TTC 標準
TTC STANDARD

JT-T125
オーディオグラフィック会議のための多地点通信サービス －プロトコル仕様

Multipoint Communication Service for Audiographic and Audiovisual Teleconferencing - Protocol Definition

第1版

1995年11月28日制定

社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

オーディオ グラフィック会議のための多地点通信サービス　－プロトコル仕様

＜参考＞

１．国際勧告等との関連

本標準は、オーディオ グラフィック会議における、多地点通信サービス規定するもので、決議１による郵便投票により１９９４年４月に承認されたITU-T勧告T.125に準拠したものである。

２．上記国際勧告等に対する追加項目等

２．１　オプション選択項目

なし

２．２　ナショナルマター決定事項

なし

２．３　その他

(1) 本標準は、上記ＩＴＵ－Ｔ勧告に対し、先行している項目はない。

(2) 本標準は、上記ＩＴＵ－Ｔ勧告に対し、追加した項目はない。

(3) 本標準は、上記ＩＴＵ－Ｔ勧告に対し、削除した項目はない。

２．４　原勧告との章立て構成比較表

本標準では、 第１章に「概要」を追加し、原勧告第２章「参照」を本＜参照＞中に移動したため、 Ｔ．１２５勧告に対して、「範囲」が１章から２章に繰り下がっているが、３章以降は原勧告と同一の章立てとなっている。

３．改版の履歴

版　数	制　定　日	改　版　内　容
第１版	平成７年１１月２８日	制　定

４．その他

(1) 参照している勧告、標準等

ＴＴＣ標準　　：ＪＴ－Ｔ１２２、ＪＴ－Ｔ１２３、

ＩＴＵ－Ｔ勧告　：Ｘ．２００、Ｘ．２１４、Ｘ．２０８、Ｘ．２０９

ＩＳＯ／ＩＥＣ標準　：ＩＳＯ／ＩＥＣ　ＤＩＳ８８２５－２

５．工業所有権

本標準に関わる「工業所有権の実施の権利に係る確認書」の提出状況は、ＴＴＣホームページでご覧になれます。

目　次

1．概要

この標準は多地点通信ドメインの階層を通じて動作するプロトコルを定義する。この標準はプロトコルメッセージのフォーマットと、一組のトランスポート接続上でのプロトコルメッセージ交換を制御する手続きについて詳述している。プロトコルの目的はTTC標準JT-T122で定義された多地点通信サービスを実装することである。

2．範囲

この標準は、以下を詳述する。

a. ある MCS プロバイダから相手の MCS プロバイダへのデータおよび制御情報の転送に対するプロトコル手続き。

b. データおよび制御情報の転送に使用される MCS プロトコル データ ユニットの構造およびコード化。

手続きは、以下のように定義される。

a. MCS プロトコル データ ユニットの交換によるMCS プロバイダの間の相互動作。

b. MCS プリミティブの交換による MCS プロバイダと MCS ユーザの間の相互動作。

c. トランスポートサービス プリミティブの交換による MCS プロバイダおよびトランスポートサービス プロバイダの間の相互動作。

これらの手続きは、MCSをサポートし、開放型システム環境で相互接続するシステム間での 多地点通信に適用できる。

3．定義

（注）これらの定義は、 弟4章で定義された省略形を使用する。

この仕様は、ITU-T 勧告 X.200 で開発された概念に基づいており、その中に定義された以下の語を利用する。

a. フロー制御(flow control)
b. 再組み立て（reassembling）
c. recombining
d. segmenting
e. 順序制御
f. splitting
g. transfer syntax
h. トランスポート コネクション(transport connection)
i. トランスポート コネクション終端識別子(transport connection endpoint identifier)
j. トランスポートサービス(transport service)
k. トランスポートサービス アクセス ポイント(transport service access point)
l. トランスポートサービス アクセス ポイント アドレス(transport service access point address)
m. トランスポートサービス データ ユニット(transport service data unit)

この仕様は、TTC標準JT-T122 で開発された概念に基づき、そこに定義された以下の用語を使用する。

a. Control MCSAP(Control MCSAP)
b. MCS アタッチメント(MCS attachment)
c. MCS チャネル(MCS channel)
d. MCS コネクション(MCS connection)
e. MCS ドメイン(MCS domain)
f. MCS ドメインセレクター(MCS domain selector)
g. MCS プライベートチャネル(MCS private channel)

h. MCS プライベートチャネル マネージャ (MCS private channel manager)

i. MCS プロバイダ(MCS provider)

（MCSプロバイダは図中もしくは本文中で単にプロバイダと表記することがある）

j. MCS サービス アクセス ポイント(MCS service access point)

k. MCS ユーザ(MCS user)

l. MCS ユーザ ID(MCS user id)

m. トップ MCS プロバイダ(Top MCS provider)

この仕様のために、以下の定義が当てはまる。

３．１ ＭＣＳサービス データ ユニット

中身が送信側から受信側への転送で維持される MCS ユーザ データ全体。１つの MCS-SEND-DATA-要求または１つの MCS-UNIFORM-SEND-DATA-要求の中身（コンテント）。

３．２ ＭＣＳインタフェース データ ユニット

１つの相互動作中の MCS ユーザと MCS プロバイダ間の MCSAP を経由して転送された情報の単位。各 MCS インタフェース データ ユニットは、インタフェース制御情報を含み、また、MCS サービス データ ユニットのすべてまたは一部を含むかもしれない。

３．３ ＭＣＳプロトコル データ ユニット

MCSプロトコルで交換される1つの情報単位で、MCSプロバイダ間を転送される制御情報から成っている。このMCSプロバイダは、MCSプロバイダ間の結合処理と、サービスを提供するMCSユーザのために転送されるデータを調整する。

３．４ ＭＣＳデータ転送プライオリティ

４つのレベルの１つ:top, high, medium, low。その値（1=top, 2=high, 3=medium, 4=low ）は、送信側から受信側まで不変である。MCS ドメインパラメータで示される実装されているデータ転送プライオリティの数によって、２またはそれ以下のプライオリティは、同じ品質のサービスになる場合がある。

３．５ 有効なＭＣＳＰＤＵ

この仕様に応じた構造、コード化を持つ MCSPDU。

３．６ 無効なＭＣＳＰＤＵ

有効な MCSPDU でない MCSPDU。

３．７ プロトコル エラー

この仕様の手続きと一致していない方法での MCSPDU の使用。

３．８ Ｃｏｎｎｅｃｔ ＭＣＳＰＤＵ

Connect-Initial, Connect-Response, Connect-Additional, Connect-Result の１つ。

３．９ Ｄｏｍａｉｎ ＭＣＳＰＤＵ

Connect MCSPDU でない MCSPDU。

３．１０ Ｄａｔａ ＭＣＳＰＤＵ

SDrq, SDin, USrq, USin の１つ。

3．11　Control　MCSPDU

Data MCSPDU でない Domain MCSPDU。

3．12　初期ＴＣ（initial TC）

Control MCSPDU および最大プライオリティのData MCSPDU を交換するために使われる MCS コネクションにとって最初のトランスポートコネクション。

3．13　付加ＴＣ（additional TC）

プライオリティの低い交換データ用 MCSPDUに使われる、ある MCS コネクションに属するトランスポートコネクション。

3．14　ＭＣＳプロバイダの サブツリー

MCS ドメイン下におけるMCSプロバイダ自身とMCSアタッチメントおよびその配下のMCSプロバイダ。

3．15　ＭＣＳプロバイダの高さ

MCS ドメイン 下において、MCS プロバイダ階層の高さを示す。最下位の MCS プロバイダが、高さ1である。

4．省略形

MCS :　多地点通信サービス(Multipoint Communication Service)
MCSAP : MCS サービス アクセス ポイント(MCS service access point)
MCSPDU : MCS プロトコル データ ユニット(MCS protocol data unit)
TC : トランスポートコネクション(Transport connection)
TS : トランスポートサービス(Transport service)
TSAP : トランスポートサービス アクセス ポイント(Transport service access point)
TSDU : トランスポートサービス データ ユニット(Transport service data unit)

5．ＭＣＳプロトコルの概要

5．1　ＭＣＳ層のモデル

MCS プロバイダは、TTC標準 JT-T122 の中で定義された MCS プリミティブにより MCSAP 経由で MCS ユーザと通信する。これらのプリミティブは、1つの MCS コネクションを使用している相手の MCS プロバイダ間の MCSPDU交換の原因か結果であるか、1つの MCS プロバイダ内でとられた動作の原因か結果である。MCSPDU 交換は、同じ MCS ドメインをつかさどる MCS プロバイダ間に起こる。

1つの MCS プロバイダは、複数の相手MCSプロバイダを持つことが出来る。それは異なるMCSコネクションによる直接的方法および相手MCSプロバイダ経由の間接的方法による。1つの MCS コネクションは、MCS ドメインの中で実装されたデータ転送プライオリティの数に依存して、1つ以上のトランスポート コネクションから成る。プロトコル交換は、一組のTSAP を通してトランスポート層のサービスを使用して、実行される。MCS 層のモデルを、図 5-1/JT-T125 に示す。

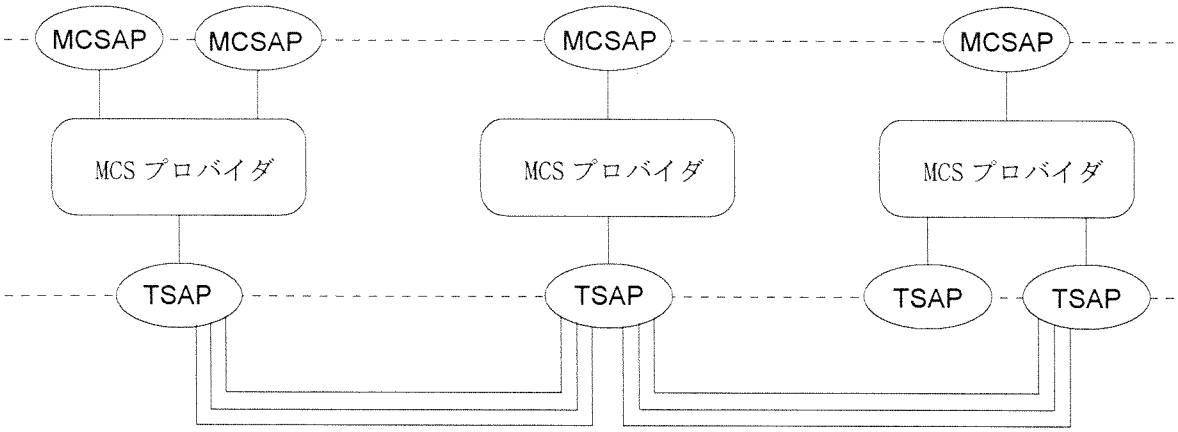


図 5-1 /JT-T125　MCS 層のモデル
(ITU-T T.125)

5．2　ＭＣＳ層によって提供されるサービス

MCS プロトコルは、TTC標準JT- T122 で定義されたサービスをサポートする。表 5-1/JT-T125 に記載された MCS プリミティブを使用してMCS ユーザに対して情報が送受信される。

表 5-1/JT-T125　MCS プリミティブ
(ITU-T T.125)

機能単位	プリミティブ	関連するMCSPDU
ドメイン管理	MCS-CONNECT-PROVIDER-要求	Connect-Initial
	MCS-CONNECT-PROVIDER-指示	Connect-Initial
	MCS-CONNECT-PROVIDER-応答	Connect-Response
	MCS-CONNECT-PROVIDER-確認	Connect-Response
	(副効果)	Connect-Additional Connect-Result PDin EDrq MCrq MCcf PCin MTrq MTcf PTin
	MCS-DISCONNECT-PROVIDER-要求	DPum
	MCS-DISCONNECT-PROVIDER-指示	DPum RJum
	MCS-ATTACH-USER-要求	AUrq
	MCS-ATTACH-USER-確認	AUcf
	MCS-DETACH-USER-要求	DUrq
	MCS-DETACH-USER-指示	DUin

		MCcf PCin MTcf PTin
チャネル管理	MCS-CHANNEL-JOIN-要求	CJrq
	MCS-CHANNEL-JOIN-確認	CJcf
	MCS-CHANNEL-LEAVE-要求	CLrq
	MCS-CHANNEL-LEAVE-指示	MCcf PCin
	MCS-CHANNEL-CONVENE-要求	CCrq
	MCS-CHANNEL-CONVENE-確認	CCcf
	MCS-CHANNEL-DISBAND-要求	CDrq
	MCS-CHANNEL-DISBAND-指示	MCcf PCin
	MCS-CHANNEL-ADMIT-要求	CArq
	MCS-CHANNEL-ADMIT-指示	CAin
	MCS-CHANNEL-EXPEL-要求	CErq
	MCS-CHANNEL-EXPEL-指示	CEin CDin MCcf PCin
データ転送	MCS-SEND-DATA-要求	SDrq
	MCS-SEND-DATA-指示	SDin
	MCS-UNIFORM-SEND-DATA-要求	USrq
	MCS-UNIFORM-SEND-DATA-指示	USin
トークン管理	MCS-TOKEN-GRAB-要求　request	TGrq
	MCS-TOKEN-GRAB-確認	TGcf
	MCS-TOKEN-INHIBIT-要求	TIrq
	MCS-TOKEN-INHIBIT-確認	TIcf
	MCS-TOKEN-GIVE-要求	TVrq
	MCS-TOKEN-GIVE指示	TVin
	MCS-TOKEN-GIVE-応答	TVrs
	MCS-TOKEN-GIVE-確認	TVcf

	MCS-TOKEN-PLEASE-要求 MCS-TOKEN-PLEASE-指示	TPrq TPin
	MCS-TOKEN-RELEASE-要求 MCS-TOKEN-RELEASE-確認	TRrq TRcf
	MCS-TOKEN-TEST-要求 MCS-TOKEN-TEST-確認	TTrq TTcf

5．3　トランスポート層の利用

MCS プロトコルは、CCITT 勧告 X.214 で定義されたコネクションオリエンテッドのトランスポートサービスのサブセットの使用を想定している。情報は、表 5-2/JT-T125 に示されたプリミティブを使用して、TS プロバイダに対して送受信される。

表 5-2/JT-T125 トランスポート サービス プリミティブ
(ITU-T T.125)

プリミティブ	利用	パラメータ	利用
T-CONNECT-要求 T-CONNECT-指示	X X	着アドレス 発アドレス 優先データ オプション サービス品質 TS ユーザ データ	X X － X －
T-CONNECT-応答 T-CONNECT-確認	X X	応答アドレス 優先データ オプション サービス品質 TS ユーザ データ	－ － X －
T-DATA-要求 T-DATA-指示	X X	TS ユーザ データ	X
T-EXPEDITED-DATA-要求 T-EXPEDITED-DATA-指示	－ －	TS ユーザ データ	－
T-DISCONNECT-要求	X	TS ユーザ データ	－
T-DISCONNECT-指示	X	理由 TS ユーザ データ	－ －

X:　MCS プロトコルは、この機能が常に利用可能であると仮定する
－:　MCS プロトコルは、この機能を使用しない

５．４　ＭＣＳ層の機能

表５-１/JT-T125はＭＣＳプリミティブとそれに関連したＭＣＳＰＤＵを示している。ＭＣＳＰＤＵは第７章に定義されている。プリミティブとＭＣＳＰＤＵは、それぞれ互いに原因と結果の関係にある。例えば、MCS-ATTACH-USER-要求 はＡＵｒｑを発生させ、ＡＵｃｆは MCS-ATTACH-USER-確認 を発生させる。

他の場合はさらに複雑である。例えば、MCS-CONNECT-PROVIDER発行後、４フェーズ プリミティブの副効果として、付加的なＭＣＳＰＤＵの送受信を要求する。

MCS-DETACH-USER-指示は５種類のＭＣＳＰＤＵ（DUin,MCcf,PCin,MTcf,PTin）のいずれかを受信すると発行され、MCS-CHANNEL-EXPEL-指示(indication)は４種類のＭＣＳＰＤＵ（CEin,CDin,MCcf,PCin）のいずれかを受信すると発行される。

５．４．１　ドメイン管理

ＭＣＳ層は１つのＭＣＳドメイン中のＭＣＳコネクションの完全性を維持する。１つのＭＣＳコネクションは上位の階層の終端に向けて接続される。それぞれのドメインのトップには１つのトップＭＣＳプロバイダが存在する。１つのＭＣＳコネクションの確立は、２つのドメインを１つに統合する。ＭＣＳ層は１つのトップＭＣＳプロバイダが存続することを保証する。それは、識別や、または排他的な所有権の競合を解決する。

１つのＭＣＳコネクションを切断すると、ドメインは２つに分割される。トッププロバイダを含むドメインが生き残り、一方（ボトム側）は消滅する。

ＭＣＳ層はドメインにアタッチしたユーザを一意に認識する。ユーザはＭＣＳプリミティブによるインタラクションによってお互いを認知する。１つのユーザがデタッチされると、ＭＣＳ層はドメイン中の全てのユーザへその旨を通知する。そして、ＭＣＳ層はデタッチされたユーザのリソースを取り戻す。

５．４．２　チャネル管理

ＭＣＳ層は、データを受信すべき宛て先に送付するために、どの部分のドメインが、ある１つのチャネルに一人以上のユーザが加入しているかを記録する。

ＭＣＳ層は選定されたユーザのみが加入できる単一メンバチャネルとしてユーザＩＤを扱う。

要求により、許可されたユーザがアクセスできるプライベートチャネルを形成し、また、現在、誰も加入していないパブリックチャネルを割り当てる。

５．４．３　データ転送

ＭＣＳ層はチャネルに加入しているユーザに対して順序づけられたデータフローを維持する。一つのチャネルは事実上、宛て先無しの場合から全員への同報の場合の範囲でのマルチキャスト分配リストとなる。

デフォルトとして、ＭＣＳ層はＭＣＳコネクションの最も短いパスでデータを伝送する。オプションとして、特定のＭＣＳサービス データ ユニットをトップＭＣＳプロバイダを通して伝送する。これにより、送信者を含む全ての受信ユーザまでデータの順序性が保たれることを保証する。

ＭＣＳ層は１つ、または複数のデータ転送のプライオリティを認識し、優先処理を実行する。

分割と組立を使用すると、ＭＣＳサービスデータ単位のデータ長を無制限にすることができる。

ＭＣＳ層はドメイン内で全体的なデータ転送フローを調整する。決められた速度でデータを受信できない端末があると、バックプレッシャを発生させ、それが送信端末がブロックされる原因となる。最小の受信速度を維持できないユーザはデタッチされる。

送信側と宛て先側ユーザがアタッチしており、宛て先側ユーザがチャネルに参加しているかぎり、MCS層は送信されたデータのエラーの無い受信を保証する。

しかしながら、優先度の高いデータが先行し、それが多すぎると、優先度の低いデータ伝送が無制限に

遅れる。

５．４．４　トークン管理

ＭＣＳ層ではトップＭＣＳプロバイダにトークンオペレーションを実装し、一貫性と排他性を保証する。

５．５　階層的な処理

図5-2/JT-125はMCSドメインにおける階層的な処理を説明している。

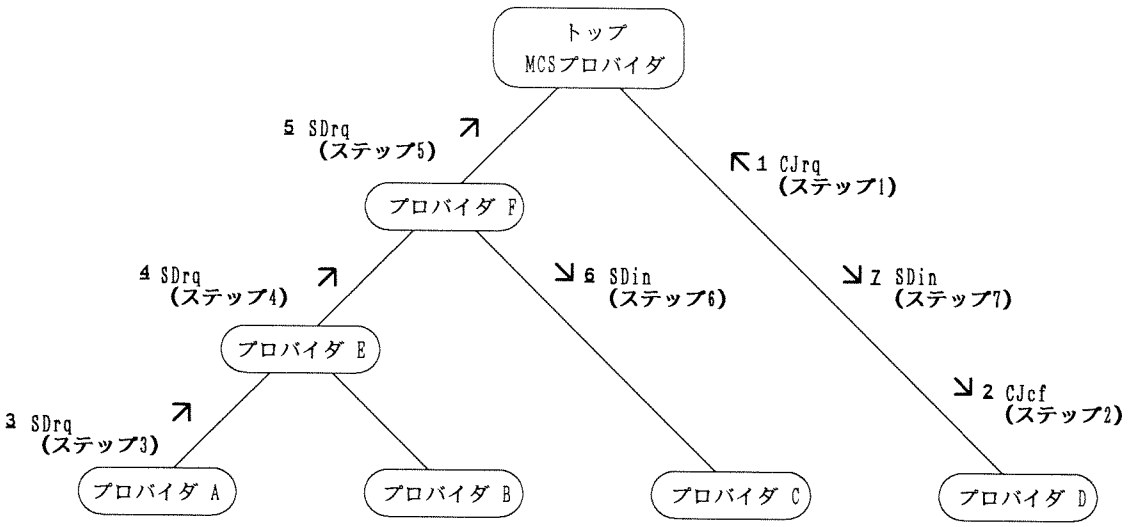


図 5-2/JT-T125　ＭＣＳドメインにおける階層的処理
(ITU-T T.125)

図でノードはＭＣＳプロバイダを表している。またラベル付きの矢印はＭＣＳＰＤＵを表している。

この図は、ＭＣＳプロバイダの接続によりドメインが形成され、データ転送が開始したところを示している。

ステップ１として、プロバイダＤは（ユーザの代理として）チャネルへの加入要求（ＣＪｒｑ）を送信する。ステップ２では、その要求に対する確認（ＣＪｃｆ）を受信する。ステップ３として、プロバイダＡにアタッチしているユーザからのデータ送信要求により対応するＳＤｒｑを送信する。ここで、プロバイダＡ，Ｃ，Ｄがデータ転送チャネルに加入していると仮定すると、ＭＣＳＰＤＵはプロバイダＣとＤへ送られる。このＭＣＳＰＤＵはＳＤｉｎであり、ステップは６と７である。プロバイダＥは下位が受信する必要がなければ、受信したＳＤｒｑを単にプロバイダＦへ送信する（ステップ４）。プロバイダＦは上位のプロバイダへＳＤｒｑを送信する（ステップ５）とともに、下位のプロバイダへＳＤｉｎを送信する（ステップ６）。ここで、プロバイダＦはプロバイダＣがチャネルに加入していることを認知している。

ＭＣＳプロバイダは階層構造において自分がどの階層に位置しているかを知る必要がないが、少なくともドメインの階層（高さ）における制限を維持するための自己の役割や、自分がトップＭＣＳプロバイダ

か否かといった広い意味での役割は知っておく必要がある。トップＭＣＳプロバイダはそれより上方へのコネクションはもたない。ほかのすべてのプロバイダは１つだけ上方へのコネクションを持つ。

ＭＣＳプロバイダはドメインのサブツリー内で使用されているチャネルとトークンについて、以下の情報を記録する。

・サブツリー内に存在しているチャネルと、それぞれのチャネルに加入しているユーザの位置
・下位層のＭＣＳプロバイダとの接続
・サブツリー内に割り当てられたユーザＩＤとそのユーザの位置
・サブツリー内での管理者または許容されたユーザが加入しているプライベートチャネルと、その関連したユーザＩＤ
・サブツリー内のユーザが所有しているトークン、または所有を抑制されたトークンと、その関連したユーザＩＤ

ＭＣＳプロバイダはユーザＩＤが、本来のアタッチメントもしくは下位層へのＭＣＳコネクションに合法的にアサインされているかを確かめるためにサブツリーからの要求を調べる。これにより、トップＭＣＳプロバイダを不正なユーザから保護する。

幾つかの例外はあるが、ＭＣＳ層の一般的な処理を以下に示す。

a. MCSアタッチメントにおけるＭＣＳプロバイダはユーザからのＭＣＳプリミティブによりＭＣＳＰＤＵを生成し、それをトップＭＣＳプロバイダへ向けて送信する。ＭＣＳドメインについてのすべての情報が得られたところでＭＣＳＰＤＵは実行される。

b.トップＭＣＳプロバイダは、要求を出したアタッチメントへ結果を返すために、確認用のＭＣＳＰＤＵを生成、送信する。それを通過させたＭＣＳプロバイダは、サブツリー上のオペレーションに従って、その記録を更新する。確認 は経路にあるＭＣＳプロバイダの記録を調べることにより、要求を出したユーザへ転送される。

c.指示用のＭＣＳＰＤＵは、要求を他のユーザに通知するために生成される。このＭＣＳＰＤＵは、宛先の端末のある下位層へのコネクションがあると、そこで折り返される。ＭＣＳプロバイダは処理の記録を更新する。

上記記載は概念的フレームワークの概要である。詳細は後の章でＭＣＳプロバイダで記録される情報に関してと、どのようにＭＣＳＰＤＵが処理されるかについて述べられる。

上記の内容に対し、以下の例外がある。

・いくつかの要求、特にＣＪｒｑとＣＬｒｑはトップＭＣＳプロバイダまで達するとは限らない。
・いくつかの指示、特にＳＤｉｎはトップＭＣＳプロバイダより下位の層でも生成する。
・ＴＶｒｓは、応答のカテゴリに属する。
・ＭＣＳプロバイダはＤＵｉｎに対するＣＬｒｑ等、異なった種類のＭＣＳＰＤＵを生成する場合がある。

5．6　ドメインパラメータ

ドメイン内のＭＣＳプロバイダは、リソースの割り当てを行い、以下のパラメータに従って動作する。パラメータの値は、ドメイン内において同一である。

a.同時に使用できるＭＣＳチャネルの最大数

これは、ユーザが加入しているチャネル、割り当てられたユーザＩＤ、生成されたプライベートチャネルを含んでいる。

b.同時に割り当てることができるユーザＩＤの最大数

これは、上記のパラメータにより制限される。

c.同時に獲得できる、または所有を抑制できるトークンＩＤの最大数

d.実装されるデータ転送のプライオリティの数

これは、ＭＣＳコネクションにおけるトランスポートコネクションの数に等しい。

ＭＣＳユーザは制限を越えたプライオリティのデータを送受信するかもしれないが、この場合、これらのデータは最も低いプライオリティとして扱われる。

e.スループットの強制

ドメイン内のデータ転送速度は、最も遅い受信端末の速度に制限されるが、受信端末はかってに動作を遅くしてはならない。さもないと、会議に参加した１ユーザが他の全てのユーザを妨害することになる。ＭＣＳプロバイダは、このパラメータを用いて各ＭＣＳアタッチメントと各下位層のＭＣＳコネクションにおける受信速度を最小とする。違反をおかしている端末はデタッチもしくは切断される。

f.階層の高さ

このパラメータは、全てのＭＣＳプロバイダ、特にトップＭＣＳプロバイダの階層の高さを決める。

g.Domain ＭＣＳＰＤＵの最大サイズ

全体的なフロー制御は、ＭＣＳプロバイダ内のDomain ＭＣＳＰＤＵ用（Control ＭＣＳＰＤＵ用ではなく）のバッファの大きさに依存する。簡単にするため、バッファの大きさは固定とすると、ＭＣＳプロバイダはこれより大きいサイズのＭＣＳＰＤＵを生成してはならない。これにより、１つのControl ＭＣＳＰＤＵに入る情報量が制限され、これより大きいユーザデータは分割してData ＭＣＳＰＤＵに入れる必要がある。

h.プロトコルバージョン

これは、Domain ＭＣＳＰＤＵの異なったコード化のいずれかをとる。

（注）ＭＣＳプロバイダのローカルなリソース、すなわち、バッファ用のメモリの大きさ、そのプロバイダに接続する端末の最大数、他のプロバイダとのＭＣＳコネクションの最大数等はローカルな問題である。ＭＣＳドメイン間をまたいでは通過されない。

