA\_SDUs' = BtoA\_SDUs \land$
$AtoB\_SDUs' = AtoB\_SDUs \land$
$sp! = DisInd)$

ProviderB

$ProviderB$ では、$ProviderA$ で定義したオペレーションを名前換えの機能を利用して以下のように簡潔に記述できる。

ConReqB

ConReqA [*stateB*/*stateA*, *stateB'*/*stateA'*]

ConIndB

ConIndA [*stateB*/*stateA*, *stateA*/*stateB*,
stateB'/*stateA'*, *stateA'*/*stateB'*]

ConRespB

ConRespA [*stateB*/*stateA*, *stateB'*/*stateA'*]

ConConfB

ConConfA [*stateB*/*stateA*, *stateA*/*stateB*,
stateB'/*stateA'*, *stateA'*/*stateB'*]

DatReqB

DatReqA [*stateB*/*stateA*, *stateB'*/*stateA'*,
BtoA_SDUs/*AtoB_SDUs*,
BtoA_SDUs'/*AtoB_SDUs'*]

DatIndB

DatIndA [*stateB*/*stateA*, *stateB'*/*stateA'*,
AtoB_SDUs/*BtoA_SDUs*,
AtoB_SDUs'/*BtoA_SDUs'*]

DisReqB

DisReqA [*stateB*/*stateA*, *stateA*/*stateB*,
stateB'/*stateA'*, *stateA'*/*stateB'*,
BtoA_SDUs/*AtoB_SDUs*,
AtoB_SDUs/*BtoA_SDUs*,
BtoA_SDUs'/*AtoB_SDUs'*,
AtoB_SDUs'/*BtoA_SDUs'*]

DisIndB

DisIndA [*stateB*/*stateA*, *stateA*/*stateB*,
stateB'/*stateA'*, *stateA'*/*stateB'*,
BtoA_SDUs/*AtoB_SDUs*,
AtoB_SDUs/*BtoA_SDUs*,
BtoA_SDUs'/*AtoB_SDUs'*,
AtoB_SDUs'/*BtoA_SDUs'*]

5 評価

本節では、前節で与えたアブラカダブラサービスの仕様記述を通してZ仕様の評価を行なう。評価項目としては、仕様の抽象度、簡潔性、理解性を対象とする。比較評価においては、文献[9]で与えられているアブラカダブラサービスのEstelle仕様とLOTOS仕様を用いる。

5.1 抽象度

Zは状態をベースとした言語であるから、前節のZ仕様にあるようにサービスの状態が陽に仕様上に反映される。しかしこれは、サービスの具体的な実現構造を表すものではない。また、サービスとユーザの間のサービスプリミティブの交換はオペレーションスキーマにより直接に表され、交換に関連するサービス動作の条件と効果は、その実現の仕方とは全く独立に論理式を主体として数学的に表現される。従って、Z仕様は比較的抽象度の高い実現独立な仕様である。

Estelle仕様では、モジュールやモジュール間のチャネルの概念とともに、仕様上にサービスの実現構造が一部現れた形となる。そしてサービス全体は、通信を伴って並行動作するモジュール群の状態遷移の観点から記述される。各モジュールの状態と状態遷移は陽に記述され、処理があればその内容がPascal風に記述される。従って、抽象度の比較的低い実現向きの仕様が構成される。

LOTOSには種々の記述スタイルがあり、このサービスの記述では制約指向スタイル（constraint-oriented style）が採用されている。仕様は、ユーザ毎に独立なサービスプリミティブの順番と内容に関する制約、および、ユーザと他のユーザとのサービスプリミティブ間のグローバルな関係を与える制約の組合せとして表現される。従って、Zと同様、LOTOS仕様はサービスの実現とは独立な抽象的な仕様となる。

5.2 簡潔性

多少の粗さはあるが、ここでは記述の行数により、Z、Estelle、LOTOSのそれぞれの仕様の簡潔性を評価する。

前節で記述したように、Z仕様の行数は132行である。これは、垂直形（vertical form）で分かりやすく表現したものであり、水平形（horizontal form）で表せばより簡潔になる。Estelle仕様においては254行である。一方、LOTOS仕様は329行である。