6．トランスポート サービスの使用

6．1　トランスポート サービスのモデル

この記述は、優先データを利用しないと仮定している ITU-T 勧告 X.214 の関連した部分を言い替える。

トランスポート サービスは、これらの機能を TS ユーザに対して提供する。

a. TSDU を交換するために他の TS ユーザとの間に TC を確立する手段。1つ以上の TC が、同じペアの TS ユーザ間に存在するかもしれない。

b. それぞれの TC 確立時、要求およびネゴシエーションが行われ、スループット、転送遅延、誤り率、プライオリティのような特徴を表わすパラメータによって指定されるサービス品質が、TS プロバイダによって同意される。

c. TC 上で TSDU を転送する手段。オクテット単位で構成される TSDU の転送は、TSDU の境界および TSDU の内容が、TS プロバイダによって不変に維持されるために、透過的である。

d. 送信 TS ユーザが、データを送信する速度を、受信 TS ユーザが制御する手段。

e. 無条件で、それ故に破壊的ともなりうる TC の解放。

TC の動作は、2つの TSAP を連結しているキューのペアによって抽象的にモデル化される。情報の向き毎に1つのキューが存在する。各々の TC は、別々のキューのペアによってモデル化される。

キュー モデルは、フロー制御機能を表すために使われる。キューは限られた容量を持つ、しかし、この容量は必ずしも固定されたものまたは確定できるものではない。接続、 TSDU、切断のオブジェクトは、2つの TSAP での相互動作の結果としてキューから受け入れられたり、削除されたりする。オブジェクトをキューに加える TS ユーザの能力は、オブジェクトをそのキューから削除する TS ユーザの動作とキューの状態によって決定される。TS プロバイダによってキューに付加できる唯一のオブジェクトは切断オブジェクトである。オブジェクトは、TS プロバイダによる制御に従ってキューに加えられる。オブジェクトは、通常、受信 TS ユーザによる制御に従ってキューから削除される。オブジェクトは、通常、それらが加えられたのと同じ順序で削除される。通常の除去に対する唯一の例外は、続くオブジェクトが切断オブジェクトの場合、オブジェクトが TS プロバイダによって削除されることである。

TS ユーザおよび TS プロバイダが TSAP でいくつかの TC を識別する必要があるならば TC 終端識別機構が、ローカルに提供されなければならない。すべてのプリミティブは、それらが当てはまる TC を識別するためにこの識別機構を利用しなければならない。この暗黙の識別は、TS プリミティブのパラメータとして示されず、そして T-CONNECT のアドレス パラメータと混同してはならない。

6．2 複数のコネクションの使用

1つの MCS コネクションは、MCS プロバイダの同じペアの間の1つ以上の TC から成る。確立された最初の TC は、 初期TC (initial TC) と呼ばれる。その後確立したものは、 付加TC (additional TC) と呼ばれる。1つの MCS-CONNECT-PROVIDER -要求 に対する反応において、1つの MCS コネクションに属しているすべての TC が、同じ MCS プロバイダによって確立される。この 要求 は、発着TSAP アドレスと呼ぶアドレス パラメータを含む。これらは、結果として T-CONNECT-要求 の中で修正されずに使用される。

MCS コネクション当りの TC の数は、1つの MCS ドメインの中では一定である。このドメイン パラメータは、実装されたデータ転送プライオリティ レベルの数に等しい。個々の TC が、各々のフロー制御を実現するために必要である。低いプライオリティ データによる障害が、高いプライオリティ データに対して悪影響を与えてはならない。フル実装時には、低いプライオリティ データと高いプライオリティ データは、異なる TC で運ばれなければならない。

TC のために要求されたサービスの品質は、確立されるデータ プライオリティに基づいて変化する。これらのサービス品質目標は、１つの MCS ドメイン内で一定である必要はない。

重要なサービス品質項目として、最大または平均のスループットおよび転送遅延がある。高いプライオリティ データは、リアルタイム応答のために低い転送遅延を要求するかもしれないが、高いスループットを必要としないかもしれない。一方、低いプライオリティ データは、バルク転送のために高いスループットを要求するかもしれないが、低い転送遅延を必要としないかもしれない。

TCプライオリティは、サービス品質のもう１つの項目である。しかしそれは、厳密には MCS データ転送プライオリティの概念とは一致しない。必要に応じて、TC が下位のサービス品質を持つ、関連するオーダーを指定する。MCS プライオリティ データが優先処理を受けることを保証するために、高い TC プライオリティが、低い転送遅延のような他の特徴とともに要求されるかもしれない。

Connect MCSPDU は、 TC のそれぞれの方向で運ばれる最初の TSDU としてのみ発生する。Connect-Initial および Connect-Response は１つの MCS コネクションの 初期TC (initial TC) を経由する。もしあれば、Connect-Additional および Connect-Result は付加TC (additional TC) を経由する。

T-CONNECT-要求 を出す発側 MCS プロバイダは、 TC が、同じ MCS コネクションの一部となるよう、また、示されたデータ転送プライオリティが何であるかによって自身の動作を制御する。

着側 MCS プロバイダは、T-CONNECT-指示 を受けて、一般に TC を受け入れなければならなず、そしてその意味を知る前にその最初の TSDU を読む。Connect-Initial は、受け入れる TC を新しい MCS コネクションの始めとして確認する。Connect-Additional は、受け入れるTCを現在設定しつつある MCS コネクションの一部として確認する。

Connect-Additional は、着側 MCS プロバイダによって設定されて、 初期TC (initial TC) 上の Connect-Response の中で発側 MCS プロバイダへ運ばれた値を含む。そしてその内容が、同じ MCS コネクションに属すように、 付加TC (additional TC) を指名する。Connect-Additional は、また、 TC が表わすデータ プライオリティを明らかにする。

Connect MCSPDU は、 TC 確立に続いて、直ちに交換される。いったん MCS コネクションが確立したならば、それは、階層的な MCS ドメインの一部になる。それ以来、MCS コネクションは、Domain MCSPDU を運ぶ。

Data MCSPDU を除いて、Domain MCSPDU は、１つの MCS コネクション内の 初期TC (initial TC) を経由する。Data MCSPDU は、そのデータ プライオリティに対応した TC を経由する。指定されたプライオリティがMCS ドメインに実装された番号の範囲外にあるならば、Data MCSPDU は、実装されるもっとも低いプライオリティの TC を経由する。

もし１つのプライオリティだけが、１つの MCS ドメインの中で実装されたなら、その MCS コネクションは、各々、１つの TC から成り、Connect-Additional または Connect-Result は利用されない。そして、すべての MCSPDU は、プロバイダの間に順序通りに転送される。

６．３　トランスポート コネクション解放

TS プロバイダは、下位のネットワーク サービスにおける欠点を補うのに十分なプロトコルを実行することによって、信頼性をエンド・エンド コネクションに加えるものである。１つの MCS プロバイダは、この機能を二重には持たない。つまり、転送失敗による自動回復は試みない。

回復不可能な誤りは、T-DISCONNECT-指示 を通して通知される。もし、1つの MCS コネクションに属していている TC のどれかが切断されたなら、他も直ちに切断される。これが、ユーザによって要求された場合を除き、1つの MCS-DISCONNECT-PROVIDER-指示 は、生成され、「プロバイダ発信（provider-initiated）」という理由が与えられる。

一方、1つの MCS-DISCONNECT-PROVIDER-要求 は、「ユーザ要求(user-requested）」として与えられたという理由を持ち、1つの指示 として相手側に伝えられなければならない。ITU-T勧告X.214 における保証にもかかわらず、トランスポート プロトコルのもっとも単純なクラス（クラス0）が、T-DISCONNECT の中でユーザ データの通過を許さない。それゆえに、切断理由コードが、 MCSPDU で転送される。この MCSPDU は、それを受信したMCS プロバイダに対して、それを運んだ MCS コネクションを切断することを要請する。

7．ＭＣＳＰＤＵの構造

ＭＣＳＰＤＵの構造はITU-T勧告X.208のASN.1を使用して定義される。ＭＣＳＰＤＵの使用法と意味は第９章と第１０章に記述している。

```
MCS-PROTOCOL DEFINITIONS ::=
BEGIN

-- Part 1:  Fundamental MCS types

ChannelId       ::= INTEGER (0..65535)          -- range is 16 bits

StaticChannelId ::= ChannelId (1..1000)          -- those known permanently

DynamicChannelId::= ChannelId (1001..65535)      -- those created and deleted

UserId          ::= DynamicChannelId            -- created by Attach-User
                                                -- deleted by Detach-User

PrivateChannelId::= DynamicChannelId            -- created by Channel-Convene
                                                -- deleted by Channel-Disband

AssignedChannelId::=DynamicChannelId            -- created by Channel-Join zero
                                                -- deleted by last Channel-Leave

TokenId         ::= INTEGER (1..65535)          -- all are known permanently

TokenStatus     ::= ENUMERATED
{
                    notInUse            (0),
                    selfGrabbed         (1),
                    otherGrabbed        (2),
                    selfInhibited       (3),
                    otherInhibited      (4),
                    selfRecipient       (5),
                    selfGiving          (6),
                    otherGiving         (7)
}

DataPriority    ::= ENUMERATED
{
                    top                 (0),
                    high                (1),
                    medium              (2),
```

```
                    low                     (3)
}

Segmentation    ::= BIT STRING
{
                    begin                   (0),
                    end                     (1)
} (SIZE (2))

DomainParameters::= SEQUENCE
{
        maxChannelIds   INTEGER (0..MAX),
                    -- a limit on channel ids in use,
                    -- static + user id + private + assigned
        maxUserIds      INTEGER (0..MAX),
                    -- a sublimit on user id channels alone
        maxTokenIds     INTEGER (0..MAX),
                    -- a limit on token ids in use
                    -- grabbed + inhibited + giving + ungivable + given
        numPriorities   INTEGER (0..MAX),
                    -- the number of TCs in an MCS connection
        minThroughput   INTEGER (0..MAX),
                    -- the enforced number of octets per second
        maxHeight       INTEGER (0..MAX),
                    -- a limit on the height of a provider
        maxMCSPDUsize   INTEGER (0..MAX),
                    -- an octet limit on domain MCSPDUs
        protocolVersion INTEGER (0..MAX)
}

-- Part 2:  Connect provider

Connect-Initial         ::= [APPLICATION 101] IMPLICIT SEQUENCE
{
        callingDomainSelector   OCTET STRING,
        calledDomainSelector    OCTET STRING,
        upwardFlag              BOOLEAN,
                            -- TRUE if called provider is higher
        targetParameters        DomainParameters,
        minimumParameters       DomainParameters,
        maximumParameters       DomainParameters,
        userData                OCTET STRING
}

Connect-Response        ::= [APPLICATION 102] IMPLICIT SEQUENCE
{
        result                  Result,
        calledConnectId         INTEGER (0..MAX),
                            -- assigned by the called provider
                            -- to identify additional TCs of
                            -- the same MCS connection
        domainParameters        DomainParameters,
        userData                OCTET STRING
}

Connect-Additional      ::= [APPLICATION 103] IMPLICIT SEQUENCE
{
        calledConnectId         INTEGER (0..MAX),
        dataPriority            DataPriority
}

Connect-Result          ::= [APPLICATION 104] IMPLICIT SEQUENCE
{
```

```
        result                  Result
}

-- Part 3:  Merge domain

PDin    ::= [APPLICATION 0] IMPLICIT SEQUENCE     -- plumb domain indication
{
        heightLimit             INTEGER (0..MAX)
                                -- a restriction on the MCSPDU receiver
}

EDrq    ::= [APPLICATION 1] IMPLICIT SEQUENCE     -- erect domain request
{
        subHeight               INTEGER (0..MAX),
                                -- height in domain of the MCSPDU transmitter
        subInterval             INTEGER (0..MAX)
                                -- its throughput enforcement interval in milliseconds
}

ChannelAttributes ::= CHOICE
{
        static                  [0] IMPLICIT SEQUENCE
        {
                channelId               StaticChannelId
                                        -- joined is implicitly TRUE
        },
        userId                  [1] IMPLICIT SEQUENCE
        {
                joined                  BOOLEAN,
                userId                  UserId
                                        -- TRUE if user is joined to its user id
        },
        private                 [2] IMPLICIT SEQUENCE
        {
                joined                  BOOLEAN,
                                        -- TRUE if channel id is joined below
                channelId               PrivateChannelId,
                manager                 UserId,
                admitted                SET OF UserId
                                        -- may span multiple MCrq
        },
        assigned                [3] IMPLICIT SEQUENCE
        {
                channelId               AssignedChannelId
                                        -- joined is implicitly TRUE
        }
}

MCrq    ::= [APPLICATION 2] IMPLICIT SEQUENCE     -- merge channels request
{
        mergeChannels           SET OF ChannelAttributes,
        purgeChannelIds         SET OF ChannelId
}

MCcf    ::= [APPLICATION 3] IMPLICIT SEQUENCE     -- merge channels confirm
{
        mergeChannels           SET OF ChannelAttributes,
        purgeChannelIds         SET OF ChannelId
}

PCin    ::= [APPLICATION 4] IMPLICIT SEQUENCE     -- purge channels indication
{
```

```
        detachUserIds           SET OF UserId,
                                -- purge user id channels
        purgeChannelIds         SET OF ChannelId
                                -- purge other channels
}

TokenAttributes ::= CHOICE
{
        grabbed                 [0] IMPLICIT SEQUENCE
        {
                tokenId                 TokenId,
                grabber                 UserId
        },
        inhibited               [1] IMPLICIT SEQUENCE
        {
                tokenId                 TokenId,
                inhibitors              SET OF UserId
                                        -- may span multiple MTrq
        },
        giving                  [2] IMPLICIT SEQUENCE
        {
                tokenId                 TokenId,
                grabber                 UserId,
                recipient               UserId
        },
        ungivable               [3] IMPLICIT SEQUENCE
        {
                tokenId                 TokenId,
                grabber                 UserId
                -- recipient has since detached
        },
        given                   [4] IMPLICIT SEQUENCE
        {
                tokenId                 TokenId,
                recipient               UserId
                -- grabber released or detached
        }
}

MTrq    ::= [APPLICATION 5] IMPLICIT SEQUENCE    -- merge tokens request
{
        mergeTokens             SET OF TokenAttributes,
        purgeTokenIds           SET OF TokenId
}

MTcf    ::= [APPLICATION 6] IMPLICIT SEQUENCE    -- merge tokens indication
{
        mergeTokens             SET OF TokenAttributes,
        purgeTokenIds           SET OF TokenId
}

PTin    ::= [APPLICATION 7] IMPLICIT SEQUENCE    -- purge tokens indication
{
        purgeTokenIds           SET OF TokenId
}

-- Part 4:  Disconnect provider

DPum    ::= [APPLICATION 8] IMPLICIT SEQUENCE    -- disconnect provider ultimatum
{
        reason                  Reason
}
```

```
RJum    ::= [APPLICATION 9] IMPLICIT SEQUENCE    -- reject MCSPDU ultimatum
{
        diagnostic              Diagnostic,
        initialOctets           OCTET STRING
}

-- Part 5:  Attach/Detach user

AUrq    ::= [APPLICATION 10] IMPLICIT SEQUENCE   -- attach user request
{
}

AUcf    ::= [APPLICATION 11] IMPLICIT SEQUENCE   -- attach user confirm
{
        result                  Result,
        initiator               UserId OPTIONAL
}

DUrq    ::= [APPLICATION 12] IMPLICIT SEQUENCE   -- detach user request
{
        reason                  Reason,
        userIds                 SET OF UserId
}
DUin    ::= [APPLICATION 13] IMPLICIT SEQUENCE   -- detach user indication
{
        reason                  Reason,
        userIds                 SET OF UserId
}

-- Part 6:  Channel management

CJrq    ::= [APPLICATION 14] IMPLICIT SEQUENCE   -- channel join request
{
        initiator               UserId,
        channelId               ChannelId
                               -- may be zero
}

CJcf    ::= [APPLICATION 15] IMPLICIT SEQUENCE   -- channel join confirm
{
        result                  Result,
        initiator               UserId,
        requested               ChannelId,
                               -- may be zero
        channelId               ChannelId OPTIONAL
}

CLrq    ::= [APPLICATION 16] IMPLICIT SEQUENCE   -- channel leave request
{
        channelIds              SET OF ChannelId
}

CCrq    ::= [APPLICATION 17] IMPLICIT SEQUENCE   -- channel convene request
{
        initiator               UserId
}

CCcf    ::= [APPLICATION 18] IMPLICIT SEQUENCE   -- channel convene confirm
{
        result                  Result,
        initiator               UserId,
        channelId               PrivateChannelId OPTIONAL
}
```

```
CDrq    ::= [APPLICATION 19] IMPLICIT SEQUENCE   -- channel disband request
{
        initiator             UserId,
        channelId             PrivateChannelId
}

CDin    ::= [APPLICATION 20] IMPLICIT SEQUENCE   -- channel disband indication
{
        channelId             PrivateChannelId
}

CArq    ::= [APPLICATION 21] IMPLICIT SEQUENCE   -- channel admit request
{
        initiator             UserId,
        channelId             PrivateChannelId,
        userIds               SET OF UserId
}

CAin    ::= [APPLICATION 22] IMPLICIT SEQUENCE   -- channel admit indication
{
        initiator             UserId,
        channelId             PrivateChannelId,
        userIds               SET OF UserId
}

CErq    ::= [APPLICATION 23] IMPLICIT SEQUENCE   -- channel expel request
{
        initiator             UserId,
        channelId             PrivateChannelId,
        userIds               SET OF UserId
}

CEin    ::= [APPLICATION 24] IMPLICIT SEQUENCE   -- channel expel indication
{
        channelId             PrivateChannelId,
        userIds               SET OF UserId
}

-- Part 7:  Data transfer

SDrq    ::= [APPLICATION 25] IMPLICIT SEQUENCE   -- send data request
{
        initiator             UserId,
        channelId             ChannelId,
        dataPriority          DataPriority,
        segmentation          Segmentation,
        userData              OCTET STRING
}

SDin    ::= [APPLICATION 26] IMPLICIT SEQUENCE   -- send data indication
{
        initiator             UserId,
        channelId             ChannelId,
        dataPriority          DataPriority,
        segmentation          Segmentation,
        userData              OCTET STRING
}

USrq    ::= [APPLICATION 27] IMPLICIT SEQUENCE   -- uniform send data request
{
        initiator             UserId,
        channelId             ChannelId,
        dataPriority          DataPriority,
```

```
        segmentation            Segmentation,
        userData                OCTET STRING
}

USin    ::= [APPLICATION 28] IMPLICIT SEQUENCE    -- uniform send data indication
{
        initiator               UserId,
        channelId               ChannelId,
        dataPriority            DataPriority,
        segmentation            Segmentation,
        userData                OCTET STRING
}

-- Part 8:  Token management

TGrq    ::= [APPLICATION 29] IMPLICIT SEQUENCE    -- token grab request
{
        initiator               UserId,
        tokenId                 TokenId
}

TGcf    ::= [APPLICATION 30] IMPLICIT SEQUENCE    -- token grab confirm
{
        result                  Result,
        initiator               UserId,
        tokenId                 TokenId,
        tokenStatus             TokenStatus
}

TIrq    ::= [APPLICATION 31] IMPLICIT SEQUENCE    -- token inhibit request
{
        initiator               UserId,
        tokenId                 TokenId
}

TIcf    ::= [APPLICATION 32] IMPLICIT SEQUENCE    -- token inhibit confirm
{
        result                  Result,
        initiator               UserId,
        tokenId                 TokenId,
        tokenStatus             TokenStatus
}

TVrq    ::= [APPLICATION 33] IMPLICIT SEQUENCE    -- token give request
{
        initiator               UserId,
        tokenId                 TokenId,
        recipient               UserId
}

TVin    ::= [APPLICATION 34] IMPLICIT SEQUENCE    -- token give indication
{
        initiator               UserId,
        tokenId                 TokenId,
        recipient               UserId
}

TVrs    ::= [APPLICATION 35] IMPLICIT SEQUENCE    -- token give response
{
        result                  Result,
        recipient               UserId,
        tokenId                 TokenId
}
```

```
TVcf    ::= [APPLICATION 36] IMPLICIT SEQUENCE   -- token give confirm
{
        result                  Result,
        initiator               UserId,
        tokenId                 TokenId,
        tokenStatus             TokenStatus
}

TPrq    ::= [APPLICATION 37] IMPLICIT SEQUENCE   -- token please request
{
        initiator               UserId,
        tokenId                 TokenId
}

TPin    ::= [APPLICATION 38] IMPLICIT SEQUENCE   -- token please indication
{
        initiator               UserId,
        tokenId                 TokenId
}

TRrq    ::= [APPLICATION 39] IMPLICIT SEQUENCE   -- token release request
{
        initiator               UserId,
        tokenId                 TokenId
}

TRcf    ::= [APPLICATION 40] IMPLICIT SEQUENCE   -- token release confirm
{
        result                  Result,
        initiator               UserId,
        tokenId                 TokenId,
        tokenStatus             TokenStatus
}

TTrq    ::= [APPLICATION 41] IMPLICIT SEQUENCE   -- token test request
{
        initiator               UserId,
        tokenId                 TokenId
}

TTcf    ::= [APPLICATION 42] IMPLICIT SEQUENCE   -- token test confirm
{
        initiator               UserId,
        tokenId                 TokenId,
        tokenStatus             TokenStatus
}

-- Part 9:  Status codes

Reason          ::= ENUMERATED          -- in DPum, DUrq, DUin
{
                rn-domain-disconnected          (0),
                rn-provider-initiated           (1),
                rn-token-purged                 (2),
                rn-user-requested               (3),
                rn-channel-purged               (4)
}

Result          ::= ENUMERATED          -- in Connect, response, confirm
{
                rt-successful                   (0),
                rt-domain-merging               (1),
                rt-domain-not-hierarchical      (2),
```

```
                rt-no-such-channel              (3),
                rt-no-such-domain               (4),
                rt-no-such-user                 (5),
                rt-not-admitted                 (6),
                rt-other-user-id                (7),
                rt-parameters-unacceptable      (8),
                rt-token-not-available          (9),
                rt-token-not-possessed          (10),
                rt-too-many-channels            (11),
                rt-too-many-tokens              (12),
                rt-too-many-users               (13),
                rt-unspecified-failure          (14),
                rt-user-rejected                (15)
}

Diagnostic      ::= ENUMERATED          -- in RJum
{
                dc-inconsistent-merge           (0),
                dc-forbidden-PDU-downward       (1),
                dc-forbidden-PDU-upward         (2),
                dc-invalid-BER-encoding         (3),
                dc-invalid-PER-encoding         (4),
                dc-misrouted-user               (5),
                dc-unrequested-confirm          (6),
                dc-wrong-transport-priority     (7),
                dc-channel-id-conflict          (8),
                dc-token-id-conflict            (9),
                dc-not-user-id-channel          (10),
                dc-too-many-channels            (11),
                dc-too-many-tokens              (12),
                dc-too-many-users               (13)
}

-- Part 10:  MCSPDU repertoire

ConnectMCSPDU    ::= CHOICE
{
        connect-initial         Connect-Initial,
        connect-response        Connect-Response,
        connect-additional      Connect-Additional,
        connect-result          Connect-Result
}

DomainMCSPDU     ::= CHOICE
{
        pdin                    PDin,
        edrq                    EDrq,
        mcrq                    MCrq,
        mccf                    MCcf,
        pcin                    PCin,
        mtrq                    MTrq,
        mtcf                    MTcf,
        ptin                    PTin,
        dpum                    DPum,
        rjum                    RJum,
        aurq                    AUrq,
        aucf                    AUcf,
        durq                    DUrq,
        duin                    DUin,
        cjrq                    CJrq,
        cjcf                    CJcf,
```

```
        clrq                CLrq,
        ccrq                CCrq,
        cccf                CCcf,
        cdrq                CDrq,
        cdin                CDin,
        carq                CArq,
        cain                CAin,
        cerq                CErq,
        cein                CEin,
        sdrq                SDrq,
        sdin                SDin,
        usrq                USrq,
        usin                USin,
        tgrq                TGrq,
        tgcf                TGcf,
        tirq                TIrq,
        ticf                TIcf,
        tvrq                TVrq,
        tvin                TVin,
        tvrs                TVrs,
        tvcf                TVcf,
        tprq                TPrq,
        tpin                TPin,
        trrq                TRrq,
        trcf                TRcf,
        ttrq                TTrq,
        ttcf                TTcf
}

END
```

8．ＭＣＳＰＤＵの符号化

各MCSPDUは、１つのTSDUとしてあるMCSコネクションに属するTC上を伝送される。ConnectMCSPDUのサイズには制限がない。ドメインMCSPDUのサイズはMCSドメインのパラメータによって制限される。

標準のASN.1データ値符号化が、等位のMCSプロバイダ間でMCSPDUを伝送するために用いられる。このプロトコルには２つのバージョンが定義され、それらは符号化規則の仕様のみが異なっている。

バージョン１は全てのMCSPDUに対して、ITU-T勧告X.209のベーシック符号化規則(BER)を用いる。

バージョン２はConnectMCSPDUに対してベーシック符号化規則を用い、引き続く全てのDomainMCSPDUに対してはISO/IEC 8825-2のパックド符号化規則(PER)を用いる。特に、BASIC-PARのALIGNED形式がASN.1タイプDomainMCSPDUに適用されるべきである。生成されたビット列は整数オクテットとして運ばれるべきである。このストリングの最初のビットは、最初のオクテットの最上位ビットに一致する。

付録　1　に、バージョン１と２の互いに異なった符号化方式におけるデータ転送MCSPDUの一例を示す。

初期TC上で、Connect-Initial、Connect-Response MCSPDUを交換することによりプロトコルのバー

ジョンがネゴシエーションされる。これら２つのMCSPDUは常にベーシック符号化規則で符号化される。DomainMCSPDU開始以前に交換されるConnect-AdditionalとConnect-Result MCSPDUも同様である。DomainMCSPDUはTC上で送信される２番目のTSDUとして送信される。

プロトコルのバージョン２は、パックド符号化規則がITU-T勧告もしくはISO/IEC国際標準として採用されるまで用いられるべきでない。

注.

1　パックド符号化規則により、MCSPDUヘッダはよりコンパクトになる。

2　BERもPERも、各々の符号化されたMCSPDUの終点を定める十分な情報を含んでいるという意味において自己完結型である。TSDUは不要であり、このプロトコルはオクテット列をTSDUの境界を保存しないで転送する非標準トランスポートサービスの上に実装され得るということが、議論されるかも知れない。しかし、そのようなアプローチは実装時にエラーを受け易い。MCSPDUの境界が失われた場合、回復は困難であろう。

９．ＭＣＳＰＤＵのルーティング

９．１　ＣｏｎｎｅｃｔＭＣＳＰＤＵ

図9-1/JT-T125は、ConnectMCSPDUの交換を示したものである。

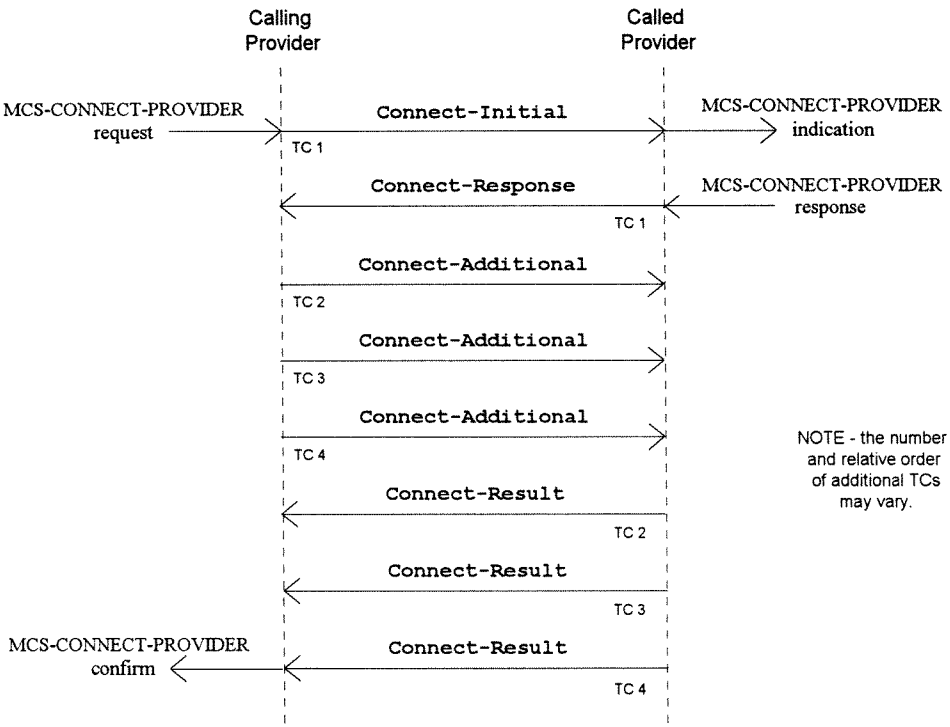


図9-1/JT-T125　ConnectMCSPDUのメッセージフロー

(ITU-T T.125)

Connect-Responseを受信することで、発側MCSプロバイダはドメインにインプリメントされたデータ転送プライオリティの数についてのネゴシエーション結果を知ることができる。図では、付加TC上のコネ

クトMCSPDUは、着側MCSプロバイダで厳密に２、３、４の順番に処理されていることが示されている。実際には、トランスポート コネクションは要求された順番に設定されないかもしれない。Connect-Additionalは、送られた順番と異なって到着するかもしれないし、それはまた異なった順でConnect-Resultの返送される原因にもなる。あるいは後者は、たとえ順番に送られたとしても、列び換えが伝送中に起こるかも知れない。発側MCSプロバイダは最初のConnect-Additionalを送出するまでに、全ての付加 TCが設定されるのを待つ必要はない。着側MCSプロバイダは、全セットのConnect-Additional MCSPDUが、最初のConnect-Resultを返送する以前に到着するのを待つ必要はない。どのような順番であれ、発側MCSプロバイダは全セットの成功結果を受信すると、MCS-CONNECT-PROVIDER-確認を上げる。

MCSPDU交換中において、理由表示に「失敗(unsuccessful)」を持つConnect-Responseや、Connect-Result、もしくはT-DISCONNECT-通知は、それまでそのMCSコネクションに属していた全てのＴＣを切断する要因となり、理由表示に「失敗(unsuccessful)」を持つMCS-CONNECT-PROVIDER-確認を上げる。

MCS-CONNECT-PROVIDER-要求は、２つのMCSプロバイダのどちらが高位であるかを規定する。この階層関係により、その後のドメインMCSPDUのルーティングが決定され、発側と着側のMCSプロバイダの差別はそれ以後無意味となる。例えば、次のステップで、MCSレイヤが元々独立していた２つのドメインのリソースを融合する場合などである。MCrqとMTrqは、下位のMCSプロバイダで生成され、上位のMCSプロバイダ方向へ新しいMCSコネクションを通じて送信される。MCSPDUが送信される方向は、発側から着側もしくは着側から発側への何れかであり、これは上方向フラグがどのようにセットされているかどうかで決まる。

９．２　ＤｏｍａｉｎＭＣＳＰＤＵ

表9-1/JT-T125は、DomainMCSPDUのルーティングを表したものである。

表9-1/JT-T125

(ITU-T T.125)

ドメインMCSPDUのルーティング

カテゴリ	MCSPDU	TC	方向
要求	EDrq　MCrq　MTrq AUrq　DUrq CJrq　CLrq　CCrq CDrq　CArq　CErq	I	Up

	TGrq TIrq TVrq TPrq TRrq TTrq		
	SDrq USrq	A	
指示	PDin PCin PTin DUin CDin CAin CEin TVin TPin	I	Down
	SDin USin	A	
応答	TVrs	I	Up
確認	MCcf MTcf AUcf CJcf CCcf TGcf TIcf TVcf TRcf TTcf	I	Down
通告	DPum RJum	I	Up or Down

I: MCSPDUはイニシャルTCを通過する。

A: MCSPDUはデータのプライオリティに従い付加TCを通過する。

Up: MCSPDUはトップMCSプロバイダに向かう。

Down: MCSPDUはトップMCSプロバイダから離れて行く。

もしMCSプロバイダがカテゴリ「要求」に属するMCSPDUを生成もしくは転送するなら、それはユニークなMCSコネクション上を上位に向かって進む。EDrq, CJrq, CLrqは途中のMCSプロバイダで消滅する。他の要求は、要求の内容が無効でない限り、トップMCSプロバイダにより処理されるために上位方向へと転送されるが、無効な場合にはそのMCSPDUは確認なしに無視される。

もし、MCSプロバイダがカテゴリ「指示」に属するMCSPDUを生成もしくは転送するなら、そのコピーは、おそらく内容を修正されるだろうが、下記の原則に従ってMCSコネクション上を下位方向へ進む。

a. PDinは、下位方向の全てのMCSコネクションに転送される。それを含む上限値は1つ減じられる。上限値として0の値を持つMCSPDUを受信したMCSプロバイダは切断すべきである。

b. PCinは、下位方向の全てのMCSコネクションに転送される。転送されたユーザIDは変化せず、全てのデタッチされたユーザは残されたユーザに通知される。転送された他のチャネルIDは、サブツリーにおいて使用中のチャネルIDに制限されるかも知れない。まだ上位ドメインに併合されている以前のトップ プロバイダでは、ユーザIDと他のチャネルIDの何れも、上位ドメインへの受け入れが確認されているものだけに制限される。

c. PTinは、下位方向の全てのMCSコネクションに転送される。転送されたトークンIDは、サブ ツリーで使用されているIDに制限される。まだ上位ドメインに併合されている以前のトップ プロバイダでは、トークンIDは上位ドメインへの受け入れが確認されているものだけに制限される。

d. DUinは、下位方向の全てのMCSコネクションに転送される。転送されたユーザIDは変化せず、全てのデタッチされたユーザは残されたユーザに通知される。まだ上位ドメインに併合されている以前のトップ プロバイダでは、ユーザIDは上位ドメインへの受け入れが確認されているものだけに制限される。

e. CDinは、サブツリー内のプライベート チャネルのマネージャもしくは許可されているあらゆるユーザを含む下位方向の全てのMCSコネクションに転送される。

f. CAinとCEinは、サブツリー内の1以上の影響を受けるユーザを含む下位方向の全てのMCSコネクションに転送される。転送されるユーザIDはサブツリー内に存在するものだけに制限される。

g. TVinは、サブツリー内の指名された受信者を含む下位方向の1つのMCSコネクションに転送される。

h. TPinは、サブツリー内でトークンを獲得、抑制、または譲渡中のユーザを含む下位方向の全てのMCSコネクションに転送される。

i. SDinとUSinは、SDinが生成されたとき、それが、SDrqが到着するコネクション上を返送されない場合を除いて、指定されたチャネルが結合される下位方向の全てのMCSコネクションに転送される。

指示として用いられる PCin, PTin, DUin, CAin, CEinは、もしIDのセットが空のものを含んでいるなら、転送される必要がない。

もし、MCSプロバイダがカテゴリ「応答」のMCSPDUを生成もしくは転送するなら、それはユニークなMCSコネクション上を上位の方向に進む。それは、内容が無効であると決定されない限り、トップMCSプロバイダにより処理されるために上位方向に転送されていく。

もし、MCSプロバイダがカテゴリ「確認」に属するMCSPDUを生成もしくは転送するなら、次の規則に従って1つのMCSコネクション上を下位方方向へ進む。

a. MCcfは、まだ確認により回答されていない最初のMCrqのパスを反対方向に引き返す。
これは、各々のMCSプロバイダがペンディング状態の要求のファースト イン ファー スト アウトのキューを保持することを要求する。

b. MTcfは、まだ確認により回答されていない最初のMTrqのパスを反対方向に引き返す。これは、各々のMCSプロバイダがペンディング状態の要求を格納するファースト イン ファースト アウトのキ

ューを保持することを要求する。

c. AUcfは、まだ確認により回答されていない初期のAUrqのパスを反対方向に引き返す。AUrqは、1つ以上がペンディングかどうかは問題ではないが、それぞれのMCSプロバイダの正しい動作のためには、ファースト イン ファースト アウトのキューを保持すべきである。AUcfの送信時、MCSプロバイダは、ユーザIDが含まれているどのサブツリーが割り付けられているかを記録すべきである。

d. 他のカテゴリ「確認」に属するMCSPDUは、ちょうど説明したように以前のAUcfの動作を通して割り当てられた要求者のユーザIDを含む。これらのMCSPDUは、ユーザIDが割り付けられたサブツリーにつながるMCSコネクション上を下位方向に転送される。この方法を継続し、最終的にそれらは、要求しているMCS アタッチメントをホストするプロバイダへと戻っていく。

確認は、要求と同様な処理の結果で生成される。CJcfを除く全ては、トップMCSプロバイダにより生成される。

もし、MCS プロバイダがカテゴリ「通告」に属する MCSPDU を生成したなら、それは1つの MCS コネクション上を上下何れかの方向に進む。DPum は、受信側 MCS プロバイダにそれを運んできた MCS コネクションを切断するように命令する。RJum は、診断コードで誤った MCSPDU を拒絶し、それを送信した MCS プロバイダに切断するように促す。DPum と RJum は転送されない。

１０．ＭＣＳＰＤＵの意味付け

表１０－１から１０－４７は第７章で定義された個々のＭＣＳＰＤＵの内容を再掲するものである。

１０．１ Ｃｏｎｎｅｃｔ－Ｉｎｉｔｉａｌ

Ｃｏｎｎｅｃｔ－Ｉｎｉｔｉａｌは、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－要求によって生成される。それは、新しいＭＣＳコネクションの初期ＴＣ上の最初のＴＳＤＵとして送られる。受信側では、それはＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－指示を生成する。

表１０－１/JT-T125　Connect-Initial MCSPDU

(ITU-T T.125)

内　　容	ソース	シンク
発側 ドメイン セレクタ(Calling Domain Selector)	要求	指示
着側 ドメイン セレクタ(Called Domain Selector)	要求	指示
上方向フラグ(Upward Flag)	要求	指示
ターゲット ドメイン パラメータ(Target Domain Parameters)	要求	指示

ドメイン パラメータ 最小値 (Minimum Domain Parameters)	要求	指示
ドメイン パラメータ 最大値 (Maximum Domain Parameters)	要求	指示
ユーザデータ (User Data)	要求	指示

要求＝要求プリミティブ
指示＝指示プリミティブ

発側トランスポート アドレスと着側トランスポート アドレスは、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－要求と指示の付加パラメータである。それらは、Ｔ－ＣＯＮＮＥＣＴのパラメータとなり、どんなＭＣＳＰＤＵ内でも明確に通知されることはない。
このトランスポート アドレスの対は、同じＭＣＳコネクションに属している全てのＴＣにリクエストするために使う。

トランスポートのサービス品質（ＱＯＳ）は、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－要求の付加パラメータであるが、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－指示にはない。ＱＯＳは、あるＴＣと他のＴＣとで異なっていてもよく、有効なサービスは、そのＴＣを確立する過程でのみ明らかにされる。ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲでドメインパラメータの実装データプライオリティの数をネゴシエーションするまで、必要な付加TCの数がわからないことから、このプリミティブはそれらのトランスポートのＱＯＳを、同時に、十分ネゴシエーションすることは出来ない。そのため、発側ＭＣＳプロバイダによって任意に決定され、かつ、各々のＴ－ＣＯＮＮＥＣＴを通して指示される最小値を満たす限り、着側ＭＣＳプロバイダは、着信するＴＣを提案されたＱＯＳで自動的に受け入れる。

ドメインセレクタ値の解釈は、各々のＭＣＳプロバイダによるローカルな問題である。これは、アドレスを表すオクテットストリングである。受け入れられる値は、ＭＣＳプロバイダを設定する過程を通して決定される。１つ以上の値が、同じドメインによって採用されてもよい。不確定なドメインセレクタは、長さ０のオクテットストリングである。これは、ローカルな協定を通して、明確な値に決められる。

上方向フラグは、新しいＭＣＳコネクションの方向を明確に示す。もし、着側プロバイダが発側プロバイダより上位になる場合は真、そうでない場合は偽、である。ＭＣＳプロバイダは、関係するＭＣＳコネクションの方向によりドメインの基本階層の中で、ある役割を果たす。プロバイダは上位のプロバイダに２つのコネクションを許してはならない。上位のプロバイダにコネクションを持たないプロバイダは、トップＭＣＳプロバイダとして動作する。

Ｃｏｎｎｅｃｔ－Ｉｎｉｔｉａｌのターゲット ドメインパラメータは、最小値から最大値までの間で個々に特定された値である。（訳注：１つのプロバイダが複数のＭＣＳコネクションを確立するような場合を想定）ＭＣＳプロバイダは、リクエストされたドメインパラメータを、実装された限界値を示すか、ドメインのメンバーの中で既に合意された値を課するかで、修正する。それは、最小値を増加させ、最大値を減少させる。（修正を）繰り返す間、ターゲットをその範囲内に保つためだけに、それを変更するものとする。ＭＣＳプロバイダは、提案した値の範囲内で返ってくるどんなレスポンスも受け入れるための準備をしなければならない。

ユーザデータは、任意の一連のオクテットである。それは、長さ０であってもよい。

ＭＣＳプロバイダは、着信するＴＣを容量の限界まで、自動的に受け入れる。

Ｔ－ＣＯＮＮＥＣＴ内のユーザデータは使えない。データ転送され最初に受信したＴＳＤＵ（Ｃｏｎｎｅｃｔ－ＩｎｉｔｉａｌかＣｏｎｎｅｃｔ－ＡｄｄｉｔｉｏｎａｌのＭＣＳＰＤＵのどちらか）が、そのＴＣの性質を決める。もしその内容が受け入れられないなら、着側ＭＣＳプロバイダは、ＴＣを直ちに切断してもよい。このような場合には、なぜＭＣＳコネクションが失敗したのか説明するための、Ｃｏｎｎｅｃｔ－ＲｅｓｐｏｎｓｅかＣｏｎｎｅｃｔ－Ｒｅｓｕｌｔを返すことが望ましい。発側ＭＣＳプロバイダは、その後で切断しなければならない。

１０．２　Ｃｏｎｎｅｃｔ－Ｒｅｓｐｏｎｓｅ

Ｃｏｎｎｅｃｔ－Ｒｅｓｐｏｎｓｅは、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－応答により生成される。それは、新しいＭＣＳコネクションの初期ＴＣ上に、逆方向に送られた最初のＴＳＤＵである。それは、発側ＭＣＳプロバイダにＭＣＳコネクションの容認を伝え、それに続けて、発側ＭＣＳプロバイダはいくつかの必要な付加TCを確立する。

表１０－２/JT-T125　Connect-Response MCSPDU

(ITU-T T.125)

内　　容	ソース	シンク
結果 (Result)	応答	確認
ドメイン パラメータ (Domain Parameters)	応答	確認
着側接続ID(Called Connect Id)	着側プロバイダ	発側プロバイダ
ユーザデータ (User Data)	応答	確認

要求＝要求プリミティブ
指示＝指示プリミティブ
応答＝応答プリミティブ
確認＝確認プリミティブ

もし結果が成功なら、このＭＣＳＰＤＵは事実上、ドメインパラメータを決定する。
実装されたＭＣＳデータ転送のプライオリティの数は、ＭＣＳコネクション内のＴＣの数と同じである。もしこれが１より大きいなら、Ｃｏｎｎｅｃｔ－ＡｄｄｉｔｉｏｎａｌやＣｏｎｎｅｃｔ－ＲｅｓｕｌｔのＭＣＳＰＤＵの交換を通して付加TCが確立され、ＭＣＳコネクションに関係付けられる。

着側接続ID(Called Connect Id)は、その着側ＭＣＳプロバイダへの着信付加TCを、この初期ＴＣに関係付ける手段として使われる。その値は、ただこの目的のためだけに利用される。
それは、着側プロバイダにおいて確立中の、１つのＭＣＳコネクションを一意に識別できなければならない。このＩＤは、ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲの完了後は使用されない。

ＭＣＳ－ＣＯＮＮＥＣＴ－ＰＲＯＶＩＤＥＲ－確認のパラメータの大部分は、Ｃｏｎｎｅｃｔ－Ｒｅｓｐｏｎｓｅで伝わる。もし、結果が失敗か、付加TCが必要ないなら、確認が直ちに生成される。そうでな

ければ、ＭＣＳコネクションに付加TCが関係付けられたことを知らされるまで、それを延期する。

１０．３　Ｃｏｎｎｅｃｔ－Ａｄｄｉｔｉｏｎａｌ

Connect-Additional は、Connect-Response の受信に従って生成される。それは、新しい MCS コネクションの付加 TC 上の最初の TSDU にて送られる。

表 10-3/JT-T125　Connect-Additional MCSPDU

(ITU-T T.125)

内　　容	ソース	シンク
着側接続 ID(Called Connect Id)	発側プロバイダ	着側プロバイダ
データプライオリティ(Data Priority)	発側プロバイダ	着側プロバイダ

データプライオリティ(Data Priority)は、要求された付加 TC の数まで、high、medium、low の順の値となる。

１０．４　Ｃｏｎｎｅｃｔ－Ｒｅｓｕｌｔ

Connect-Result は、Connect-Additional の受信に従って生成される。それは、新しい MCS コネクションの付加 TC 上で逆方向に送られた最初の TSDU である。

表 10-4/JT-T125　Connect-Result MCSPDU

(ITU-T T.125)

内　　容	ソース	シンク
結果(Result)	着側プロバイダ	確認

確認＝確認プリミティブ

もし結果が失敗なら、MCS-CONNECT-PROVIDER-確認が直ちに生成される。MCS コネクションと連携した全ての TC は切断され、それらが転送する MCSPDU は無視される。

一方、結果が成功なら、各々の付加 TC の確立を全て確認するのを待って、成功を示す MCS-CONNECT-PROVIDER-確認が生成される。

MCS-CONNECT-PROVIDER-確認の後、ドメイン MCSPDU は MCS コネクションの相手側へ伝わる。各々の MCS コネクションは、一つのドメインに属する。MCS プロバイダが１つ以上のドメインのホストを務める構造では、MCSPDU を運ぶ MCS コネクションは、それらを適用するドメインを決定する。この章の以下の部分で示すドメイン MCSPDU の記述は、一つのドメイン内で設定されることを想定している。

10.5 PDin

PDin は、成功した MCS-CONNECT-PROVIDER の完了に従って生成される。それはサイクル(ループ)が生成されなかったことを保証するために新しいMCSコネクションの下のMCSプロバイダの階層の深さを測る。PDin は、ドメインの最大の高さを強制するためにも、トップMCS プロバイダによって生成される。

表 10-5/JT-T125　PDin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
高さ制限(Height Limit)	以前のトップ またはトップ	下位者

PDin は、ドメインのトップであることをやめたプロバイダによって、新しい MCS コネクションの低位の終端で生成される。その内容は、ドメインの最大の高さのドメインパラメータで初期化される。PDin は、全ての下方の MCS コネクション上に送信される。

必要ならば、PDin はトップ MCS プロバイダにおいて同じように生成される。

どこで PDin が受信されても、それが含む「高さ制限(height limit)」は検査される。もし 0 よりも大きいならば、「高さ制限(height limit)」は 1 減算され、PDin は全ての下方の MCS コネクション上に転送される。それに反して、0 値は受信側がトッププロバイダから非常に遠くにあることを意味し、上方への MCS コネクションを切断することで反応するべきである。これは、サブツリー全体を削除して、ドメインの高さを修復することを助ける。

サイクルの存在中は、PDin の「高さ制限(height limit)」は、0 になるまで減算されるに違いない。これ(0 値)を見つけたプロバイダは、サイクル中のそのドメインから全ての下位層のプロバイダを削除する事を犠牲にしてサイクルを破壊する。

注　MCS コネクションの純粋にローカルな知識では、MCS プロバイダは、サイクルの作成を妨げることは出来ない。常に上位へのコネクションが 1 以内であることを保証することは出来るが、上位へのコネクションが、ある下方のプロバイダへのループバックを作成しないことを保証することは出来ない。サイクルが作成されるとき、直接の原因はトップ MCS プロバイダからの誤った上位へのコネクションがあるためである。MCS コネクションの生成を指定する制御アプリケーションはその様なエラーを避けようと努力しなければならない。

10.6 EDrq

EDrq は、MCS-CONNECT-PROVIDER の成功に続いて生成される。それは、プロバイダの高さとそ

れらのスループットの強制インターバルの変更を上方へ伝達する。

EDrq は、その高さあるいはインターバルを変更するときは、いつでも MCS プロバイダにより生成される。

表 10-6/JT-T125　EDrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
ドメインの高さ (Height in Domain)	下位者	上位プロバイダ
スループット強制インターバル (Throughput Enforcement Interval)	下位者	上位プロバイダ

MCS プロバイダの高さは、MCS コネクションが追加されたり切断されたとき、あるいは下位のプロバイダが EDrq を通して変更を報告したときに変化する。ドメイン パラメータで指定された最小スループットを強制するための監視インターバルは、下位者によって報告されたインターバルや他の理由に適合させることによって変化する。もし、いずれかの値が変化するなら、MCS プロバイダはその直接上位者に EDrq を送信しなければならない。

１０．７　ＭＣｒｑ

MCrq は、MCS-CONNECT-PROVIDER の成功に続いて生成される。MCrq は、マージしたドメインの中に組み込むために、以前のトップ プロバイダが保持していたチャネルの属性を上方へ伝達する。

表 10-7/JT-T125　MCrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
マージ チャネル(Merge Channels)	以前のトップ	トップ プロバイダ
削除チャネル ID(Purge Channel Ids)	中間プロバイダ	トップ プロバイダ

MCrq は、MCSPDU サイズのドメイン制限まで複数のチャネルの属性を含めることができる。第 7 章の ASN.1 の定義で詳細に示すように、使用中の 4 種類のチャネル（スタティック、ユーザ ID、プライベート、割当）は、それぞれ関連する属性を持っている。これらの属性は、トップ MCS プロバイダの情報ベースにおいて保持され、部分的にチャネルが利用されているサブツリーにその複製が置かれる。2 個のドメインが MCS-CONNECT-PROVIDER によってマージされる時には、下位のドメインで使用中のチャネ

ルは、上位のドメインの情報ベースに組み入れられるか、または下位のドメインから削除されなければならない。この決定は、マージした（上位側）ドメインのトップ プロバイダの責任である。

各々のチャネルは、個別に考えなければならない。もし、使用中のチャネルに関するドメイン制限がそれを許容し、すでにチャネル ID が上位側で使用されていないならば、上位のドメインは、そのチャネルを組み込むように拡張しなければならない。スタティック チャネル ID として使用する場合は、衝突することはない。下位のドメインの中でプライベート チャネル ID として使用している場合は、もしそれが上位のドメインの中でもプライベート チャネル ID として使用され、また、マネージャのユーザ ID が同じであるならば衝突とはならない。他の全ての同時使用の組合せは却下され、下位のドメインからチャネル ID を削除することが要求される。

もし、1つのプライベート チャネルの許可されたユーザが非常に多数ならば、その属性は単一の MCrq に納まらないかもしれない。この場合、その属性は複数の MCSPDU により上位へ送信される。しかし、同じプライベート チャネルをマージするための 2 番目以降の要求は、1 番目の要求に対する応答の MCcf を受信するまで待たなければならない。そのときのみ、上位のドメインでチャネルを使用することがドメイン制限に許容されたかどうかが知らされる。もし、最初の要求が失敗したならば、その許可されたユーザについての残りの要求を繰り返してはならない。

各々の MCrq に対しては、同じ順番でトップ MCS プロバイダから MCcf が返される。MCcf は、先行の MCrq を明示的に識別するものは何も含まない。応答は、ただ単にこれらの MCSPDU が受信された順にルーティングされる。以前のトップ MCS プロバイダの上位になった MCS プロバイダは、対応する MCcf が同じ MCS コネクションを経て戻るように、未応答の MCrq の各々と、それがどこから到着したかを記録する。