Z仕様の簡潔さは、あらかじめZに用意されている記法上の慣例や種々の関数、また、集合表記などを、仕様を記述する上で効果的に使用できることにあると考えられる。興味深いものとして、前節のスキーマ *DatInd* 中の述語

$$BtoA\_SDUs = sdu! \frown BtoA\_SDUs'$$

がある。これはオペレーション後の $BtoA\_SDUs$ と sdu の出力を非常に簡潔に表現している。

Estelle 仕様の多少の複雑さは、モジュールの状態と状態遷移を網羅的に列挙しなければならないことと、モジュールの初期化やそれに伴うモジュール間の結合などを仕様中に含む必要があることによる。

LOTOS 仕様がもっとも複雑で長い。これは主として、抽象データ型の定義部分の多さによる。LOTOS は Z と異なり、組み込みの型や関数が少なく、また、省略記法の用意がなされていない。そのため、仕様化対象によらず、通常、抽象データ型の部分が仕様中の多くを占めることとなる。

5.3 理解性

仕様の理解性を左右する要因は、使用する言語に対する習熟度や言語の背景となるモデルへの精通によるところが多い。この観点から、集合、関数、一階論理などの数学的知識を持つものは Z 仕様への理解性が高いと考えられる。また、前項で述べた簡潔さの点からも理解性が高い。一方、通常のソフトウエア技術者は、Estelle 仕様の方が手続き的であり、より理解がしやすい。状態遷移の概念さえ知っていれば、実際、アブラカダブラサービスの Estelle 仕様は容易に理解できる。LOTOS 仕様については、プロセスの概念と抽象データ型の代数的記述法の概念の両方を知る必要があり、さらに、それらが一つの仕様中に相補的に現れるため、仕様が複雑になった場合は理解性が損なわれることになる。また、アブラカダブラサービスの LOTOS 仕様は、制約指向スタイルについての知識も必要とし、読解性に難がある。

6 むすび

本論文では、アブラカダブラサービスの仕様記述を通してプロトコル分野への Z の適用性と有効性を示した。具体的には、このサービスの記述モデルを考察し、サービスプリミティブの交換の観点から対応する Z 仕様を構成した。そして、この記述を土台として他の FDT による形式記述との比較評価を行ない、この Z 仕様が比較的簡潔であり、また、抽象度の高いものとなっていることを確認した。

今後の課題としては、（1）仕様のモジュール性などをより高めるため Object-Z を使用すること、（2）プロトコル仕様記述の際の有用な汎用的あるいは生成的なスキーマをライブラリ的に使えるよう具備すること、（3）アブラカダブラプロトコルの仕様記述とともにそのサービス仕様との間の関係の検証を行なうことなどがある。

参考文献

[1] ISO : Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, *ISO 8807* (1989).

[2] ISO : Information Processing Systems - Open Systems Interconnection - Estelle - A Formal Description Technique based on an Extended State Transition Model, *ISO 9074* (1989).

[3] CCITT : Functional Specification and Description Language(SDL), *CCITT Recommendation Z.100* (1988).

[4] Spivey, J. M. : The Z Notation - A Reference Manual, *Prentice Hall* (1989).

[5] Diller, A. : Z - An Introduction To Formal Methods, *John Wiley and Sons* (1990).

[6] Duke, R. , King, P. , Rose, G. and Smith, G. : The Object-Z Specification Language Version 1, *Technical Report No. 91-1, The University of Queensland* (1991).

[7] Duke, R. , Hayes, I. , King, P. , Rose, G. : Protocol Specification and Verification Using Z, *Protocol Specification Testing and Verification VIII*, pp.33-46, *North-Holland* (1989).

[8] Hayes, I. J. , Mowbray, M. , Rose, G. A. : Signalling System No.7 The Network Layer, *Protocol Specification Testing and Verification IX*, pp.3-14, *North-Holland* (1990).

[9] ISO: Information technology - Open Systems Interconnection - Guidelines for the application of Estelle, LOTOS and SDL, *ISO/IEC TR 10167* (1991).

[10] Wing, J. M. : A Specifier's Introduction to Formal Methods, *IEEE COMPUTER* (1990).