中間の MCS プロバイダは、プライベート チャネルの属性の中に宣言されたユーザ ID が MCrq を発信したサブツリーに正しく割り当てられていることを保証するために、それらの正当性を検証する。無効なユーザ ID は削除する。もし、プライベート チャネルのマネージャが消去されるなら、チャネルのすべての属性は、マージの要求から消去され、チャネル ID だけを削除されるセットの方に移さなければならない。このユーザ ID の正当性の検証を除いて、中間の MCS プロバイダは、MCrq の内容を修正しない。

以前のトップ プロバイダは、スタティック、割当、またはプライベート チャネルをマージするために属性を提示し始める前に、全てのユーザ ID と全てのトークン ID がマージしたドメイン中に組み込まれるか削除されるのを待つ必要がある。

１０．８　ＭＣｃｆ

MCcfは、前述のMCrqに対する応答である。MCcfは、MCrqと同じセットのチャネルIDおよび属性のサブセットをパラメータとして持つ。マージされたドメインに組み入れられなかったチャネル属性は、除去されるチャネルIDとして報告される。

表１０－８／ＪＴ－Ｔ１２５　MCcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
マージ チャネル(Merge Channels)	トップ プロバイダ	中間プロバイダ
削除チャネルID(Purge Channel Ids)	トップ プロバイダ	以前のトップ

受け入れられたチャネルIDは、トップMCSプロバイダの情報ベースに入力された属性が反映されている。中間のプロバイダは、この情報に一致させるため、自身の情報ベースを更新する。

下位のドメインから除去されるチャネルは、チャネルIDだけ列挙される。もし、上位のドメインにおいて同じチャネルIDが利用されるなら、上位ドメイン側のチャネルIDは除去されずに残る。中間のプロバイダは除去されたチャネルIDをそのまま転送する。

MCSプロバイダは、MCrqを転送したときの記録を利用して、MCcfをMCrqの発信元に向け送信する 。MCcfは、MCrqを生成した以前のトッププロバイダに返送される。そこでは、返答（MCcf）を受信するまでの間、チャネルIDは情報ベース中にあるため、マージされたチャネルは無視してもよい。削除されたチャネルIDは、PCinのパラメータとして引き継がれた後に情報ベースから削除される。

中間のMCSプロバイダは、プライベートチャネル属性内に宣言されたユーザIDが、MCcfが発行されるサブツリーに割り当てられることを確認する。もし、先行するMCrqが有効となってから、プライベートチャネルマネージャーがデタッチされ、どこかに再割当されるならば、プライベートチャネルがドメインマージの障害となる中間のプロバイダははCDrqを発生し、削除されたチャネルIDのセット(Purged Channel Ids)に移動させる。どのような許可されたユーザーがどこかに再割当されていても、チャネルからそれらを排除する。

１０．９　ＰＣｉｎ

PCinは、以前のトッププロバイダにおいてMCcfの受信に従って生成される。PCinは下方へ同報送信され、下位のプロバイダから指定されたチャネルIDを削除する。

表10-9/JT-T125　PCin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
デタッチユーザID (Detach User Ids)	以前のトップ	下位者
削除チャネルID (Purge Channel Ids)	以前のトップ	下位者

チャネルIDの現在の用途に依存して、削除することによる以下の影響がある。

・ユーザIDチャネルの場合は、全てのユーザへMCS-DETACH-USER-指示が発行される。

・スタティック、または割当チャネルIDの場合、加入したユーザへMCS-CHANNEL-LEAVE-指示が発行される。
・プライベートチャネルIDの場合、マネージャへMCS-CHANNEL-DISBAND-指示と許可されたユーザへMCS-CHANNEL-EXPEL-指示が発行される。

下位のドメインの中で全てのチャネルの使用状況を知っている以前のトッププロバイダは、チャネルIDだけから正しい指示を生成することができる。しかし、それより下位のプロバイダは、部分的な知識のみ持つ。それらは、どのチャネルIDが、デタッチユーザを表すか指示が常に生成されるのかを知らされなければならない。また、チャネルの他の種類が下位のプロバイダで使用中である場合には、指示が生成されるために、知らされなければならない。そのため、PCinは、削除されるチャネルIDをこれらの2つのカテゴリーに分ける。

PCinによるユーザIDの削除は、DUinによる削除と同じ結果を持つ。 ただし、PCinの受信側がもはやトッププロバイダではないので、副作用としてCDinまたはTVcfを発生する必要がないことを除く。

注　プロバイダは、加入していないスタティックあるいは割当チャネルID、あるいは、そのアタッチメントがマネージャあるいは許可されたユーザでもないプライベートチャネルIDをPCinの中で受信するかもしれない。情報ベースに保持された記録を使用することによって、プロバイダにそのようなチャネルIDのプリミティブの指示を削除することができる。

10.10 MTrq

MTrqはMCS－CONNECT－PROVIDERの成功に続いて生成される。MTrqは、マージしたドメインの中に組み込むために、以前のトップ プロバイダが保持していたトークンの属性を上方へ伝達する。

表10-10/JT-T125　MTrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
マージ トークン(Merge Tokens)	以前のトップ	トップ プロバイダ
削除トークンID(Purge Token Ids)	中間プロバイダ	トップ プロバイダ

MTrqは、MCSPDUサイズのドメイン制限まで複数のトークンの属性を含めることができる。第7章のASN.1の定義で詳細に示すように、使用中のトークンの個々の状態（「獲得(grabbed)」、「抑制(inhibited)」、「譲渡中(giving)」、「譲渡不可(ungivable)」、「譲渡(given)」）は、それぞれ関連する属性を持っている。これらの属性は、トップMCSプロバイダの情報ベースにおいて保持され、部分的にトークンが利用されているサブツリーにその複製が置かれる。2個のドメインがMCS-CONNECT-PROVIDERによってマー

ジされる時には、下位のドメインで使用中のトークンは、上位のドメインの情報ベースに組み入れられるか、または下位のドメインから削除されなければならない。この決定は、マージした（上位側）ドメインのトップ プロバイダの責任である。

各々のトークンは個別に考えなければならない。もし、使用中のトークンに関するドメイン制限がそれを許容し、すでにトークンIDが上位側で使用されていないならば、上位のドメインはそのトークンを組み込むように拡張しなければならない。上位のドメインにおいてもトークンIDが抑制されるならば、下位のドメインにおけるトークンIDの抑制は衝突とはならない。他の全ての同時使用の組み合わせは却下され、下位のドメインからトークンIDを削除することが要求される。

もし、1つのトークンを抑制しているユーザが非常に多数ならば、その属性は単一のMTrqに納まらないかもしれない。この場合、その属性は複数のMCSPDUにより上位へ送信される。しかし、同じ抑制されたトークンをマージするための2番目以降の要求は、1番目の要求に対する応答のMTcfを受信するまで待たなければならない。そのときのみ、上位のドメインでトークンを使用することがドメイン制限に許容されたかどうかが知らされる。もし、最初の要求が失敗したならば、その抑制者についての残りの要求を繰り返してはならない。

各々のMTrqに対しては、同じ順番でトップMCSプロバイダからMTcf が返される。MTcfは、先行のMTrqを明示的に識別するものは何も含まない。応答は、ただ単にこれらのMCSPDUが受信された順にルーティングされる。以前のトップMCSプロバイダの上位になったMCSプロバイダは、対応するMTcfが同じMCSコネクションを経て戻るように、未応答のMTrqの各々と、それがどこから到着したかを記録する。

中間のMCSプロバイダは、トークン属性の中に宣言されたユーザIDがMTrqを発信したサブツリーに正しく割り当てられていることを保証するために、それらの正当性を検証する。無効なユーザIDは削除する。獲得者もしくは受取人のどちらか一方が削除されても、譲渡途中のトークンは獲得されたままになる。もし、トークンがユーザIDの削除によって解放されるようになるならば、そのすべての属性はマージの要求から消去され、トークンIDだけを削除されるセットの方に移さなければならない。抑制者が他のMCSPDUで残るかもしれないので、たとえすべての抑制者が削除されても、抑制されたトークンは、属性を空集合のままにしてMTrqにおいて抑制され続ける。このユーザIDの正当性の検証を除いて、中間のMCSプロバイダはMTrqの内容を修正しない。

以前のトップ プロバイダは、トークンをマージするために属性を提示し始める前に、すべてのユーザIDがマージしたドメインに組みこまれるか削除されるのを待つ必要がある。

10.11 MTcf

MTcfは、前述のMTrqに対する応答である。MTcfはMTrqと同じセットのトークンIDおよび属性のサブセットをパラメータとして持つ。マージされたドメインに組み入れられなかったトークン属性は、除去されるトークンIDとして報告される。

表１０－１１／ＪＴ－Ｔ１２５　MTcf MCSPDU

(ITU-T　T.125)

Contents	ソース	シンク
マージ トークン(Merge Tokens)	トップ プロバイダ	中間プロバイダ
削除トークン ID(Purge Token Ids)	トップ プロバイダ	以前のトップ

受け入れられたトークンIDは、トップMCSプロバイダの情報ベースに登録された属性が反映されている。中間のプロバイダは、この情報に一致させるため、自身の情報ベースを更新する。

下位のドメインから削除されるトークンは、トークンIDだけ列挙される。もし、上位のドメインにおいて同じトークンIDが利用されるなら、上位ドメイン側のトークンは削除されずに残される。中間のプロバイダは削除されたトークンIDをそのまま転送する。

MCSプロバイダは、 MTrqを転送したときの記録を利用して、MTcfをMTrqの発信元に向け送信する。MTcfは、MTrqを生成した以前のトップMCSプロバイダに返送される。そこでは、返答（MTcf）を受信するまでの間、トークンIDは情報ベース中にあるため、マージされるトークンは無視してもよい。削除されたトークンIDは、PTinのパラメータとして引き継がれた後に、情報ベースから削除される。

中間のMCSプロバイダは、トークン属性において宣言されたユーザIDが、MTcfが転送されるサブツリーに割り当てられることを確認する。先行するMTrqが有効となってから、どのようなユーザIDがデタッチされ、どこかに再割当されても、そのユーザIDがドメインマージの障害となる中間のプロバイダは、理由コード「チャネル削除(channel purged)」でDUrqを生成する。もし、非抑制状態であるトークンIDが、このユーザIDの削除によって解放されるならば、削除されたトークンIDのセット(Purge Token Ids)に移される。

１０．１２　ＰＴｉｎ

PTinは、以前のトップMCSプロバイダにおいてMTcfの受信に従って生成される。PTinは下方に同報送信され、下位のプロバイダから指定されたトークンIDを削除する。

表 10-12/JT-T125　PTin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
削除トークン ID (Purge Token Ids)	以前のトップ	下位者

トークンを削除することによる悪影響は大きい。すなわち、「獲得(grabbed)」か、「抑制(inhibitted)」か、または「譲渡(given)」となりつつあるそのトークンIDの１つを所有している、どのユーザへもMCS-DETACH-USER-指示が発行される。プロバイダは、MCS-DETACH-USER-指示を受けたユーザに代わって、「トークンが除去された(token purged)」という「理由(reason)」を含んだDUrqを生成してこ

れを実行する。

注　TTC標準JT-T122は将来、この環境で MCS-TOKEN-RELEASE-指示を許容するように改訂されることが予想される。このことは、影響を受けたユーザが、たとえそのトークンを利用する権利を取り上げられたとしてもアタッチしたままであることを許容するものである。

１０．１３　ＤＰｕｍ

DPumは、MCS-DISCONNECT-PROVIDER-要求により生成される。そして、MCSコネクションの相手側終端で MCS-DISCONNECT-PROVIDER-指示が生成される。DPumを伝送したMCSコネクションは、受信側から切断される。

表１０－１３／ＪＴ－Ｔ１２５　DPum MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
理由(Reason)	要求したプロバイダ	指示

DPumはまた、MCSプロバイダがドメイン階層におけるサイクル（ループ）のようなエラーを検出したとき、MCSプロバイダにより生成される。そのような場合、「理由(reason)」は「ユーザ要求(user requested)」以外である。

１０．１４　ＲＪｕｍ

MCSプロバイダが無効なMCSPDUを受信したか、あるいはMCSプロトコルエラーを検出した時、RJumは生成される。起こり得ない状況からの回復は保証できないため、RJumを使用してMCSコネクションの相手側プロバイダにコネクション切断を依頼する。

表１０－１４／ＪＴ－Ｔ１２５　RJum MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
診断(Diagnostic)	リジェクしたプロバイダ	リジェクトされたプロバイダ
イニシャル オクテット(Initial Octets)	リジェクトしたプロバイダ	リジェクトされたプロバイダ

RJumがエラーを診断し、誤りのあるTSDUの先頭部分を返す。なお、この中には、MCSＰＤＵの最大サイズ以下で、検出した誤りまでのオクテットを含める。受信側プロバイダには、切断かあるいは続行するかの選択権がある。

１０．１５　ＡＵｒｑ

AUrqは、MCS-ATTACH-USER-要求により生成される。それは、トップMCSプロバイダまで伝送され、トップMCSプロバイダはAUcfを返送する。もし、ユーザIDの数がドメイン制限の許容内であるなら、新しいユーザIDが生成される。

表10-15/JT-T125　AUrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
なし	－	－

AUrqは、MCSPDUタイプ以外の情報を含まない。ユーザがアタッチするドメインは、MCSPDUを伝送するMCSコネクションにより決定される。生成されるユーザIDの唯一の基本特性は番号が重複しないことである。MCSプロバイダは、応答のAUcfをAUrq発信元に返送できるように、受信したAUrqのうち未応答のものと、それが到達したMCSコネクションを記録する。返答を公正に分配するために、各プロバイダはファーストイン ファーストアウトのキューを持つべきである。

１０．１６　ＡＵｃｆ

AUcfは、AUrqを受信したトップMCSプロバイダにおいて生成される。要求を発行したプロバイダにAUcfが返送されると、そのプロバイダは MCS-ATTACH-USER-確認を生成する。

表１０－１６／ＪＴ－Ｔ１２５　AUcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者(オプション)(Initiator (optional))	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

もし、結果が「成功(successful)」のときに限り、AUcfはユーザIDを含む。結果が「成功(successful)」であるAUcfを受信したプロバイダは、ユーザIDを情報ベースに登録する。MCSプロバイダは、AUrqを転送したときの記録を利用して、AUcfをAUrqの発信元に送信する。AUcfを送信するプロバイダは、後に他の要求においてユーザIDの正当性を検証できるように、下方のMCSコネクションに新しいユーザIDを割り当てたことを記録する。

１０．１７　ＤＵｒｑ

DUrqは、MCS-DETACH-USER-要求によって生成される。もし有効なら、トップMCSプロバイダまで

上がる。トップMCSプロバイダは、情報ベースからユーザを削除し、また、その変更を他のプロバイダに通知するためにDUinを同報送信する。

表10-17/JT-T125　DUrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
理由(Reason)	要求したプロバイダ	トップ プロバイダ
ユーザID(User Ids)	要求したプロバイダ	トップ プロバイダ

MCS-DETACH-USER-要求は、「ユーザ要求(user-requested)」の「理由(Reason)」と、1つのユーザIDを含んだDUrqを生成する。

DUrqは、下方へのMCSコネクションが切断された時にもMCSプロバイダによって生成される。その時点で、影響を受けたサブツリー内の全てのユーザは、デタッチしたことを「ドメイン切断(domain disconnected)」の「理由(Reason)」とともに報告される。もし、ユーザIDの割り当て途中であるときは、それ以降の返答のMCcfまたはAUcfは、切断されたサブツリー内の要求元には返送されない。切断されたMCSコネクションの上位側のプロバイダもまた、割り当てのできないユーザIDを削除するためにDUrqを生成する。DUrqを受信したプロバイダは、それに含まれるユーザIDが発信元のサブツリーに正しく割り当てられていることを保証するために、それらの正当性を検証する。無効なユーザIDは削除される。ユーザIDが残っていなければ、DUrqは無視される。

DUrqに含まれるユーザIDは、プロバイダがDUinを受信するまで情報ベースから削除してはならない。これはトップMCSプロバイダと一貫性を保つために必要である。

注

もし、MCSドメイン中に2つ以上のデータプライオリティが実装された場合には、DUinは、同じユーザから先に送信された低いプライオリティのデータよりも早くプロバイダに到着するかもしれない。送信者がデタッチしたことをDUinにより通知された後に、その送信者からのデータをアタッチメントに配信することを、このプロトコルでは抑制していない。

10.18　DUin

DUinは、トップMCSプロバイダにおいて、DUrqの受信により生成される。DUinは、下方の全てのプロバイダへ同報送信され、また、全てのアタッチメントにおいてMCS-DETACH-USER-指示を生成する。

表10-18/JT-T125　DUin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
理由(Reason)	トッププロバイダ	指示
ユーザ ID(User Ids)	トッププロバイダ	指示

指示＝指示プリミティブ

残存しているアタッチメントにおいて、DUinはそのパラメータである各々のユーザIDに対し１つのMCS-DETACH-USER-指示を生成する。通知されたユーザが、それ以前に、デタッチされたユーザの存在を知っていたか否かは問題ではない。

自分のユーザIDを含んだDUinを受信すると、MCSアタッチメントは消滅する。ユーザがデタッチして非加入となった、スタティックまたは割当チャネルは、CLrqにより削除される。

DUinを受信したプロバイダは、マネージャまたはプライベートチャネルに許可されたユーザが含まれる情報ベースから指定されたユーザIDを削除する。デタッチしたユーザによって管理されていたプライベートチャネルは、もし他に加入しているユーザが無ければ削除される。もし、他のユーザが残っている場合には、マネージャの削除が、トップMCSプロバイダにそれらのユーザへCDinを同報送信させることとなる。もし、プライベートチャネルに加入したユーザセットが空になって、マネージャがサブツリー内に存在しないなら、チャネルIDは情報ベースから削除される。さもなければ、もし、プライベートチャネルがユーザデタッチの結果として非加入なら、プロバイダが対応するCLrqを生成する。

影響を受けたプライベートチャネルとサブツリーの各々に関して、サブツリーがその後もプライベートチャネルに加入したアタッチメントを含んでいるかどうか、DUinを同報送信しているプロバイダは計算する。もし、無いなら、プロバイダは対応する下位のプロバイダがもはやプライベートチャネルに加入していないと判断し、そして、CLrqを待たずに、ただちに、情報ベースをこの結果でアップデートする。

デタッチしたユーザにより「獲得(grabbed)」か、「譲渡中(being given)」か、あるいは「抑制(inhibited)」の状態であったトークンは、それらの状態を適合させる。トークンを譲渡したユーザがトークンを解放せず、かつそのユーザがデタッチしないときに、トークンの受取人が削除されたら、トップMCSプロバイダはトークンを譲渡したユーザへ失敗のTVcfを生成する。

１０．１９　ＣＪｒｑ

CJrqは、MCS-CHANNEL-JOIN-要求によって生成される。もし有効なら、それはCJcfで応答するための十分な情報をもつMCSプロバイダまで上がる。これはトップMCSプロバイダかも知れない。

表１０-１９/JT-T125　CJrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	上位プロバイダ
チャネルID(Channel Id)	要求	上位プロバイダ

要求＝要求プリミティブ

要求したMCSアタッチメントのユーザIDは、要求プリミティブを受け取ったMCSプロバイダによって供給される。その後CJrqを受信したプロバイダは、それに含まれるユーザIDが発信元のサブツリーに正しく割り当てられていることを保証するために、その正当性を検証する。もしユーザIDが無効なら、MCSPDUは無視される。

注

これは、下方に向かう「要求者(Initiator)」のユーザIDの削除と競合してCJrqが上方に向かう可能性を考慮している。PCinを先に受信したプロバイダが、その後すぐに無効なユーザIDを含むCJrqを受け取るかもしれない。これは正常な出来事であって、MCSPDUを拒絶する理由にはならない。

CJrqは、その情報ベースに要求されたチャネルIDを持つMCSプロバイダまで上がる。そのようなプロバイダは、トップMCSプロバイダと一貫性が保たれているので、要求が成功するべきかどうかについても一致するであろう。もし要求が失敗なら、プロバイダは失敗のCJcfを生成する。もしそれが成功し、かつプロバイダが同じチャネルに既に加入しているなら、プロバイダは成功のCJcfを生成する。これらの２つのケースでは、MCS-CHANNEL-JOINがトップMCSプロバイダに届く前に完了するかもしれない。一方、もし要求が成功するべきで、しかしチャネルにまだ加入していないなら、プロバイダはCJrqを上位へ上げるべきである。

もしCJrqがトップMCSプロバイダまで上がり、その要求するIDがゼロなら、無効なIDであるため情報ベースにはない。その場合、もし許可されたチャネルの値がドメイン制限内なら、新しい割当チャネルIDが生成され、成功のCJcfが返される。もし要求されたチャネルIDがスタティックの範囲で、許可された値がドメイン制限内なら、チャネルIDは情報ベース中に登録され、同様に成功のCJcfが返される。さもなければ、トップMCSプロバイダの情報ベースに既にチャネルIDが存在する場合に限り、要求は成功する。ユーザIDチャネルは同一ユーザのみ加入することができる。プライベート チャネルIDはそのマネージャによって許可されたユーザだけが加入することができる。割当チャネルIDはどんなユーザも加入することができる。

１０．２０　ＣＪｃｆ

CJcfは、CJrqを受信した上位MCSプロバイダにより生成される。要求を発行したプロバイダに返送されると、そのプロバイダはMCS-CHANNEL-JOIN-確認を生成する。

表１０－２０／ＪＴ－Ｔ１２５　CJcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	上位プロバイダ	確認
要求者(Initiator)	上位プロバイダ	MCSPDU 経路
要求チャネル ID(Requested)	上位プロバイダ	確認
チャネル ID(オプション) (Channel Id (optional))	上位プロバイダ	確認

確認 ＝ 確認プリミティブ

「結果(Result)」が「成功(successful)」である場合に限り、CJcfは加入したチャネルIDを含んでいる。「要求チャネルID(Requested)」はCJrqと同じである。これは、要求アタッチメントがMCS-CHANNEL-JOIN-確認を先行するMCS-CHANNEL-JOIN-要求に関連付けるのを手助けする。CJrqがトップ プロバイダにあがることを必要としないので、CJcfは異なる順序で受信されるかもしれない。

もし「結果(Result)」が「成功(successful)」であるならば、CJcfが受信したMCSプロバイダは、指定された「チャネル(Channel Id)」に加入したことになる。その後、上位プロバイダはユーザーがチャネル上に送信するあらゆるデータを転送する。プロバイダは、少なくともアタッチメントあるいは下位のプロバイダが加入している限り、チャネルに加入したままとするべきである。チャネルを離脱するためには、プロバイダは CLrq を生成するべきである。

成功であるCJcfを受信するプロバイダは、情報ベース内にチャネルIDを登録するべきである。もしチャネルIDがそこに既に存在しないならば、チャネルIDはレンジによって識別され、スタティックタイプあるいは割当タイプを与えられるべきである。

CJcfは、要求者の方向へ転送されるべきである。もしMCSコネクションがもはや存在しないため、要求者に到着しないなら、プロバイダはそれがチャネルに加入しておく理由があるかどうか決めるべきである。もしそうでなければ、CLrqを生成するべきである。

１０．２１　ＣＬｒｑ

CLrqは、それ自身をチャネルのセットから除去するためにMCSプロバイダによって生成される。そのきっかけとなるのは、そのチャネルに加入している最後のアタッチメントからのMCS-CHANNEL-LEAVE-要求かもしれない。CLrqは、もし上位プロバイダが、結果として非加入になるならば、上がり続ける。

表１０-２１／JT-T125　CLrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
チャネルID (Channel Ids)	要求したプロバイダ	上位プロバイダ

CLrqを受信したプロバイダは、それを運んだMCSコネクションに対し、ユーザがそのチャネルを通して送った全てのデータの転送を止める。最後のアタッチメントあるいは下位プロバイダがチャネルを離脱する時、MCSプロバイダは対応するCLrqを生成する。

１０．２２　ＣＣｒｑ

CCrqは、MCS-CHANNEL-CONVENE--要求によって生成される。もし有効なら、トップMCSプロバイダまで上がる。トップMCSプロバイダはCCcf応答を返す。もしチャネルIDの数がドメイン制限内なら、新しいプライベートチャネルIDが生成される。

表１０-２２/JT-T125　CCrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トッププロバイダ

CCrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。
要求者はプライベート チャネルのマネージャになる。最初はチャネルは非加入で、そのマネージャは唯一許可されたユーザである。

１０．２３　ＣＣｃｆ

CCcfは、CCrqを受信したトップMCSプロバイダにおいて生成される。要求を発行したプロバイダに返送されると、そのプロバイダはMCS-CHANNEL-CONVENE-確認を生成する。

表１０－２３／Ｔ．１２５ CCcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者(Initiator)	トップ プロバイダ	MCSPDU経路
チャネルID(オプション) (Channel Id tional))	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

「結果(Result)」が「成功(successful)」である場合に限り、CCcfはプライベートチャネルIDを含んでいる。

成功であるCCcfを受信するプロバイダは、その要求ユーザIDをマネージャとするプライベートチャネルとして情報ベース内のチャネルIDに登録するべきである。

CCcfは、要求ユーザーIDの方向へ転送されるべきである。もしMCS コネクションがもはや存在しないため、要求者に到着しなくてもDUinが、要求者がデタッチしたことを報告するために、後で必ず到着するので、特別な処理を必要としない。要求者はマネージャなので、結果として情報ベースからチャネルIDが削除される。

１０．２４　ＣＤｒｑ

CDrqは、MCS-CHANNEL-DISBAND-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がる。トップMCSプロバイダはプライベート チャネルIDを削除してCDinを生成する。

表１０-２４/JT-T125　CDrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
チャネルID(Channel Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

CDrqは、自分自身の要求によりチャネルを解放するために、MCSプロバイダによっても生成されるかもしれない。

CDrqは「要求者(Initiator)」のユーザIDを含み、それに含まれるユーザIDが発信元のサブツリーに正しく割り当てられていることを保証するために、その正当性を検証する。もし要求者がプライベート チャネルのマネージャでないならば、MCSPDUは無視される。

１０．２５　ＣＤｉｎ

CDinは、トップMCSプロバイダにおいてCDrqの受信により生成される。CDinは、サブツリー中のマネージャあるいは許可されたユーザの、プロバイダに向けて下方へ同報送信される。それは、許可されたユーザへ「チャネル解放(channel disbanded)」という「理由(reason)」を含むMCS-CHANNEL-EXPEL-指示を生成する。

表10-25/JT-T125　CDin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
チャネルID(Channel Id)	トッププロバイダ	指示

指示＝指示プリミティブ

プライベートチャネルのマネージャがデタッチしているときも、CDinはトップMCSプロバイダによって生成される。

CDin を受信したプロバイダが、それらの情報ベースからチャネルを削除する。

１０．２６　ＣＡｒｑ

CArqは、MCS-CHANNEL-ADMIT--要求によって生成される。もし有効なら、トップMCSプロバイダまで上がる。プライベートチャネルへ指定されたユーザを許可し、それらが属するサブツリーのプロバイダへ指示するために、 CAinを同報送信する。

表１０-２６／JT-T125　CArq　MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
チャネルID(Channel Id)	要求	トップ プロバイダ
ユーザID(User Ids)	要求	トップ プロバイダ

要求＝要求プリミティブ

CArqは「要求者(Initiator)」のユーザ ID を含み、前述のCDrqのように正当性を検証される。

CArqの「ユーザID(User Ids)」は許可されたユーザを表わし、ユーザ全体を唯一把握しているトップMCSプロバイダによって正当性を検証される。それらのうち無効なものは、結果として生じるCAinから除外される。

プロバイダがCAinを受信するまで、CArqに含まれるユーザIDをプライベート チャネルに許可されたとみなしてはならない。これはトップ MCSプロバイダとの一貫性を保つためである。

１０．２７　ＣＡｉｎ

CAin は、トップMCSプロバイダにおいてCArqの受信によって生成される。CAinは、それらのサブツ

リー中の新しく許可されたユーザのプロバイダに向けて下方へ同報送信される。それは影響を受けたアタッチメントにおいて、 MCS-CHANNEL-ADMIT-指示を生成する。

表10-27/ JT-T125　CAin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	トッププロバイダ	指示
チャネルID(Channel Id)	トッププロバイダ	指示
ユーザID(User Ids)	トッププロバイダ	MCSPDUルート

指示＝指示プリミティブ

通常、 CAin を受信したプロバイダが、そのサブツリーに属する指定されたユーザが許可されたことを反映して情報ベース中のそのチャネルの情報をアップデートする。しかしながら、もしプロバイダがMCS-CONNECT-PROVIDERの結果としてマージされた、下位のドメインの以前のトップであるならば、影響を受けたユーザIDのために「チャネル追放(channel purged)」という「理由(reason)」を含むDUrqを生成することによって許可を拒絶する。

１０．２８　ＣＥｒｑ

CErqは、MCS-CHANNEL-EXPEL-要求によって生成される。もし有効ならトップMCSプロバイダまで上がる。トップMCSプロバイダはプライベート チャネルから指定されたユーザを追放し、それらの属するサブツリー中のプロバイダにCEinを同報送信で通知する。

表１０-２８/JT-T125　　CErq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
チャネルID(Channel Id)	要求	トップ プロバイダ
ユーザID(User Ids)	要求	トップ プロバイダ

要求＝要求プリミティブ

CErqは「要求者(Initiator)」のユーザIDを含み、前述のCDrqのように正当性を検証される。

CErqの「ユーザID(User Ids)」は追放されるユーザを表し、許可されたユーザの全体を唯一知っているトップMCSプロバイダによって正当性を検証される。許可されていなかったユーザは、結果として生じるCEinから除外される。

プロバイダがCEinを受信するまで、CErqに含まれたユーザIDをプライベート チャネルから追放されたとみなしてはならない。これはトップMCSプロバイダとの一貫性を保つためである。

１０．２９　ＣＥｉｎ

CEinは、トップMCSプロバイダにおいてCErqの受信により生成される。CEinは、それらのサブツリー中の追放されるユーザのプロバイダに向けて下方へ同報送信される。それは、影響を受けたアタッチメントにおいて、「ユーザ要求(user requested)」という「理由(reason)」を含むMCS-CHANNEL-EXPEL-指示を生成する。

表10-29/ JT-T125　CEin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
チャネルID(Channel Id)	トッププロバイダ	指示
ユーザID(User Ids)	トッププロバイダ	MCSPDU経路

指示＝指示プリミティブ

CEin を受信したプロバイダが、チャネルに許可されたユーザのセットから指定されたユーザを削除して、それらの情報ベース中のチャネルをアップデートする。もし、プライベートチャネルに許可されたユーザのセットが空になり、またマネージャがサブツリーに属さないならば、チャネル ID は情報ベースから削除される。その他に、もしチャネルが排除の結果として非加入となるならば、プロバイダは対応するCLrqを生成する。

各宛先サブツリーに関して、プライベートチャネルへ許可されたアタッチメントをも含むかどうか CEin を転送するプロバイダが計算する。もし含まないならば、プロバイダは、対応する下位のプロバイダがもはやプライベートチャネルに参加していないと判断し、そして、CLrqを待たずに、ただちにその情報ベースをこの結果でアップデートする。

１０．３０　ＳＤｒｑ

SDrq は、 MCS-SEND-DATA-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がる。通信路に沿って、プロバイダが同一の内容を伴ったSDinをそのSDrqから生成し、下方へ同報送信する。

表10-30/JT-T125　SDrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	上位プロバイダ
チャネルID(Channel Id)	要求	上位プロバイダ
データ プライオリティ (Data Priority)	要求	上位プロバイダ
セグメンテーション (Segmentation)	要求したプロバイダ	上位プロバイダ
ユーザデータ(User Data)	要求	上位プロバイダ

要求＝要求プリミティブ

SDrqは「要求者(Initiator)」のユーザID を含み、前述のCJrqのように正当性を検証される。

もし、この「チャネルID(Channel Id)」が受信MCSプロバイダの情報ベース中にプライベート　チャネルとして記録されており、SDrqの要求者が許可されたユーザでないならば、 MCSPDUは無視される。

ドメイン内で実装されたプライオリティの数を考慮に入れて、SDrqを伝送する初期TCか付加TCのデータ プライオリティを一致させなければならない。間違ったTCによってMCSコネクション上を伝送されたMCSPDUは拒絶される。

セグメンテーションフラグである*begin*と*end*は、MCSサービス データ ユニットの境界に対して、SDrqの中のユーザデータの関係を示すために、プロバイダによってセットされる。プロバイダはユーザデータの完全性を破壊しないかぎり、プロバイダが同一のMCSサービス　データ ユニットの一部分であるMCSPDUを分解したり、再組み立てする自由をもつ。但し、ドメインのMCSPDUのサイズは一定であるため、このような操作には利点が殆どない。

プロバイダは、SDrqから同じ内容を持ったSDinを生成する。またそれを、SDrqを上方に転送した下位プロバイダを除き、指定されたチャネルに加入している全てのプロバイダへ転送する。そのチャネルが、そのサブツリーに属しているユーザIDとしてプロバイダの情報ベース中に記録されていなければ、上向きにSDrqを転送する。

１０．３１　ＳＤｉｎ

SDinは、より上位の MCS プロバイダにおいてSDrqの受信により生成される。SDinは下方へ同報送

信され、チャネルに加入している全てのアタッチメントにおいてMCS-SEND-DATA-指示を生成する。

表10-31/JT-T125　SDin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	上位プロバイダ	指示
チャネルID(Channel Id)	上位プロバイダ	指示
データプライオリティ (Data Priority)	上位プロバイダ	指示
分割(Segmentation)	上位プロバイダ	指示した プロバイダ
ユーザデータ(User Data)	上位プロバイダ	指示

指示＝指示プリミティブ

ドメインで実装される多くのプライオリティを考慮に入れ、SDinを運ぶ初期TCか付加TCのデータプライオリティを一致させる。間違ったTCによってMCSコネクションの上に到着しているMCSPDUは拒絶される。

セグメンテーションフラグであるbeginとendを使うことによって、ユーザデータは完全なMCSサービスデータユニットに再組立てされる。これらのフラグは、同一チャネル、同一プライオリティでの同一ユーザから到着しているSDin MCSPDUのコンテキストの中で解釈される。分割されたストリームは異なったプライオリティや異なったチャネルや別ユーザなどから送られる、いろいろな種類のMCSPDUと混在している。

アタッチしているMCSユーザにサービスデータユニットが如何に指示されるかはローカルな実装問題である。含まれたセグメンテーションフラグと共に、別々のインターフェイスデータユニットとして、各々の MCSPDU を渡す方法もあるし、他に、受信側のプロバイダで大きなサービスデータユニットのための準備をして再組立を行い、ユーザにサービスデータユニットが相対的に到着した順番通りに渡す方法もある。

SDinを受信したプロバイダは、チャネルに加入した全下位者に配送する。

１０．３２　ＵＳｒｑ

USrqはMCS-UNIFORM-SEND-DATA-要求により生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダが同一内容のUSinを生成して下方に同報送信する。

表10-32/JT-T125　USrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	上位プロバイダ
チャネルID(Channel Id)	要求	上位プロバイダ
データ プライオリティ(Data Priority)	要求	上位プロバイダ
セグメンテーション(Segmentation)	要求したプロバイダ	上位プロバイダ
ユーザデータ(User Data)	要求	上位プロバイダ

要求＝要求プリミティブ

USrqは「要求者(Initiator)」のユーザID を含み、前述のCJrqのように正当性を検証される。

もし、この「チャネルID(Channel Id)」が受信MCSプロバイダの情報ベース中にプライベート　チャネルとして記録されており、USrqの要求者が許可されたユーザでないならば、 MCSPDUは無視される。

ドメイン内で実装されたプライオリティの数を考慮に入れて、USrqを伝送する初期TCか付加TCのデータ プライオリティを一致させなければならない。間違ったTCによってMCSコネクション上を伝送されたMCSPDUは拒絶される。

セグメンテーションフラグである*begin*と*end*は、MCSサービス データ ユニットの境界に対して、USrqの中のユーザデータの関係を示すために、プロバイダによってセットされる。プロバイダはユーザデータの完全性を破壊しないかぎり、プロバイダが同一のMCSサービス　データ ユニットの一部分であるMCSPDUを分解したり、再組み立てする自由をもつ。但し、ドメインのMCSPDUのサイズは一定であるため、このような操作には利点が殆どない。

トップMCSプロバイダはUSrqから同じ内容のUSinを生成する。

１０．３３　ＵＳｉｎ

USinはトップMCSプロバイダにおけるUsrqの受信により生成される。USinは下方へ同報送信され、チャネルに加入している全てのアタッチメントにおいてMCS-UNIFORM-SEND-DATA-指示を生成する。

表10-33/JT-T125　USin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	トッププロバイダ	指示
チャネルID(Channel Id)	トッププロバイダ	指示
データプライオリティ (Data Priority)	トッププロバイダ	指示
分割(Segmentation)	トッププロバイダ	指示したプロバイダ
ユーザデータ(User Data)	トッププロバイダ	指示

指示＝指示プリミティブ

ドメインで実装される多くのプライオリティを考慮に入れ、USrqを運ぶ初期TCか付加TCの、そのデータプライオリティを一致させる。間違ったTCによりMCSコネクション上に到着しているMCSPDUは拒絶される。

セグメンテーションフラグであるbeginとendを使うことによって、ユーザデータは完全なMCSサービスデータユニットに再組立てされる。これらのフラグは同一チャネル、同一プライオリティでの同一ユーザから到着しているUSin MCSPDUのコンテキストの中で解釈される。分割されたストリームは、異なったプライオリティや異なるチャネルや別ユーザなどから送られている、いろいろなMCSPDUと混在している。

アタッチしているMCSユーザにサービスデータユニットが如何に指示されるかはローカルな実装問題である。

USinを受信したプロバイダは、チャネルに加入した全下位者に配送する。

１０．３４　ＴＧｒｑ

TGrqはMCS-TOKEN-GRAB-要求により生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはTGcfで応答する。

表10-34/JT-T125　TGrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

TGrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

もしトークンが解放されていて(Not in Use)、使用中のトークンの数のドメイン制限に許される場合は、トークンは「獲得(grabbed)」状態となる。トークンが要求者のみによる「抑制(inhibited)」状態の時にもトークンは「獲得(grabbed)」状態となる。他の場合には、トークンの状態は変わらない。

１０．３５　ＴＧｃｆ

TGcfは、 TGrqを受信したトップMCSプロバイダによって生成される。要求を発行したプロバイダにTGcfが返送されると、そのプロバイダはMCS-TOKEN-GRAB-確認を生成する。

表１０－３５／ＪＴ－Ｔ１２５ TGcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者(Initiator)	トップ プロバイダ	MCSPDU 経路
トークン ID(Token Id)	トップ プロバイダ	確認
トークン ステータス(Token Status)	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

トークンの状態が既に「フリー(not in use)」になっている場合、あるいは同じユーザにより状態が「抑制(inhibited)」から「獲得(grabbed)」に変わった場合、その「結果(result)」は「成功(successful)」となる。他の「結果(result)」には、「トークン過多(too many token)」と「トークン利用不可(token not available)」がある。後者は既に他者に獲得されたトークンに対して適用される。このことはトークンの状態を検査することによって推測できる。

TGcfを受信したプロバイダは、返値である「トークン ステータス(token status)」と一致させるため、情報ベース内のトークン状態を更新する。

TGcfは要求者の方向に転送される。MCSコネクションがもはや存在しないため、要求者に到達しなくても、特別な処理をする必要はない。なぜなら、要求者がデタッチしたことを報告するため、DUinが必ず後に到着するからである。結果として、情報ベース上に保持されているトークンIDが解放される。

１０．３６　ＴＩｒｑ

TIrqはMCS-TOKEN-INHIBIT-要求により生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはTIcfで応答する。

表10-36/JT-T125　TIrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

TIrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

もしトークンが解放されていて(Not in Use)、使用中のトークンの数のドメイン制限に許される場合は、トークンは「抑制(inhibited)」状態となる。トークンが要求者による「獲得(grabbed)」状態の時には、「抑制(inhibited)」状態に変わる。もしトークンが既に「抑制(inhibited)」状態の場合は、要求者が抑制者のグループに加えられる。他の場合には、トークンの状態は変わらない。

１０．３７　ＴＩｃｆ

TIcfは、TIrqを受信したトップMCSプロバイダによって生成される。TIcfが要求を発行したプロバイダに返送されると、そのプロバイダはMCS-TOKEN-INHIBIT-確認を生成する。

表１０－３７／ＪＴ－Ｔ１２５　TIcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者 Initiator)	トップ プロバイダ	MCSPDU 経路
トークン ID(Token Id)	トップ プロバイダ	確認
トークン ステータス(Token Status)	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

トークンの状態が既に「フリー(not in use)」または「抑制(inhibited)」になっている場合、あるいは同じユーザにより状態が「獲得(grabbed)」から「抑制(inhibited)」に変わった場合、その結果は「成功(successful)」となる。他の「結果(result)」には、「トークン過多(too many token)」と「トークン利用不可(token not available)」がある。

TIcfを受信したプロバイダは、返値である「トークン ステータス(token status)」と一致させるため、情報ベース内のトークン状態を更新する。

このMCSPDUはTGcfと同じ方式で転送される。

１０．３８　ＴＶｒｑ

TVrq は、MCS-TOKEN-GIVE-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはTVinか失敗のTVcfのいずれかを生成する。

表10-38/JT-T125　TVrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ
受取人(Recipient)	要求	トップ プロバイダ

要求＝要求プリミティブ

TVrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

もしトークンが要求者による「獲得(grabbed)」状態で、受取予定者が存在する場合、TVinは受取人の方へ送られる。さもなければ、要求は失敗し、トークンの状態は変わらず、「トークン非所有(token not possessed)」もしくは「該当ユーザなし(no such user)」という「結果(Result)」のTVcfが要求者の方へ送られる。

１０．３９　ＴＶｉｎ

TVinは、トップMCSプロバイダにおけるTVrqの受信により生成される。受取予定者に発送され、MCS-TOKEN-GIVE-指示を生成する。

表10-39/JT-T125　TVin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	トッププロバイダ	指示
トークンID(Token Id)	トッププロバイダ	指示
受取人(Recipient)	トッププロバイダ	MCSPDUルート

指示＝指示プリミティブ

通常、TVinを受信したプロバイダが、要求者から受信者に「譲渡中(being given)」状態にその情報ベース

の中のトークンIDを更新する。しかしながら、プロバイダが、MCS-CONNECT-PROVIDER の結果としてマージされた下位ドメインの以前のトップであるなら、「ドメインマージ(domain merging)」という「理由(reason)」を含むTVrsによって、提供されたトークンを拒絶する。

TVinは、受取人ユーザIDの方向に転送される。もし、MCSコネクションがもはや存在しないことにより、その受取人まで到着不能であっても、特別の措置を取る必要はない。なぜなら、受取人がデタッチしたと報告するDUinが、後で到着するからである。このことは、情報ベースの中の保持していたトークンIDを解放するであろう。

１０．４０　ＴＶｒｓ

TVrs は、MCS-TOKEN-GIVE-応答によって生成される。もし有効であるならば、TVrsは、トップMCSプロバイダまで上がる。そして、トップMCSプロバイダは、トークンの提供者に結果を通知するためTVcfを生成する。

表１０－４０／ＪＴ－Ｔ１２５ TVrs MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	応答	トップ プロバイダ
受取人(Recipient)	応答したプロバイダ	トップ プロバイダ
トークンID(Token Id)	応答したプロバイダ	トップ プロバイダ

応答 ＝ 応答プリミティブ

結果の「成功(successful)」は、提供されたトークンに対する受取人の承認を示す。

応答した MCS アタッチメントのユーザIDが、応答プリミティブを受信する MCS プロバイダによって供給される。その後、TVrs を受信するプロバイダは、それに含まれるユーザIDが発信元のサブツリーに正しく割当られていることを保証するために、それらの正当性を検証する。ユーザIDが無効であるならば、MCSPDUは、無視される。

プロバイダの情報ベースに、トークンIDが受取人に対して「譲渡中」として登録されていないならば、MCSPDUは無視される。結果が「成功(successful)」である場合、トークンIDがまだ提供者によって獲得されているのなら、状態は受取人によって「獲得(grabbed)」へ更新される。さもなければ、提供者がプロバイダのサブツリーに属するかどうかによって、提供者が「獲得(grabbed)」であるという状態に復帰するか、情報ベースから削除される。トークンIDが提供者によって解放されて、「結果(result)」が「失敗(unsuccessful)」である応答を受けたなら、なら、トークンは、プロバイダの情報ベースから削除される。

MCSPDU が無効でもなく、無視もされないならば、それは上方へ転送される。前述されるようにトップ MCS プロバイダは、TVrs に従う。加えて、提供者が、まだトークンを解放していないなら、トッププロバイダは、TVrs と同じ結果を含むTVcfを生成する。

１０．４１　ＴＶｃｆ

TVcf は、TVrs を受信したトップ MCS プロバイダによって生成される。要求を発行したプロバイダに返送されると、そのプロバイダはMCS-TOKEN-GIVE-確認を生成する。

表１０－４１／ＪＴ－Ｔ１２５　　TVcf MCSPDU

(ITU-T　T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者(Initiator)	トップ プロバイダ	MCSPDU経路
トークンID(Token Id)	トップ プロバイダ	確認
トークン ステータス(Token Status)	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

トークンを予定された受取人に提供することができないならば、TVrq を受信したトップ MCS プロバイダによって TVcf が生成される。これは、TVin の生成の代わりとなる。受取人が、TVrs が受信される前にデタッチしたならば、TVcf は、結果「該当ユーザなし(no such user)」をとして生成される。

TVcf を受信するプロバイダが、返値である「トークン ステータス(token status)」に一致させるため、情報ベース内のトークン状態を更新する。

この MCSPDU は、TGcf と同じ方法で転送される。

１０．４２　ＴＰｒｑ

TPrqは、MCS-TOKEN-PLEASE-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはトークンの現在のユーザに警報を出すためにTPinを同報送信する。

表10-42/JT-T125　TPrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

TPrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

１０．４３　ＴＰｉｎ

TPinは、トップMCSプロバイダにおけるTPrqの受信により生成される。TPinは、下方へ同報送信され、MCS-TOKEN-PLEASE-指示を生成する。

表10-43/JT-T125 TPin MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	トッププロバイダ	指示
トークンID(Token Id)	トッププロバイダ	指示

指示＝指示プリミティブ

TPin を受信したプロバイダは、「獲得(grabbed)」か、「抑制(inhibited)」か、または「譲渡(given)」となりつつある状態の指定されたトークンを所有するユーザに向けて、そのユーザを含むサブツリー中の全下位者へTPinを転送する。

１０．４４　ＴＲｒｑ

TRrq は、MCS-TOKEN-RELEASE-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはTRcfで応答する。

表10-44/JT-T125　TRrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

TRrqは「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

もしトークンが要求者による「獲得(grabbed)」状態ならば、トークンは「フリー(Not in Use)」状態になる。もしそれが「抑制(inhibited)」状態ならば、要求者は抑制者のグループから削除される。このグループが空になるならば、トークンは「フリー(Not in Use)」状態になる。トークンが要求者によって譲渡されるプロセスにあるならば、TVrsを受信するまで、受取予定人に対する「譲渡(given)」という、もう一つの中間の状態に入る。他の場合には、トークンの状態は変わらない。

１０．４５　ＴＲｃｆ

TRcf は、TRrq の受信によりトップMCSプロバイダで生成される。要求を発行したプロバイダに返送さると、そのプロバイダはMCS-TOKEN-RELEASE-確認を生成する。

表１０－４５／ＪＴ－Ｔ１２５　TRcf MCSPDU

(ITU-T T.125)

内容	ソース	シンク
結果(Result)	トップ プロバイダ	確認
要求者(Initiator)	トップ プロバイダ	MCSPDU経路
トークンID(Token Id)	トップ プロバイダ	確認
トークン ステータス(Token Status)	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

トークンが、要求者によって「獲得(grabbed)」あるいは「抑制(inhibited」されるか、または要求者がトークンを「譲渡中(giving)」であるならば、結果は「成功(successful)」である。他に起こり得る結果は、「トークン非所有(token not possessed)」である。

TRcf を受信するプロバイダが、返値である「トークン ステータス(token status)」に一致させるため、情報ベース内のトークン状態を更新する。

この MCSPDU は、TGcf と同じ方法で転送される。

１０．４６　ＴＴｒｑ

TTrqは、MCS-TOKEN-TEST-要求によって生成される。もし有効なら、トップMCSプロバイダまで上がり、トップMCSプロバイダはTTcfで応答する。

表10-46/JT-T125　TTrq MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	要求したプロバイダ	トップ プロバイダ
トークンID(Token Id)	要求	トップ プロバイダ

要求＝要求プリミティブ

TTrq は「要求者(Initiator)」のユーザIDを含み、前述のCJrqのように正当性を検証される。

１０．４７　ＴＴｃｆ

TTcf は、TTrqを受信したトップMCSプロバイダによって生成される。要求を発行したプロバイダに返送されると、そのプロバイダはMCS-TOKEN-TEST-確認を生成する。

表１０－４７／ＪＴ－Ｔ１２５　TTcf MCSPDU

(ITU-T T.125)

内容	ソース	シンク
要求者(Initiator)	トップ プロバイダ	MCSPDU経路
トークンID(Token Id)	トップ プロバイダ	確認
トークン ステータス(Token Status)	トップ プロバイダ	確認

確認 ＝ 確認プリミティブ

TTcf を受信するプロバイダは、情報ベース内のトークン状態が、返値である「トークン ステータス(token status)」と一致していることを見つけるはずである。

この MCSPDU は、TGcf と同じ方法で転送される。

１１．ＭＣＳプロバイダ情報ベース

１１．１　階層的な返答

MCSプロバイダが複数のドメインをホストするとしても、その管理は個々に行われる。チャネルと使用中のトークンリソースの状態を記録するために、論理上別々の情報ベースを用いる。以下の内容が、一つのドメインのコンテキストでセットされる。

ドメイン中で管理される必要があるMCSリソースは、チャネルIDとトークンIDである。ユーザIDは、チャネルIDのサブセットである。各々のカテゴリーのどれだけのIDが同時に使用できるかを、ドメイン パラメータが制限する。この制限は、制限近くまでドメインが利用されている最悪のケースで、どれだけのメモリが情報ベースのために必要とされるかをプロバイダが計算することを許可する。

ドメインの階層中において、譲渡されたMCSプロバイダで使用されているIDは、その直接上位で使用されているIDのサブセットになる。そのIDを含んでいるMCSサービスをサポートできるところで、ID情報は記録される。より大きな記録情報は、情報をずっと更新し続けるために大きなMCSPDUトラフィックを必要とし、余分なコストがかかることがある。プロバイダで記録された情報が上位プロバイダで記録された情報と一致しているので、MCSPDU伝達遅れの範囲内で、プロバイダ情報ベースがドメイン階層を通して部分的に折り返されているといえる。

ドメインパラメータは、ドメインの最初のMCSコネクションの確立でフィックスされ、その後変更できない。各々のカテゴリーの中で指定されたIDの最大数のための容量が足りないプロバイダは、不正な見かけ上の要求を出し、ドメインに加入することをネゴシエーションしてもよい。階層中の下位に位置するプ

ロバイダは、全体の情報を管理するように求められることはない。アタッチメントと下位装置は、プロバイダにサービスの全範囲を要求することはないだろう。それでもなおその容量が実際に越えるまで、そのようなプロバイダは、そのドメインの対等なメンバであるように見えるかもしれない。この方法は、限られた能力でターミナルノードに訴えることがある。

IDは、まずトップMCSプロバイダに与えられる。次にMCSPDUの選択的な下方への通知により下位のプロバイダに与えられる。ほとんどは、同じトップダウン方法で削除される。IDを削除するMCSPDUが中継中であるために、上位のプロバイダにはないIDを、下位のプロバイダにおいて使用中として記録しておくインターバルが必要である。しかし、これは長時間ではない。制御MCSPDUは、その伝達の順序で受信し、処理される。処理が次の入力イベントに移る前に、チャネルとトークンIDの生成または削除を含むMCSPDUの処理結果が効力を生ずる。

前述のパラグラフに対する例外は、スタティック チャネルIDと割当チャネルIDの削除である。チャネルIDはCJcfの（上方から）下方への通知により使用されるが、削除はこれとは逆さまに、下方から（上方へ向かって）行なわれる。特に、全てのアタッチメントからのMCS-CHANNEL-LEAVE-要求と、全ての下位プロバイダからのCLrq MCSPDUを受けたとき、チャネル非加入となり、チャネルIDは削除される。このプロバイダからのCLrq MCSPDUの伝送は、さらに上方のプロバイダへのCLrqの伝送のきっかけとなる。したがって、これらのように、使用中として記録されたチャネルIDは、上位プロバイダで用いられているチャネルIDの厳密なサブセットである。これは、データ転送の準備としてのチャネル管理を高速化するための最適設計である。

チャネルIDは、MCcf、AUcf、CJcf、CCcf、CAinによって使用される。それらは、MCcf、PCin、DUin、CLrq、CDin、CEinによって削除される。トークンIDは、MTcf、TGcf、TIcf、TVinによって使用される。それらは、MTcf、PTin、TRcf、TVrs、TVcfによって削除される。IDが譲渡されたプロバイダで使用されるとき、MCSPDUは0、1、いくつか、またはすべてを下位のプロバイダに転送する原因になることがある。IDの使用者は、増減するかもしれない。例えば、あるユーザは、プライベート チャネルの使用を許可されて、また他のユーザは排除される。1つのIDが、プロバイダから削除されるとき、そのIDがまだ使用されていると記録しているすべての下位装置にMCSPDUは転送される。

IDの使用は、ドメインにアタッチしたユーザによって、チャネルまたはトークン上のアクションとして最終的に関係づけられる（MCSPDUの伝達を通した通信の交換において、いくらかの遅れが起こるかもしれない）。MCSプロバイダに使用中として記録されたIDは、そのプロバイダのサブツリーにおいてユーザによって活発に使われている。すべての上位プロバイダにおけるIDは、登録されたIDのサブセットである。

ユーザIDの削除は、サブツリーの中で唯一のユーザのチャネルIDとトークンIDを削除する。

チャネルIDとトークンIDの使用に関して考慮すべきことは、以下のパラグラフの中で規定される。

１１．２　チャネル情報

4種類のチャネルに対して、ある譲渡されたアタッチメントがそのチャネルIDを使用中とみなすかどうか、それゆえプロバイダの情報ベースで記述されるべきかどうか、を判断するための基準が設けられている。

ａ．スタティック チャネルID（1～1000）は、ユーザが成功のMCS-CHANNEL-JOIN-確認を受け取ることによりそのチャネルに加入し、MCS-CHANNEL-LEAVE-要求や指示によってそのチャネルを離脱しない限り使用中である。

b．ユーザIDチャネルは、成功のMCS-ATTACH-USER-確認でユーザに割り当てられ、ユーザがMCS-DETACH-USER-要求によりデタッチしない限り使用中である。

c．プライベート チャネルIDは、成功のMCS-CHANNEL-CONVENE-確認でチャネルを生成するかMCS-CHANNEL-ADMIT-指示で加入が許可され、MCS-CHANNEL-EXPEL-指示で排除されない限り、またMCS-CHANNEL-DISBAND-要求や指示でチャネルが解放されない限り使用中となる。

d．割当チャネルIDは、ユーザが成功のMCS-CHANNEL-JOIN-確認を受け取ることによりそのチャネルに加入し、MCS-CHANNEL-LEAVE-要求によって離脱しない限り使用中である。

以下の情報を、使用中のチャネルIDのために保持すべきである。

a．チャネル種別（スタティック、ユーザID、プライベート、割当）

b．どのMCSアタッチメントが、そのチャネルに加入しているか。また、下位のプロバイダにつながるどのMCSコネクションが、そのチャネルに加入しているか。

c．ユーザIDチャネルの場合、その方向、すなわちユーザIDが割り当てられているローカルMCSアタッチメント、もしくはそのサブツリーにそのユーザが所属する下位のプロバイダへの下方へのMCSコネクションかどうか。

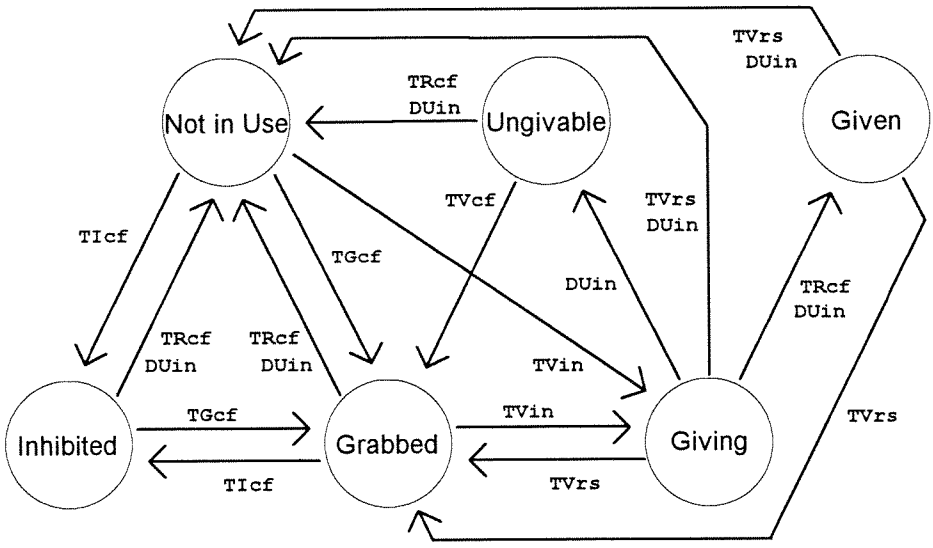
d．プライベート チャネルIDの場合、それを開設したマネージャのユーザID（マネージャ自身がそのプロバイダのサブツリー内にあるか否かにかかわらず）と、そのプロバイダのサブツリー内のそのチャネルに加入することが許可された全てのユーザID。

チャネルIDに関して記録された情報は、10章で説明したように要求MCSPDUを有効に機能させたり、指示/確認MCSPDUを経路立てて転送するために用いられる。

１１．３　トークン情報

トークンIDの状態遷移は、図11-1/JT-T125を参照。

個々のトークンIDは、一つのユーザによって**grabbed*にされるか、単一かそれ以上のユーザによって**inhibited*にされる。TVinの働きにより、トップMCSプロバイダから受取予定人までの間のドメイン階層の支線（分岐）に沿って（トークンの）状態が**giving*に変えられる。この状態は、受取人のプロバイダがTVrsで応答する前に受取人がデタッチした場合、**ungivable*に変わる。その代わりに提供者がトークンを明示的に解放するかデタッチした場合は、**given*となる。トークンの譲渡の間中、提供者から始まるドメイン階層の支線は、受取人へ向かう支線と少なくともトッププロバイダの位置で交差している。この（支線の）交差に沿ってだけ、トークン状態は*grabbed*から*giving*に変わり、またその後*ungivable*か*given*の状態になることもある。



T0812670-93

図１１－１／ＪＴ－Ｔ１２５ トークンＩＤの状態遷移
（ＩＴＵ－Ｔ．１２５）

トークンのユーザは、それの獲得者か、抑制者か、受取人か、（自分自身にトークンを譲渡するときに）獲得者と受取人の両方かの関係にある。

ａ．ユーザは、成功のMCS-TOKEN-GRAB-確認でトークンを所有し、かつ、MCS-TOKEN-RELEASE-要求か成功のMCS-TOKEN-GIVE-確認でトークンを解放しなかったか、成功のMCS-TOKEN-INHIBIT-確認でそれ（の状態）を変えなかった場合か、または、成功のMCS-TOKEN-GIVE-応答で提供されたトークンを受け入れた場合に、獲得者となる。

ｂ．ユーザは、成功のMCS-TOKEN-INHIBIT-確認でトークンを所有し、かつ、MCS-TOKEN-RELEASE-要求でそれを解放しなかったか、成功のMCS-TOKEN-GRAB-確認でそれ（の状態）を変えなかった場合、抑制者となる。

ｃ．ユーザは、MCS-TOKEN-GIVE-指示でトークンを提供され、失敗のMCS-TOKEN-GIVE-応答でそれを解放しなかった場合、受取人となる。

以下の情報が、一つの使用中のトークンIDのために記録されなければならない。

ａ．そのMCSプロバイダにおけるトークンIDの状態（トッププロバイダにおける状態と必ずしも同じではない）

ｂ．grabbedかungivableの場合、そのプロバイダのサブツリー内の獲得者のユーザID

ｃ．givingの場合、獲得者のユーザID（獲得者がそのプロバイダのサブツリー内であろうと無かろうと）

ｄ．givingかgivenの場合、そのプロバイダのサブツリー内の受取人のユーザID

ｅ．inhibitedの場合、そのプロバイダのサブツリー内の、トークンを抑制した全てのユーザーIDのセット

使用中のトークンIDのために記録された情報は、10章で説明したように、応答のMCSPDUの正当性を

検証し、指示のMCSPDUの経路を定めるために使用される。

下位のプロバイダにおけるトークンIDの状態は、トップMCSプロバイダにおける状態と同じである必要はない。これは、トークンの提供者が一般的にTVinやTVrsを処理せず、受取人が一般的に提供者のTRcfを処理しないという事実のためである。図11-2/JT-T125は、複雑なトークンの相互作用が起こる状態を表したものである。

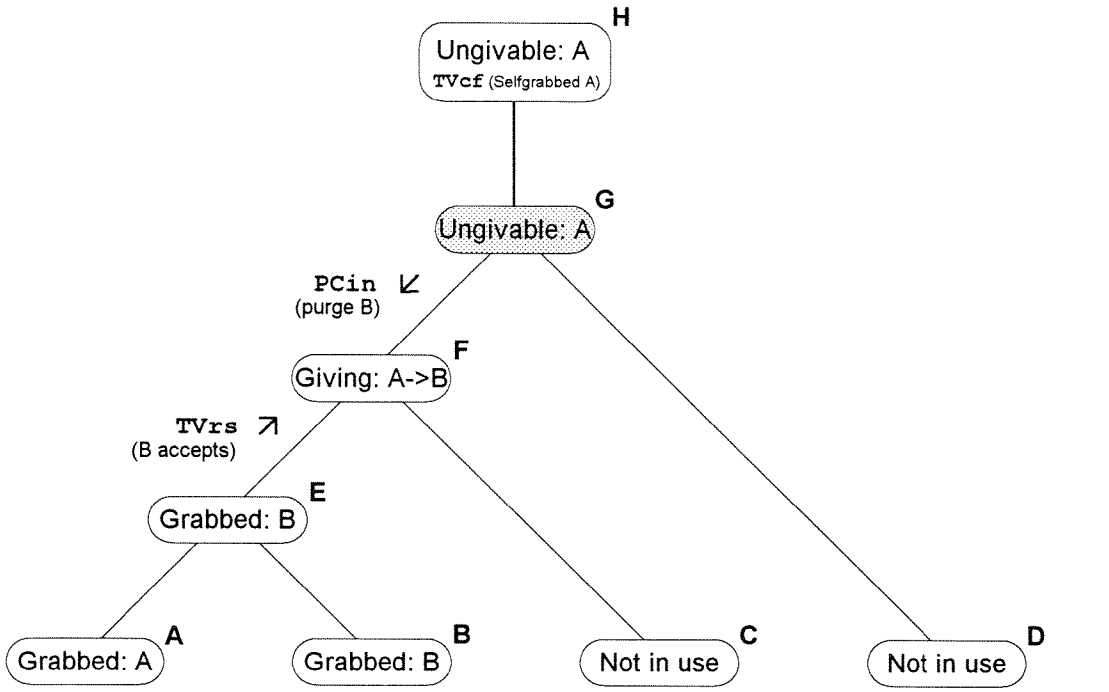


図１１－２／ＪＴ－Ｔ１２５ 複雑な相互作用が起こったトークンの状態
（ＩＴＵ－Ｔ．１２５）

図は、AからHまでのプロバイダの情報ベースにおいて、一つのトークンIDを取り上げている。このような状態に至るまでには、つぎのような経緯が考えられる。トークンが、プロバイダAにアタッチしているユーザAによって、プロバイダBにアタッチしているユーザBに譲渡された。ユーザBが応答する前ではあるが、プロバイダGが新しいトッププロバイダHにそのドメインを接続して、マージを始めた。新しいドメインは、チャネルIDの衝突のためユーザBを認めることは出来ず、PCinでその削除を開始した。このMCSPDUは、プロバイダGとDの間で、その行程の一部を（この図の時点では）完了している。その結果、プロバイダGは、*giving*から*ungivable*にトークンの状態を調整し、その状態で新しいドメイン中にトークンをマージする。トークンがそのような状態であることを知らされた新しいトッププロバイダHは、ユーザAにトークンを戻すための失敗のTVcfの送信を保留している。このとき、トークンはユーザBによって既に受け入れられ、TVrsが受取人側のドメイン階層の支線の上方向に進みつつあり、（支線の）各プロバイダでそのトークンの状態を*giving*からBによる*grabbed*に変えていた。そして、2つのMCSPDUは、プロバイダFにちょうど集まった。その後のプロバイダFでのトークンの状態は、どちらが先に到着したかによって決定される。つまり、Aによる*grbbed*になるか、一瞬Bによる*grabbed*になり、その後に、ユーザBがデ

タッチされてから、**not in use*に再び変わるかのどちらかである。どちらの場合でも、トークンの状態は、新しいトッププロバイダが保留していたTVcfを送った時、Aによる*grabbed*に落ち着く。

*（注）：*grabbed*（譲渡された）、*inhibited*（抑制された）、*giving*（譲渡中）、*ungivable*（譲渡できない）、*given*（譲渡された）、*not in use*（未使用）

１２．プロシジャの要素

１２．１　ＭＣＳＰＤＵの順序制御

コントロールMCSPDUは、MCSコネクションの単一のTC(初期TC)を通して伝送されるため、どのMCSプロバイダ間でもコントロールMCSPDUの伝送順序は維持される。MCSプロバイダはコントロールMCSPDUを受信した順番に処理し、同じ順序で転送する。これは、ドメイン階層の上位あるいは下位に転送されるコントロールMCSPDUと、要求や応答のプリミティブが指示や確認のプリミティブに変換されるコントロールMCSPDUに適用する。TCのフロー制御において送信の遅延が生じる場合でも、MCSプロバイダ内においてコントロールMCSPDUの順序は維持される。

同一プライオリティのデータ MCSPDU の順序は維持されるが、異なるプライオリティ間のデータMCSPDU の順序を維持する必要はない。このことは、高位のプライオリティのデータを、低位のプライオリティのデータよりも優先的に伝送することを可能とする。これは、同一プライオリティのデータを同じTCで伝送し、MCSプロバイダ内で各プライオリティごとに別の送信キューとして処理することを意味する。

MCS プロバイダは同一プライオリティで送信されるデータ MCSPDU の順序を維持しなければならない。これは、TTC標準JT-T122で規定される、同じチャネル上の同一プライオリティのMCSPDUの順序制御よりも信頼性が高い。

コントロールMCSPDUと最も高位のプライオリティのデータMCSPDUは同一の初期TCを使用して送信される。そして、MCSプロバイダはこれらを受信すると、これらのMCSPDUを同等に扱う。低位のプライオリティのデータは高位のプライオリティのデータの優先処理によって、その送信が遅延する場合がある。コントロールMCSPDUは、先に送信された低位のプライオリティのデータよりも先に届く場合がある。

１２．２　入力フロー制御

MCSプロバイダは、以下の要求事項を満足するべきである。

(1)一時的にデータは止まるかもしれないが、ドメイン内のデータの流れが滞らないようにする。

(2)送信側が、利用可能な帯域に対して公平なアクセスができるようにする。

(3)同じ同報データを受信している受信側間で、顕著な受信遅れが生じないようにする。

MCSはユーザデータの完全性を保てる信頼性の高いサービスである。MCSプロバイダは、すぐに転送できないMCSPDUを蓄えておくためのバッファを持つが、このバッファに空きが無いときには、入力されるデータを受け入れない。トランスポートサービスとのインタフェースの詳細はローカルな問題であるが、フロー制御が無い場合でも、受信するTSDUの順序性は維持されるべきである。TCで使用されるバッファが満杯であるとき、リモートMCSプロバイダは、バッファに空きが生じるまでMCSPDUを送信

できない。

MCS プロトコルでは、フロー制御は明示的に行われない。フロー制御は、下位レイヤの機能である。したがって、リモートプロバイダがそれ以上のデータ受信を拒否しているかどうかを TC を介して知ることは難しい。これを解決する1つの方法を以下に示す。

MCS プロバイダに、MCSPDU を蓄積するバッファを各 TC 毎に持たせる。それぞれのバッファはこのプロトコルに規定されているように処理されて、バッファ中のデータは各 TC へ送信される。データの送信はすぐに実行される場合もあるが、バッファが満杯であるときは遅延する。最後の TC へデータを送信するまでバッファへのデータ入力は行わない。全ての送信すべき TC へのデータ送信が終了した後に、そのバッファは再利用される。バッファが使い果たされた場合、その対応する TC でのデータ受信が止められる。バッファの割り当ては、コネクションが上位へのものか下位へのものかということと、データ プライオリティを考慮して決められる。

バッファによって送信側と受信側間の転送レートの不均衡を緩和することができる。TC 毎に別々のバッファを割り当てることで、一つの TC が資源を独占することを阻止できるとともに、同報データを極端な遅延なしに各々の受信側に送信することができる。なお、この方法は、予期される全てのパターンで充分に役立つとは限らない。MCSPDU の付加的な通信を必要とせずに、ローカルに実装できる、より優れたフロー制御方式の考案は、製品を差別化する手段となり得る。

12．3　スループット強制

スループット強制に対するいくつかの指針は、MCS プロトコル中に明記されている。

(1)スループットは MCS-CONNECT-PROVIDER プリミティブを使用して決定される。なお、スループットは「ドメインパラメータ(Domain Parameters)」中のパラメータである。

(2)スループットを監視する時間間隔は、MCS プロバイダが EDrq を使用してその上位階層の MCS プロバイダに通知する。

スループット強制の監視間隔を明示するためには、プロバイダがいくつかの共通の原則に基づいた動作をすることが必要である。スループット強制については、まだ技術的に改善の余地がある。

それぞれの受信側に対して最低の入力レートを要求するのは、ドメインの MCS コネクションを確立する制御アプリケーションが利用できるオプションである。このオプションはドメインパラメータの要求されたスループットから選択される。ある部分が勝手にゆっくりと動作すると他者のデータ転送を妨害するため、これは避けなければならないが、あまりに厳しいスループットを要求することは危険である。

複数の送信側が存在する複雑な場合は問題がある。データの転送が滞るのは、非常に遅い受信速度によるものの他に、以下の場合がある。

(1)下方へ向かう MCSPDU には、SDin のように、下方から上がって来てただ折り返される MCSPDU もある。そのため、同位のプロバイダを介した下方への流れは、公称帯域幅の一部分しか使用しないかもしれないし、また他のコネクションや同位のプロバイダのアタッチメントの数により使用する帯域は動的に変わるかもしれない。

(2)最も高いプライオリティの MCSPDU のみが、MCS コネクションで連続的に転送されることを期待できる。他の送信側からのより高いプライオリティのデータ受信を優先するために、同位のプロバイダによって、

低いプライオリティのデータは長い間留められる。

(3)MCS コネクション上で受信される前に留められた時間を計測すると、瞬間スループットには大きな変動がある。とりわけ、これは、同位のプロバイダ中に貯められた他のプロバイダからの MCSPDU の数と MCSPDU の順序に依存する。

それにも関わらず、スループット強制は、データ転送が均一に行われる実際の場合において有益なオプションである。MCS プロバイダが指定した時間間隔内において、それぞれの MCS アタッチメントとそれぞれの下方への MCS コネクションに対する MCSPDU の送信は最小数とすることが要求される。各コントロール MCSPDU とデータ MCSPDU の出力は、ドメインパラメータで許容された最大サイズのスループットを考慮して送信する。MCS アタッチメントへの MCSPDU の出力は、関連した指示または確認のプリミティブを配送することを意味する。下方への MCS コネクションへの MCSPDU の出力は、トランスポートサービス インタフェースへデータを渡し、対応した TC へデータを送信することを意味する。

出力は、データのプライオリティとは無関係に、与えられたアタッチメントか下方へのコネクションに対して、一つまたは複数の MCSPDU がキューに貯まっている間、監視される。キューが空になるたびに、何もアクションすること無しに監視を停止する。その後、バッファに空きが無くなり MCSPDU がキューに貯められると、直ちに監視を再開する。いくつかの MCSPDU がキューに貯まっている間、設定された時間間隔内に実際に出力された MCSPDU の数を計数する。

スループット強制のインターバルは、それぞれの MCS プロバイダにより決められる。インターバルは、少なくとも最小スループットにおいて、最大サイズの一つの MCSPDU を出力するのに十分な時間が必要である。MCS プロバイダは EDrq を転送することにより、選択したインターバルとその後の変更を上位へ通知する。プロバイダは、モニタされたスループットが決められた値を越えた時のインターバルの終了時点で、弊害となる MCS アタッチメントまたは下方へのコネクションに対して処理を行う。すなわち、ユーザをデタッチするかコネクションを切断する。

上位のプロバイダは、スループット強制のインターバルに下位のプロバイダのものに、いくらかの処理時間を加えた時間を設定する。その目的は、問題を検出した最も低位のプロバイダに最初にスループット強制の処理を行わせるためである。スループットに弊害をもたらした者が、より低位のプロバイダのインターバルの終了時点で除去されたとき、上位のプロバイダは適当なレベルにスループットが回復するのを認識するだけの時間を待つべきである。もし、十分な待ち時間がないと、必要以上のサブツリーの除去や、必要以上に正常な者を除去することとなってしまう。

ドメイン パラメータを設定する制御アプリケーションは、アプリケーション間で転送されるデータが多い場合にスループットがときどき低下することを考慮して、要求に対してスループットを控えめに見積もるべきである。単に、受信するものを何でも止めてしまうような受信側に対処するためなら、最低のスループットを十分に低く設定すればよい。最大の MCSPDU のサイズと要求されたスループットを知ることにより、コントローラは障害を検出してそれを解消するのに必要な最小のインターバルを見積もることができる。

１２．４　ドメイン コンフィグレーション

TTC標準JT-T122はMCSプロバイダによりサポートされるドメインセットの形成については、そのメカニズムを何も提供しない。これはローカルな問題として考えられるべきであり、その標準化は今後の検討

課題である。このプロトコルは、MCSプロバイダがいくつかのドメイン セレクタが有効であり、また他のドメイン セレクタが無効であることを認知することを前提とする。なお、ドメイン セレクタはMCSコネクションを確立する手段として用いられる。

MCSプロバイダは、ドメインパラメータのネゴシエーションに暗黙に参加する。発側または着側にかかわらず、それが配置されている範囲に従って、許容されるパラメータ値の範囲を制限する。ユーザがドメインにアタッチするか、または、最初のMCSコネクションが確立されたら、MCSプロバイダはドメインパラメータのネゴシエーションを凍結する。

１２．５　ドメイン マージ

ドメインは、MCS-CONNECT-PROVIDERの結果としてマージされる。一方のドメインまたは他方のドメインが空である場合は、マージすることが容易である。しかし、最も一般的な場合では、以前のトッププロバイダの情報ベースを含むように、残りのトッププロバイダにおいて情報ベースをアップデートするための準備と、明白な競合を解決するための準備がされなければならない。この詳細は第10章において説明される。

理解を助けるために、ドメインマージの例を、図12-1/JT-T125から図12-4/JT-T125のシーケンスにおいて説明する。ここでは、プロバイダEは、以前のトップを表しており、中間のプロバイダFとMCSコネクションを確立することにより（太線で図示）、新しいドメインに参加している。このとき、プロバイダEとプロバイダFのうち、どちらがMCSコネクションの確立を起動しても構わない。

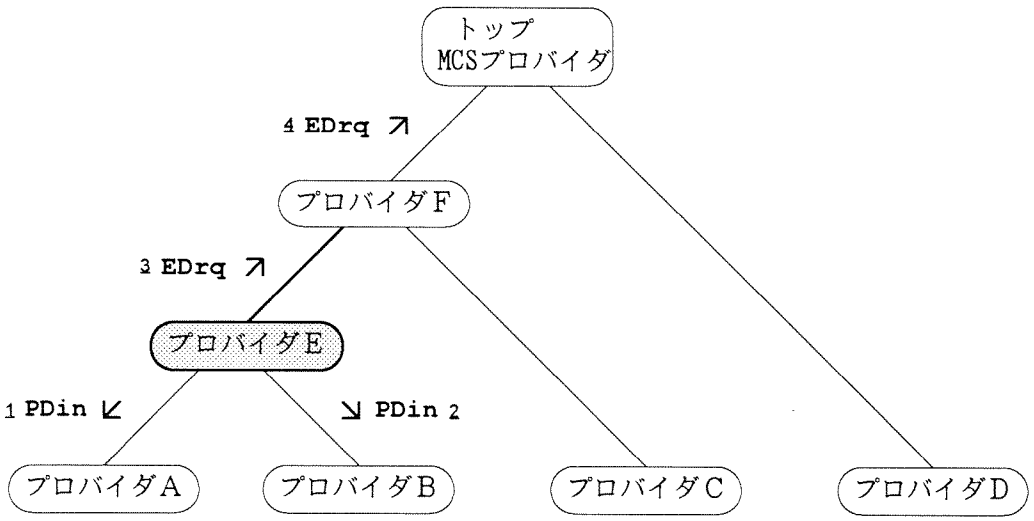


図12-1/JT-T125　　ドメインマージの第１段階：階層構造の確立
(ITU-T T.125)

プロバイダEは、上方へのコネクションの下端に自分自身が位置していることが判ったとき、マージを

実行するための責任持つ。情報ベースの内容がまだ固まらない間は、他の処理を行うのは危険であるため、プロバイダEはそのサブツリーからの入力を受け入れることを停止する。プロバイダAとBから、すでに転送中であるMCSPDU、またはマージが完了する前に生成された新しいMCSPDUは、すべて保護され、後で処理される。しかし、プロバイダEにより生成されたMCSPDU、または更に上位から転送されたMCSPDUは、そのまま下位のプロバイダへ転送される。

マージが完了するまで、上方へのユーザの要求は許されないため、この間にプロバイダEが受信する確認用MCSPDUは、MCcfとMTcfだけである。これらは、すでに使用されているチャネルとトークンIDを確認するか、または削除する。マージが個々に確認された下位のドメインからチャネルとトークンIDを削除するとき、上位のドメインから指示用MCSPDUであるPCin、PTin、DUin、およびCDinが到着する。下位のドメインにおいて未確認であったIDは、削除から保護される。なぜなら、それらは上位のドメインの同じIDとは異なった意味を持っているからである。ドメインの高さを制限するために、PDinに従わなければならない。マージが進行中の間、上記以外の指示用MCSPDUは、プロバイダEによって適用される必要がない。特に、マージが完了するまで、以前の別々なドメインの間でデータは転送される必要がない。少なくとも、データを運んでいるチャネルが整理されるまで、データは流れることができない。チャネルIDを更新するCAinとトークンIDを更新するTVinは、適用するのには不便である。情報ベースの内容が換わらないようにするために、もしプロバイダEがCAinまたはTVinを拒絶するならば、第10章の記述に従った動作を実行する。

プロバイダEは、最新のMCSコネクションが、個々のドメイン階層が1つのトッププロバイダを持つという原則を無効とするようなサイクル（ループ）を形成しないように、最初にPDinを下方へ送信する。また、ドメイン階層の高さとスループット強制インターバル(throughput enforcement interval)を報告するために、EDrqを上方に送信する。そして、プロバイダFは、高さが2から3まで上昇したEDrqをトッププロバイダへと通過させ、EDrqは高さ4に達する。

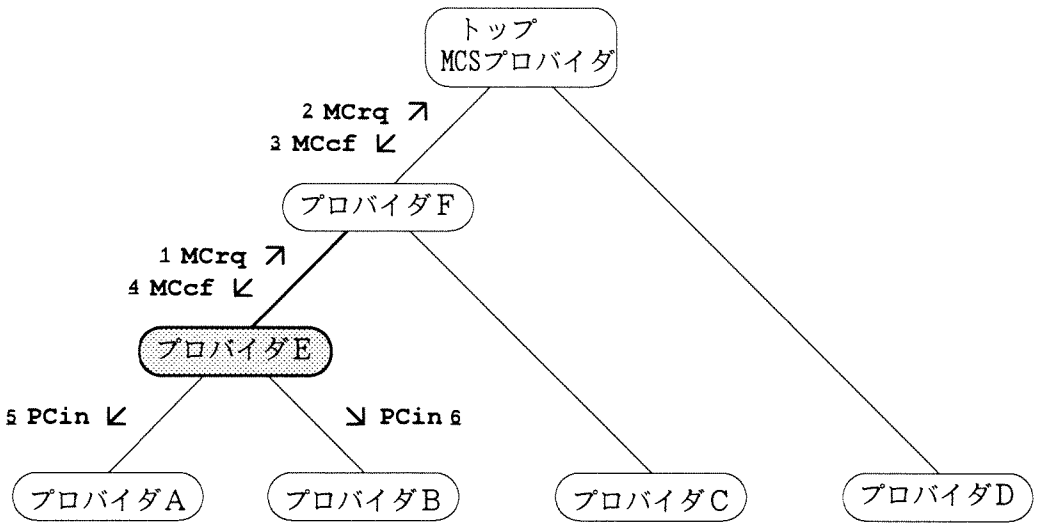


図12-2/JT-T125　　ドメインマージの第2段階：ユーザIDチャネルのマージ

(ITU-T T.125)

ドメインマージの第2段階で、プロバイダEは、その情報ベースのユーザIDと同数のMCrqを上方に送る。上位のドメインと衝突しないユーザIDは確認され、残りは多くのMCcfの中で削除される。プロバイダEは、そのサブツリー全体へこれらの削除を報告するために、MCcfからPCinを生成する。この段階は、すべてのユーザIDが明示的に確認されるか、または削除されたときに終了する。

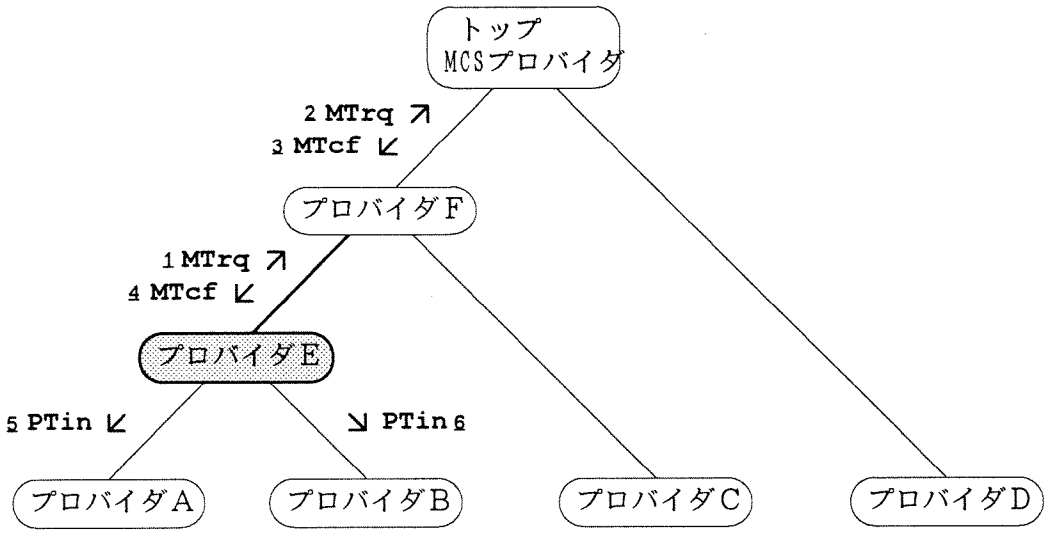


図12-3/JT-T125　　ドメインマージの第３段階：トークンIDのマージ
(ITU-T T.125)

第3段階は、ユーザIDチャネルがトークンIDに置き換わったことを除いて第2段階と似ている。ユーザIDが最初にマージされていなければ、MTrqの一部は無効であるとして拒絶され、また影響されたトークンIDは不必要に削除される。もし、１つのトークンを抑制しているユーザが非常に多数ならば、その属性は単一のMTrqに納まらないかもしれない。この場合、その属性は複数のMCSPDUにより上位へ送信される。プロバイダEは、１番目の要求に対する確認であるMTcfを受信した後に、同じ抑制されたトークンをマージするための２番目以降の要求を、そのパラメータにユーザIDを設定して送信しなければならない。これは、最初の属性のセットが、使用中のトークンIDが多すぎるという理由で拒否されたにも関らず、後から来た残りの属性が受け取られてしまうことで、情報ベースを不正なものにしてしまうということから保護する。この段階は、すべてのトークンIDが明示的に確認されるか、または削除されたときに終了する。

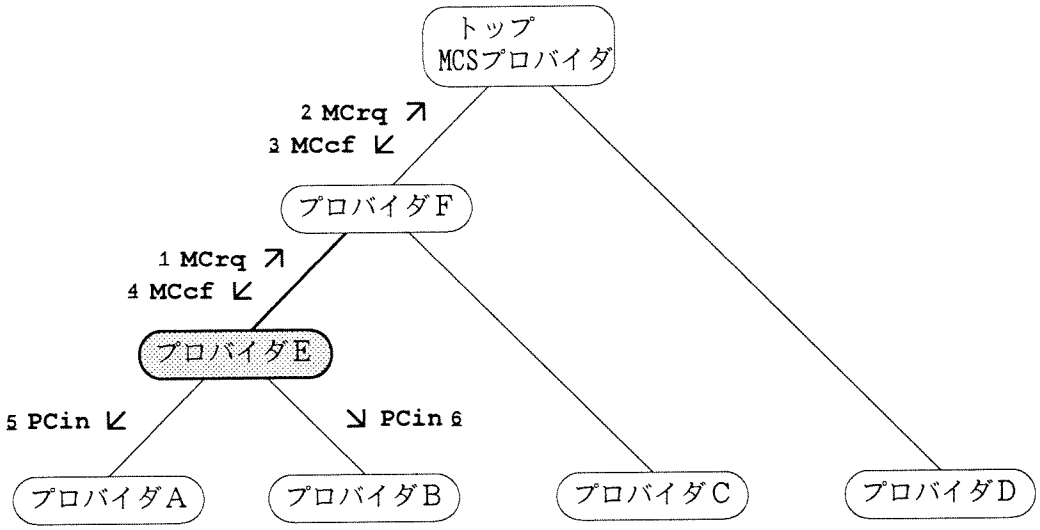


図12-4/JT-T125　　ドメインマージの第４段階：残りのチャネルIDのマージ

(ITU-T T.125)

第4段階は第2段階と同じMCSPDUを含んでいるが、それらは異なるチャネルIDを含んでいる。ユーザIDチャネルで残っているものは、スタティックチャネルID、プライベートチャネルID、割当チャネルIDである。ユーザIDが最初にマージされていなければ、MCrqの一部は無効であるとして拒絶され、また影響されたチャネルIDは不必要に削除される。もし、１つのプライベートチャネルに加入しているユーザが非常に多数ならば、その属性は単一のMCrqに納まらないかもしれない。この場合、その属性は複数のMCSPDUにより上位へ送信される。プロバイダEは、１番目の要求に対する確認であるMCcfを受信した後に、同じチャネルをマージするための２番目以降の要求を、そのパラメータにユーザIDを設定して送信しなければならない。これは、最初の属性のセットが、使用中のチャネルIDが多すぎるという理由で拒否されたにも関らず、後から来た残りの属性が受け取られてしまうことで、情報ベースを不正なものにしてしまうということから保護する。この段階は、すべてのチャネルIDが明示的に確認されるか、または削除されたときに終了する。

（注）最初にトークンIDをマージすることは、トークンの所有を排他的とし、データの流れの衝突を抑えるために有効である。さもなければ、トークンの削除によってデータの流れが衝突を起こし始める前に、チャネルIDが確認されてしまうので、ドメインの間でデータが漏れてしまうかもしれない。

１２．６　ドメイン切断

上方へのMCSコネクションが切断されるとき、MCSプロバイダは、自分の全てのMCSアタッチメントをデタッチするとともに、その他の自分への全MCSコネクションを切断することにより、ドメインの自分のサブツリーを削除する。一般に影響を受けたプロバイダは、自分自身のサブツリー中に残りのドメイン

を設立することはできない。なぜなら、対応する確認用のMCSPDUを決して受信しないような、上方へ送られた要求用のMCSPDUの記録を持っていないからである。

下方へのMCSコネクションが切断されるとき、MCSプロバイダは、サブツリーのその部分に属する全てのユーザに対して「ドメイン切断(domain disconnected)」なる理由のDUrq MCSPDUを生成する。

１２．７　チャネルＩＤ割り当て

MCS-ATTACH-USERとMCS-CHANNEL-JOIN（チャネル番号０）とMCS-CHANNEL-CONVENEのプリミティブ要求のプロセスの間に、1001以上の範囲のチャネルIDが、トップMCSプロバイダでダイナミックに割り当てられる。割り当てられた値は、いかなる特別のパターンにも適合する必要はない。むしろ、許容範囲内に値がランダムに散らされることが望ましい。そのことが、ある期間独立して働いていた２つのドメインが、後でチャネルIDの各々の割り当てに関して衝突することなしにマージされることを容易にさせる。それはまた、解放されたIDが他の目的のために再割り当てされるとき、単一のドメイン中でIDがあまりにも速く再利用されることを防止する。MCSを使用するアプリケーションは、チャネルまたはユーザIDの消滅に対応するための時間が必要であろう。

動作中のドメインの継目のないマージが要求される状況において、各々のプロバイダに固有なサブレンジからチャネルIDを選択することにより、チャネルIDの割り当てに関する対立は避けられる。1001～65535に割り当てられたチャネルIDが、幾つかの部分に動的に分割されることにより、サブレンジが作成される。さらに、サブレンジの範囲内で、IDは トップまたはボトムから１つの方向に連続的に割り当てられる。サブレンジ割り当てを使用するプロバイダは、やはり、ドメイン マージの手順を含んだ、このプロトコルの全ての面に従う。それらのプロバイダは、それによって固有のサブレンジを持たない同位のプロバイダと調停する。

（注）プロバイダが、サブレンジの割り当てをどのように選ぶかは、ローカルな実装上の問題である。前もって準備された会議は、ネットワーク管理システムによって結合される。

トークンIDは、チャネルIDとは違って、割り当てられるものではなく、また、１００１未満／以上での意味の違いもない。未使用のトークンIDは、いつでも獲得または抑制のために静的に利用できる。ただし、同時に用いられる全ての数の上でのドメイン限界を必要とする。

１２．８　トークン ステータス

トークン ステータスは、第７章で正式に定義される。それは、トークン確認のMCSPDUの構成要素として返される。また、下位プロバイダの情報ベース中のトークンIDの記録を更新することに利用される。トークン ステータスは、確認プリミティブを使用して、要求を出したユーザに直接伝えられる必要はない。しかし、結果の値を使用して間接的に伝えられるかもしれない。

１つ以上のトークン ステータス値が、明記されたトークンIDへ与えられたユーザの関係を記述するとき、優先順位は次に述べる通りである。それらがMCS-TOKEN-GIVE-指示に応答しなければならないことをユーザに注意するために、『self-recipient』が最初に報告される。次に好ましいのは、未完了な操作をしないことをユーザに注意するために、『self-giving』であり、さらに『self-grabbed』か『self-inhibited』が続く。最後は、他の一団によって単独利用された結果としての、トークンの現在の状態を反映する残りのステータス値である。

MCSプロバイダは、その情報ベース中でトークン状態を正しく更新するためには、トークン ステータス値のために明記された優先権を用いる。

１３．実装のための参照

付録２から付録７は、MCSプロバイダの動作をSDLで示している。それらは、記述されたプロトコルが適度な努力で実現されえることを証明している。この例は、実装者が同じような論理を考える手助けをし、互換性のあるシステムの導入をはかどらせる。

SDLは状態遷移表より進んだ公式な記述技法であり、MCSの複雑さにより適している。これは、テキストとグラフィックの２つの表現を持つ。付録のSDLはテキストであり、従来のプログラミング言語に翻訳しやすい。SDLの参考書として以下のものがある。

・ＣＣＩＴＴ勧告Ｚ.１００（１９８８），Specification and Description Language(SDL)

・Belina,Hogrefe,and Sarma: SDL with Applications from Protocol Specification,
(Prentice Hall,1991),ISBN 0-13-785890-6

・Belina and Hogrefe: The CCITT Specification and Description Language SDL,
Computer Networks and ISDN Systems,16(1988/89),pages 311-341

付録２は、システムとブロックの構成図より始まっている。図中のプロセス、すなわちコントロール、ドメイン、終端、アタッチメントは、付録３から付録６にそれぞれ定義されている。付録７はモデルの仮定を説明するとともに、主な信号の流れを図示している。

付録を利用することで、この標準に定義されたMCSプロトコルを正確に実装することができる。ただし、必ずしもこの付録を利用する必要はない。矛盾がある場合には、本標準の本体の内容が優先する。

MCSプロトコルは、付録の内容を必要としない別の設計方法により実現されるかもしれない。

付録２から付録７には、汎用的なMCSプロバイダを拡張するためのパラメータが用意されている。特別な場合に限定されている場合には、もっとコンパクトで効果的な方法があるかもしれない。この特別な場合の例として、MCSプロバイダが、接続するMCSコネクションを１つのみに制限され、端末ノードに当てられた場合がある。これは、下位のドメインにおいて使用されたIDを消去することによって、合併を容易にするために、さらに特殊化されるかもしれない。特別な場合におけるSDLは、今後の検討課題である。

付録１　ＭＣＳＰＤＵの符号化

（ＪＴ－Ｔ１２５に対する）

１．データ送信要求

第７章から抜粋した定義を以下に示す。

```
DomainMCSPDU        ::= CHOICE
{
        ...
        sdrq                    SDrq,
        ...
}

SDrq        ::= [APPLICATION 25] IMPLICIT SEQUENCE
{
        initiator               UserId,
        channelId               ChannelId,
        dataPriority            DataPriority,
        segmentation            Segmentation,
        userData                OCTET STRING
}

UserId              ::= DynamicChannelId

DynamicChannelId::= ChannelId (1001..65535)

ChannelId           ::= INTEGER (0..65535)

DataPriority        ::= ENUMERATED
{
                        top             (0),
                        high            (1),
                        medium          (2),
                        low             (3)
}

Segmentation        ::= BIT STRING
{
                        begin           (0),
                        end             (1)
} (SIZE (2))
```

SDrq の例を以下に示す。

```
sdrq
{
        initiator               1701,
        channelId                5,
        dataPriority            high,
        segmentation             {begin},
        userData                 '4D4353'H
}
```

2．ベーシック符号化規則

ベーシック符号化規則は、識別子、長さ、内容から構成される。

SDrq	Length	Contents
79	13	

INTEGER	Length	Contents
02	02	06 A5

INTEGER	Length	Contents
02	01	05

ENUMERATED	Length	Contents
0A	01	01

BIT STRING	Length	Contents
03	02	06 80

OCTET STRING	Length	Contents
04	03	4D 43 53

ユーザデータを除いたヘッダのオクテット数は、MCSPDUタグ、チャネルID、ユーザデータ長により、１８～２４である。

3．　パックド符号化規則

パックド符号化規則は、タグが伝送されないため、符号化規則として、ＡＳＮ.１が含まれていることが予め知られている場合のみデコードされる。これは、Domain MCSPDU の結合されたタイプを定義するための１つの理由である。Domain MCSPDU のアプリケーションタグは、０から昇順に割り当てられる。パックド符号化規則は、基準値からのオフセット値を符号化する。ベーシック符号化規則とパックド符号

化規則のいずれも、２５はSDrqを示す。

CHOICE -- 6 bits + pad
64

INTEGER (1001..65535) -- offset 1001
02 BC

INTEGER (0..65535) -- offset 0
00 05

ENUMERATED + BIT STRING (SIZE (2)) -- 2 bits + 2 bits + pad
60

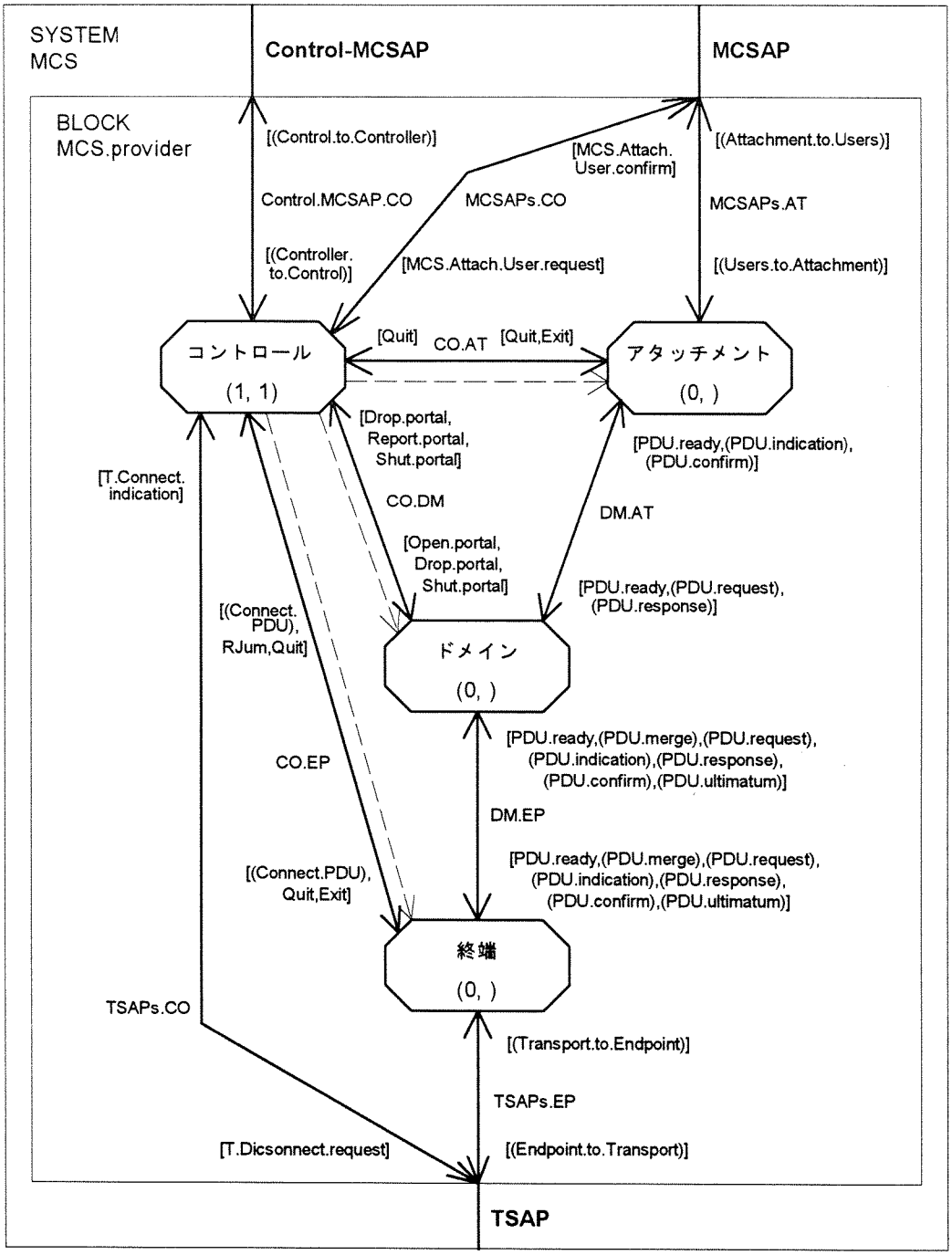
OCTET STRING -- length + contents
03 4D 43 53

ユーザデータを除いたヘッダのオクテット数は、ユーザデータ長により、７～８である。

パックド符号化規則は、１２７オクテットまでのストリング長を１オクテットで、また、１６３８４オクテットまでのストリング長を２オクテットで符号化する。

付録２　ＭＣＳプロバイダのＳＤＬ図

（ＪＴ－Ｔ１２５に対する）



付図 2-1/JT-T125　MCS プロバイダの SDL 図

(ITU-T T.125)

```
1   SYSTEM MCS;

3   SYNONYM oneSecond          Duration = 1000;    /* time is in milliseconds */

5         /* Type definitions */

7   SYNTYPE        ChannelId      = Integer CONSTANTS 0:65535
8   ENDSYNTYPE;

10  NEWTYPE        UserId         INHERITS ChannelId
11  OPERATORS ALL;
12  ADDING
13  OPERATORS
14        UserId:    ChannelId    -> UserId;      /* type cast */
15        ChannelId: UserId       -> ChannelId;   /* type cast */
16  AXIOMS
17        UserId(0) = 0;
18        FOR ALL c in ChannelId
19        (
20              ChannelId(UserId(c)) = c;
21        );
22  ENDNEWTYPE;

24  SYNTYPE        TokenId        = Integer CONSTANTS 1:65535
25  ENDSYNTYPE;

27  NEWTYPE        ChannelIdSet    SetOf(ChannelId);
28  ENDNEWTYPE;
29  NEWTYPE        UserIdSet       SetOf(UserId);
30  ENDNEWTYPE;
31  NEWTYPE        TokenIdSet      SetOf(TokenId);
32  ENDNEWTYPE;

34  NEWTYPE        TokenStatus
35  LITERALS
36        NotInUse,
37        SelfGrabbed,
38        OtherGrabbed,
39        SelfInhibited,
40        OtherInhibited,
41        SelfRecipient,
42        SelfGiving,
43        OtherGiving;
44  ENDNEWTYPE;

46  SYNTYPE        DataPriority    = Integer CONSTANTS 0:3
47  ENDSYNTYPE;

49  NEWTYPE        Segmentation
50  STRUCT
51        begin                Boolean;
52        end                  Boolean;
53  ENDNEWTYPE;

55  NEWTYPE        DomainParameters
56  STRUCT
57        maxChannelIds         Natural;
58        maxUserIds            Natural;
59        maxTokenIds           Natural;
60        numPriorities         Natural;
61        minThroughput         Natural;
62        maxHeight             Natural;
63        maxMCSPDUsize         Natural;
64        protocolVersion       Natural;
65  ENDNEWTYPE;

67  SYNTYPE        DomainSelector  = OctetString
68  ENDSYNTYPE;

70  SYNTYPE        TSAPAddress     = OctetString
71  ENDSYNTYPE;

73  NEWTYPE        TransportQOS          /* quality of service */
74  STRUCT
75        throughput            Natural;       /* octets per second */
76        transitDelay          Duration;      /* one-way */
77        dataPriority          Natural;       /* 0 is highest */
78  ENDNEWTYPE;
```

```
NEWTYPE        TransportQOSByPri      Array(DataPriority, TransportQOS);
ENDNEWTYPE;

SYNTYPE        UserData       = OctetString
ENDSYNTYPE;

SYNTYPE        TSDU           = OctetString
ENDSYNTYPE;

SYNTYPE        Octet          = Integer CONSTANTS 0:255
ENDSYNTYPE;

NEWTYPE        OctetString    String(Octet, NullString);
ENDNEWTYPE;

GENERATOR      SetOf (TYPE ItemType)   /* subsets with choice operator */
/*AS*/
       Powerset(ItemType);
ADDING
OPERATORS
       Pick:   SetOf   -> ItemType;    /* chooses any element */
AXIOMS
       Pick(Empty) == ERROR!;
       FOR ALL s IN SetOf
       (
              s /= Empty ==> Pick(s) in s;
       );
DEFAULT
       Empty;
ENDGENERATOR;

NEWTYPE        Reason
LITERALS
       RN_domain_disconnected,
       RN_provider_initiated,
       RN_token_purged,
       RN_user_requested,
       RN_channel_purged,
       RN_channel_disbanded,          /* not in MCSPDUs */
       RN_domain_not_hierarchical,     /* not in MCSPDUs */
       RN_parameters_unacceptable,     /* not in MCSPDUs */
       RN_unspecified;                /* not in MCSPDUs */
ENDNEWTYPE;

NEWTYPE        Result
LITERALS
       RT_successful,
       RT_domain_merging,
       RT_domain_not_hierarchical,
       RT_no_such_channel,
       RT_no_such_domain,
       RT_no_such_user,
       RT_not_admitted,
       RT_other_user_id,
       RT_parameters_unacceptable,
       RT_token_not_available,
       RT_token_not_possessed,
       RT_too_many_channels,
       RT_too_many_tokens,
       RT_too_many_users,
       RT_unspecified_failure,
       RT_user_rejected,
       RT_congested,                  /* not in MCSPDUs */
       RT_domain_disconnected;         /* not in MCSPDUs */
ENDNEWTYPE;

       /* The next three identifier types distinguish separate instances
       of communication across the interface between an MCS provider and
       its environment.  MCSConnectionId maps to some resource within the
       Control process.  MCSAttachmentId, which equals the process id of
       an Attachment process, could be considered implicit, since it is
       the source or destination address of corresponding signals, but
       the usage is clearer when it is made an explicit signal parameter.
       The same model is assumed for TCEndpointId. */

SYNTYPE        MCSConnectionId = Natural
ENDSYNTYPE;

SYNTYPE        MCSAttachmentId = PId
```

```
159   ENDSYNTYPE;

161   SYNTYPE        TCEndpointId    = PId
162   ENDSYNTYPE;

164         /* Block decomposition */

166   BLOCK  MCS.provider     REFERENCED;

168   CHANNEL Control.MCSAP
169         FROM ENV TO MCS.provider WITH
170               (Controller.to.Control);
171         FROM MCS.provider TO ENV WITH
172               (Control.to.Controller);
173   ENDCHANNEL;

175   SIGNALLIST Controller.to.Control =
176         MCS.Connect.Provider.request,
177         MCS.Connect.Provider.response,
178         MCS.Disconnect.Provider.request;

180   SIGNALLIST Control.to.Controller =
181         MCS.Connect.Provider.indication,
182         MCS.Connect.Provider.confirm,
183         MCS.Disconnect.Provider.indication;

185   SIGNAL  MCS.Connect.Provider.request
186         (
187               Natural,              /* requester's label */
188               TSAPAddress,           /* calling */
189               DomainSelector,
190               TSAPAddress,           /* called */
191               DomainSelector,
192               Boolean,              /* upward */
193               DomainParameters,      /* target */
194               DomainParameters,      /* minimum */
195               DomainParameters,      /* maximum */
196               TransportQOSByPri,     /* target */
197               TransportQOSByPri,     /* minimum */
198               UserData
199         );
200   SIGNAL  MCS.Connect.Provider.indication
201         (
202               MCSConnectionId,       /* provider-assigned */
203               TSAPAddress,          /* calling */
204               DomainSelector,
205               TSAPAddress,           /* called */
206               DomainSelector,
207               Boolean,              /* upward */
208               DomainParameters,      /* target */
209               DomainParameters,      /* minimum */
210               DomainParameters,      /* maximum */
211               UserData
212         );
213   SIGNAL  MCS.Connect.Provider.response
214         (
215               MCSConnectionId,
216               Result,
217               DomainParameters,
218               UserData
219         );
220   SIGNAL  MCS.Connect.Provider.confirm
221         (
222               Natural,              /* requester's label */
223               Result,
224               MCSConnectionId,       /* provider-assigned */
225               DomainParameters,
226               UserData
227         );

229   SIGNAL  MCS.Disconnect.Provider.request
230         (
231               MCSConnectionId
232         );
233   SIGNAL  MCS.Disconnect.Provider.indication
234         (
235               MCSConnectionId,
236               Reason
237         );
```

```
239   CHANNEL MCSAPs
240         FROM ENV TO MCS.provider WITH
241               MCS.Attach.User.request,
242               (Users.to.Attachment);
243         FROM MCS.provider TO ENV WITH
244               (Attachment.to.Users);
245   ENDCHANNEL;

247   SIGNALLIST Users.to.Attachment =
248         MCS.ready,
249         MCS.Detach.User.request,
250         MCS.Channel.Join.request,
251         MCS.Channel.Leave.request,
252         MCS.Channel.Convene.request,
253         MCS.Channel.Disband.request,
254         MCS.Channel.Admit.request,
255         MCS.Channel.Expel.request,
256         MCS.Send.Data.request,
257         MCS.Uniform.Send.Data.request,
258         MCS.Token.Grab.request,
259         MCS.Token.Inhibit.request,
260         MCS.Token.Give.request,
261         MCS.Token.Give.response,
262         MCS.Token.Please.request,
263         MCS.Token.Release.request,
264         MCS.Token.Test.request;

266   SIGNALLIST Attachment.to.Users =
267         MCS.ready,
268         MCS.Attach.User.confirm,
269         MCS.Detach.User.indication,
270         MCS.Channel.Join.confirm,
271         MCS.Channel.Leave.indication,
272         MCS.Channel.Convene.confirm,
273         MCS.Channel.Disband.indication,
274         MCS.Channel.Admit.indication,
275         MCS.Channel.Expel.indication,
276         MCS.Send.Data.indication,
277         MCS.Uniform.Send.Data.indication,
278         MCS.Token.Grab.confirm,
279         MCS.Token.Inhibit.confirm,
280         MCS.Token.Give.indication,
281         MCS.Token.Give.confirm,
282         MCS.Token.Please.indication,
283         MCS.Token.Release.confirm,
284         MCS.Token.Test.confirm;

286   SIGNAL  MCS.ready                    /* allows one MCS.[Uniform.]Send.Data */
287         (
288               MCSAttachmentId,
289               DataPriority
290         );

292   SIGNAL  MCS.Attach.User.request
293         (
294               Natural,              /* requester's label */
295               DomainSelector
296         );
297   SIGNAL  MCS.Attach.User.confirm
298         (
299               Natural,              /* requester's label */
300               Result,
301               MCSAttachmentId,       /* provider-assigned */
302               UserId
303         );

305   SIGNAL  MCS.Detach.User.request
306         (
307               MCSAttachmentId
308         );
309   SIGNAL  MCS.Detach.User.indication
310         (
311               MCSAttachmentId,
312               UserId,
313               Reason
314         );

316   SIGNAL  MCS.Channel.Join.request
```

```
317         (
318                MCSAttachmentId,
319                ChannelId
320         );
321   SIGNAL  MCS.Channel.Join.confirm
322         (
323                MCSAttachmentId,
324                ChannelId,            /* requested */
325                Result,
326                ChannelId
327         );

329   SIGNAL  MCS.Channel.Leave.request
330         (
331                MCSAttachmentId,
332                ChannelId
333         );
334   SIGNAL  MCS.Channel.Leave.indication
335         (
336                MCSAttachmentId,
337                ChannelId,
338                Reason
339         );

341   SIGNAL  MCS.Channel.Convene.request
342         (
343                MCSAttachmentId
344         );
345   SIGNAL  MCS.Channel.Convene.confirm
346         (
347                MCSAttachmentId,
348                Result,
349                ChannelId
350         );

352   SIGNAL  MCS.Channel.Disband.request
353         (
354                MCSAttachmentId,
355                ChannelId
356         );
357   SIGNAL  MCS.Channel.Disband.indication
358         (
359                MCSAttachmentId,
360                ChannelId,
361                Reason
362         );

364   SIGNAL  MCS.Channel.Admit.request
365         (
366                MCSAttachmentId,
367                ChannelId,
368                UserIdSet
369         );
370   SIGNAL  MCS.Channel.Admit.indication
371         (
372                MCSAttachmentId,
373                ChannelId,
374                UserId
375         );

377   SIGNAL  MCS.Channel.Expel.request
378         (
379                MCSAttachmentId,
380                ChannelId,
381                UserIdSet
382         );
383   SIGNAL  MCS.Channel.Expel.indication
384         (
385                MCSAttachmentId,
386                ChannelId,
387                Reason
388         );

390   SIGNAL  MCS.Send.Data.request
391         (
392                MCSAttachmentId,
393                ChannelId,
394                DataPriority,
395                Segmentation,
```

```
396                 UserData
397           );
398     SIGNAL  MCS.Send.Data.indication
399           (
400                 MCSAttachmentId,
401                 ChannelId,
402                 DataPriority,
403                 UserId,
404                 Segmentation,
405                 UserData
406           );

408     SIGNAL  MCS.Uniform.Send.Data.request
409           (
410                 MCSAttachmentId,
411                 ChannelId,
412                 DataPriority,
413                 Segmentation,
414                 UserData
415           );
416     SIGNAL  MCS.Uniform.Send.Data.indication
417           (
418                 MCSAttachmentId,
419                 ChannelId,
420                 DataPriority,
421                 UserId,
422                 Segmentation,
423                 UserData
424           );

426     SIGNAL  MCS.Token.Grab.request
427           (
428                 MCSAttachmentId,
429                 TokenId
430           );
431     SIGNAL  MCS.Token.Grab.confirm
432           (
433                 MCSAttachmentId,
434                 TokenId,
435                 Result
436           );

438     SIGNAL  MCS.Token.Inhibit.request
439           (
440                 MCSAttachmentId,
441                 TokenId
442           );
443     SIGNAL  MCS.Token.Inhibit.confirm
444           (
445                 MCSAttachmentId,
446                 TokenId,
447                 Result
448           );

450     SIGNAL  MCS.Token.Give.request
451           (
452                 MCSAttachmentId,
453                 TokenId,
454                 UserId
455           );
456     SIGNAL  MCS.Token.Give.indication
457           (
458                 MCSAttachmentId,
459                 TokenId,
460                 UserId
461           );
462     SIGNAL  MCS.Token.Give.response
463           (
464                 MCSAttachmentId,
465                 TokenId,
466                 Result
467           );
468     SIGNAL  MCS.Token.Give.confirm
469           (
470                 MCSAttachmentId,
471                 TokenId,
472                 Result
473           );
```

```
475   SIGNAL  MCS.Token.Please.request
476         (
477               MCSAttachmentId,
478               TokenId
479         );
480   SIGNAL  MCS.Token.Please.indication
481         (
482               MCSAttachmentId,
483               TokenId,
484               UserId
485         );

487   SIGNAL  MCS.Token.Release.request
488         (
489               MCSAttachmentId,
490               TokenId
491         );
492   SIGNAL  MCS.Token.Release.confirm
493         (
494               MCSAttachmentId,
495               TokenId,
496               Result
497         );

499   SIGNAL  MCS.Token.Test.request
500         (
501               MCSAttachmentId,
502               TokenId
503         );
504   SIGNAL  MCS.Token.Test.confirm
505         (
506               MCSAttachmentId,
507               TokenId,
508               TokenStatus
509         );

511   CHANNEL TSAPs
512         FROM MCS.provider TO ENV WITH
513               (Endpoint.to.Transport);
514         FROM ENV TO MCS.provider WITH
515               T.Connect.indication,
516               (Transport.to.Endpoint);
517   ENDCHANNEL;

519   SIGNALLIST Endpoint.to.Transport =
520         T.ready,
521         T.Connect.request,
522         T.Connect.response,
523         T.Data.request,
524         T.Disconnect.request;

526   SIGNALLIST Transport.to.Endpoint =
527         T.ready,
528         T.Connect.confirm,
529         T.Data.indication,
530         T.Disconnect.indication;

532   SIGNAL  T.ready                     /* allows one T.Data */
533         (
534               TCEndpointId
535         );

537   SIGNAL  T.Connect.request
538         (
539               Natural,              /* requester's label */
540               TSAPAddress,          /* calling */
541               TSAPAddress,          /* called */
542               TransportQOS,         /* target */
543               TransportQOS          /* minimum */
544         );
545   SIGNAL  T.Connect.indication
546         (
547               TCEndpointId,         /* provider-assigned */
548               TSAPAddress,          /* calling */
549               TSAPAddress,          /* called */
550               TransportQOS,         /* offered */
551               TransportQOS          /* minimum */
552         );
553   SIGNAL  T.Connect.response
```

```
554         (
555               TCEndpointId,
556               TransportQOS          /* selected */
557         );
558   SIGNAL  T.Connect.confirm
559         (
560               Natural,              /* requester's label */
561               TCEndpointId,         /* provider-assigned */
562               TransportQOS          /* selected */
563         );

565   SIGNAL  T.Data.request
566         (
567               TCEndpointId,
568               TSDU
569         );
570   SIGNAL  T.Data.indication
571         (
572               TCEndpointId,
573               TSDU
574         );

576   SIGNAL  T.Disconnect.request
577         (
578               TCEndpointId
579         );
580   SIGNAL  T.Disconnect.indication
581         (
582               Natural,              /* requester's label */
583               TCEndpointId          /* provider-assigned */
584         );

586   ENDSYSTEM;


589   BLOCK  MCS.provider;

591   SYNONYM maxPortalIds    Natural = EXTERNAL;     /* an implementation limit */

593         /* Data type definitions */

595   NEWTYPE         PDUKind         /* domain MCSPDUs */
596   LITERALS
597         PDin,           /* plumb domain indication */
598         EDrq,           /* erect domain request */
599         MCrq,           /* merge channels request */
600         MCcf,           /* merge channels confirm */
601         PCin,           /* purge channels indication */
602         MTrq,           /* merge tokens request */
603         MTcf,           /* merge tokens confirm */
604         PTin,           /* purge tokens indication */
605         DPum,           /* disconnect provider ultimatum */
606         RJum,           /* reject MCSPDU ultimatum */
607         AUrq,           /* attach user request */
608         AUcf,           /* attach user confirm */
609         DUrq,           /* detach user request */
610         DUin,           /* detach user confirm */
611         CJrq,           /* channel join request */
612         CJcf,           /* channel join confirm */
613         CLrq,           /* channel leave request */
614         CCrq,           /* channel convene request */
615         CCcf,           /* channel convene confirm */
616         CDrq,           /* channel disband request */
617         CDin,           /* channel disband confirm */
618         CArq,           /* channel admit request */
619         CAin,           /* channel admit confirm */
620         CErq,           /* channel expel request */
621         CEin,           /* channel expel confirm */
622         SDrq,           /* send data request */
623         SDin,           /* send data indication */
624         USrq,           /* uniform send data request */
625         USin,           /* uniform send data indication */
626         TGrq,           /* token grab request */
627         TGcf,           /* token grab confirm */
628         TIrq,           /* token inhibit request */
629         TIcf,           /* token inhibit confirm */
630         TVrq,           /* token give request */
631         TVin,           /* token give indication */
632         TVrs,           /* token give response */
```

```
633         TVcf,          /* token give confirm */
634         TPrq,          /* token please request */
635         TPin,          /* token please indication */
636         TRrq,          /* token release request */
637         TRcf,          /* token release confirm */
638         TTrq,          /* token test request */
639         TTcf;          /* token test confirm */
640   ENDNEWTYPE;

642   NEWTYPE        PDUStruct
643   STRUCT
644         kind                 PDUKind;
645         /* fields used depend on kind */
646         channelId            ChannelId;
647         channelIds           ChannelIdSet;
648         dataPriority         DataPriority;
649         detachUserIds        UserIdSet;
650         diagnostic           Diagnostic;
651         heightLimit          Natural;
652         initialOctets        OctetString;
653         initiator            UserId;
654         mergeChannels        ChannelAttributesSet;
655         mergeTokens          TokenAttributesSet;
656         purgeChannelIds      ChannelIdSet;
657         purgeTokenIds        TokenIdSet;
658         reason               Reason;
659         recipient            UserId;
660         requested            ChannelId;
661         result               Result;
662         segmentation         Segmentation;
663         subHeight            Natural;
664         subInterval          Duration;
665         tokenId              TokenId;
666         tokenStatus          TokenStatus;
667         userData             UserData;
668         userIds              UserIdSet;
669   ENDNEWTYPE;

671   NEWTYPE        ChannelKind
672   LITERALS
673         Static,        /* range 1:1000 = static: known permanently */
674         UserId,        /* dynamic: Attach-User / Detach-User */
675         Private,       /* dynamic: Channel-Convene / Channel-Disband */
676         Assigned;      /* dynamic: Channel-Join zero / last Channel-Leave */
677   ENDNEWTYPE;

679   NEWTYPE        ChannelAttributes
680   STRUCT
681         channelId      ChannelId;     /* the channel with these attributes */
682         kind           ChannelKind;   /* (Static,UserId,Private,Assigned) */
683         manager        UserId;        /* if (Private): the channel manager */
684         admitted       UserIdSet;     /* if (Private): zero or more users */
685         joined         Boolean;       /* if (UserId,Private): True if joined */
686   ENDNEWTYPE;

688   NEWTYPE        ChannelAttributesSet   SetOf(ChannelAttributes);
689   ENDNEWTYPE;

691   NEWTYPE        TokenKind
692   LITERALS
693         Grabbed,       /* assigned exclusively to one user */
694         Inhibited,     /* inhibited by one or more users */
695         Giving,        /* reassigning grabbed to a new user */
696         Ungivable,     /* the recipient has since detached */
697         Given;         /* donor released token or detached */
698   ENDNEWTYPE;

700   NEWTYPE        TokenAttributes
701   STRUCT
702         tokenId        TokenId;       /* the token with these attributes */
703         kind           TokenKind;     /* (Grabbed,Inhibited,Giving,Ungivable,Given)
*/
704         grabber        UserId;        /* if (Grabbed,Giving,Ungivable): user */
705         recipient      UserId;        /* if (Giving,Given): an intended user */
706         inhibitors     UserIdSet;     /* if (Inhibited): one or more users */
707   ENDNEWTYPE;

709   NEWTYPE        TokenAttributesSet     SetOf(TokenAttributes);
710   ENDNEWTYPE;
```

```
712   SYNTYPE        PortalId       = Integer CONSTANTS 0:maxPortalIds
713   ENDSYNTYPE;

715   NEWTYPE        PortalIdSet    SetOf(PortalId);
716   ENDNEWTYPE;

718   NEWTYPE        PortalKind
719   LITERALS
720         Attached,      /* MCS Attachment through an MCSAP */
721         Downlink,      /* MCS Connection to a provider below */
722         Uplink;        /* MCS Connection to a provider above */
723   ENDNEWTYPE;

725   NEWTYPE        PIdByPri       Array(DataPriority, PId);
726   ENDNEWTYPE;

728   NEWTYPE        Diagnostic
729   LITERALS
730         DC_inconsistent_merge,
731         DC_forbidden_PDU_downward,
732         DC_forbidden_PDU_upward,
733         DC_invalid_BER_encoding,
734         DC_invalid_PER_encoding,
735         DC_misrouted_user,
736         DC_unrequested_confirm,
737         DC_wrong_transport_priority,
738         DC_channel_id_conflict,
739         DC_token_id_conflict,
740         DC_not_user_id_channel,
741         DC_too_many_channels,
742         DC_too_many_tokens,
743         DC_too_many_users,
744         DC_OK,                        /* not in MCSPDUs */
745         DC_ignore,                    /* not in MCSPDUs */
746         DC_height_limit_exceeded,     /* not in MCSPDUs */
747         DC_throughput_inadequate;     /* not in MCSPDUs */
748   ENDNEWTYPE;

750         /* Process decomposition */

752   PROCESS  Control       (1,1) REFERENCED;      /* CO */
753   PROCESS  Attachment    (0,)  REFERENCED;      /* AT */
754   PROCESS  Domain        (0,)  REFERENCED;      /* DM */
755   PROCESS  Endpoint      (0,)  REFERENCED;      /* EP */

757   CONNECT Control.MCSAP AND Control.MCSAP.CO;

759   SIGNALROUTE Control.MCSAP.CO
760         FROM ENV TO Control WITH
761               (Controller.to.Control);
762         FROM Control TO ENV WITH
763               (Control.to.Controller);

765   CONNECT MCSAPs AND MCSAPs.CO, MCSAPs.AT;

767   SIGNALROUTE MCSAPs.CO
768         FROM ENV TO Control WITH
769               MCS.Attach.User.request;
770         FROM Control TO ENV WITH
771               MCS.Attach.User.confirm;

773   SIGNALROUTE MCSAPs.AT
774         FROM ENV TO Attachment WITH
775               (Users.to.Attachment);
776         FROM Attachment TO ENV WITH
777               (Attachment.to.Users);

779   CONNECT TSAPs AND TSAPs.CO, TSAPs.EP;

781   SIGNALROUTE TSAPs.CO
782         FROM ENV TO Control WITH
783               T.Connect.indication;
784         FROM Control TO ENV WITH
785               T.Disconnect.request;

787   SIGNALROUTE TSAPs.EP
788         FROM ENV TO Endpoint WITH
789               (Transport.to.Endpoint);
```

```
790         FROM Endpoint TO ENV WITH
791                 (Endpoint.to.Transport);

793     SIGNALROUTE CO.AT
794         FROM Control TO Attachment WITH
795                 Quit,
796                 Exit;
797         FROM Attachment TO Control WITH
798                 Quit;

800     SIGNALROUTE CO.DM
801         FROM Control TO Domain WITH
802                 Open.portal,
803                 Drop.portal,
804                 Shut.portal;
805         FROM Domain TO Control WITH
806                 Drop.portal,
807                 Report.portal,
808                 Shut.portal;

810     SIGNALROUTE CO.EP
811         FROM Control TO Endpoint WITH
812                 (Connect.PDU),
813                 Quit,
814                 Exit;
815         FROM Endpoint TO Control WITH
816                 (Connect.PDU),
817                 RJum,
818                 Quit;

820     SIGNALROUTE DM.AT
821         FROM Domain TO Attachment WITH
822                 PDU.ready,
823                 (PDU.indication),
824                 (PDU.confirm);
825         FROM Attachment TO Domain WITH
826                 PDU.ready,
827                 (PDU.request),
828                 (PDU.response);

830     SIGNALROUTE DM.EP
831         FROM Domain TO Endpoint WITH
832                 PDU.ready,
833                 (PDU.merge),
834                 (PDU.request),
835                 (PDU.indication),
836                 (PDU.response),
837                 (PDU.confirm),
838                 (PDU.ultimatum);
839         FROM Endpoint TO Domain WITH
840                 PDU.ready,
841                 (PDU.merge),
842                 (PDU.request),
843                 (PDU.indication),
844                 (PDU.response),
845                 (PDU.confirm),
846                 (PDU.ultimatum);

848     SIGNAL  Open.portal
849         (
850                 PortalId,
851                 PortalKind,
852                 PIdByPri
853         );

855     SIGNAL  Report.portal
856         (
857                 PortalId,
858                 Diagnostic
859         );

861     SIGNAL  Drop.portal
862         (
863                 PortalId,
864                 Reason
865         );

867     SIGNAL  Shut.portal
868         (
```

```
869                 PortalId
870         );

872     SIGNAL  Quit;

874     SIGNAL  Exit;

876     SIGNALLIST Connect.PDU =
877         Connect.Initial,
878         Connect.Response,
879         Connect.Additional,
880         Connect.Result;

882     SIGNAL  Connect.Initial
883         (
884                 DomainSelector,         /* calling */
885                 DomainSelector,         /* called */
886                 Boolean,               /* upward */
887                 DomainParameters,       /* target */
888                 DomainParameters,       /* minimum */
889                 DomainParameters,       /* maximum */
890                 UserData
891         );
892     SIGNAL  Connect.Response
893         (
894                 Result,
895                 Natural,
896                 DomainParameters,
897                 UserData
898         );
899     SIGNAL  Connect.Additional
900         (
901                 Natural,
902                 DataPriority
903         );
904     SIGNAL  Connect.Result
905         (
906                 Result
907         );

909     SIGNALLIST PDU.merge =
910         PDin, EDrq, MCrq, MCcf, MTrq,
911         MTcf;

913     SIGNALLIST PDU.request =
914         AUrq, DUrq, CJrq, CLrq, CCrq,
915         CDrq, CArq, CErq, SDrq, USrq,
916         TGrq, TIrq, TVrq, TPrq, TRrq,
917         TTrq;

919     SIGNALLIST PDU.indication =
920         PCin, PTin, DUin, CDin, CAin,
921         CEin, SDin, USin, TVin, TPin;

923     SIGNALLIST PDU.response =
924         TVrs;

926     SIGNALLIST PDU.confirm =
927         AUcf, CJcf, CCcf, TGcf, TIcf,
928         TVcf, TRcf, TTcf;

930     SIGNALLIST PDU.ultimatum =
931         DPum, RJum;

933     SIGNAL  PDU.ready                   /* allows one domain MCSPDU */
934         (
935               DataPriority
936         );

938     SIGNAL  PDin    (PDUStruct);    /* plumb domain indication */
939     SIGNAL  EDrq    (PDUStruct);    /* erect domain request */
940     SIGNAL  MCrq    (PDUStruct);    /* merge channels request */
941     SIGNAL  MCcf    (PDUStruct);    /* merge channels confirm */
942     SIGNAL  PCin    (PDUStruct);    /* purge channels indication */
943     SIGNAL  MTrq    (PDUStruct);    /* merge tokens request */
944     SIGNAL  MTcf    (PDUStruct);    /* merge tokens confirm */
945     SIGNAL  PTin    (PDUStruct);    /* purge tokens indication */
946     SIGNAL  DPum    (PDUStruct);    /* disconnect provider ultimatum */
947     SIGNAL  RJum    (PDUStruct);    /* reject MCSPDU ultimatum */
```

```
948   SIGNAL  AUrq    (PDUStruct);    /* attach user request */
949   SIGNAL  AUcf    (PDUStruct);    /* attach user confirm */
950   SIGNAL  DUrq    (PDUStruct);    /* detach user request */
951   SIGNAL  DUin    (PDUStruct);    /* detach user confirm */
952   SIGNAL  CJrq    (PDUStruct);    /* channel join request */
953   SIGNAL  CJcf    (PDUStruct);    /* channel join confirm */
954   SIGNAL  CLrq    (PDUStruct);    /* channel leave request */
955   SIGNAL  CCrq    (PDUStruct);    /* channel convene request */
956   SIGNAL  CCcf    (PDUStruct);    /* channel convene confirm */
957   SIGNAL  CDrq    (PDUStruct);    /* channel disband request */
958   SIGNAL  CDin    (PDUStruct);    /* channel disband confirm */
959   SIGNAL  CArq    (PDUStruct);    /* channel admit request */
960   SIGNAL  CAin    (PDUStruct);    /* channel admit confirm */
961   SIGNAL  CErq    (PDUStruct);    /* channel expel request */
962   SIGNAL  CEin    (PDUStruct);    /* channel expel confirm */
963   SIGNAL  SDrq    (PDUStruct);    /* send data request */
964   SIGNAL  SDin    (PDUStruct);    /* send data indication */
965   SIGNAL  USrq    (PDUStruct);    /* uniform send data request */
966   SIGNAL  USin    (PDUStruct);    /* uniform send data indication */
967   SIGNAL  TGrq    (PDUStruct);    /* token grab request */
968   SIGNAL  TGcf    (PDUStruct);    /* token grab confirm */
969   SIGNAL  TIrq    (PDUStruct);    /* token inhibit request */
970   SIGNAL  TIcf    (PDUStruct);    /* token inhibit confirm */
971   SIGNAL  TVrq    (PDUStruct);    /* token give request */
972   SIGNAL  TVin    (PDUStruct);    /* token give indication */
973   SIGNAL  TVrs    (PDUStruct);    /* token give response */
974   SIGNAL  TVcf    (PDUStruct);    /* token give confirm */
975   SIGNAL  TPrq    (PDUStruct);    /* token please request */
976   SIGNAL  TPin    (PDUStruct);    /* token please indication */
977   SIGNAL  TRrq    (PDUStruct);    /* token release request */
978   SIGNAL  TRcf    (PDUStruct);    /* token release confirm */
979   SIGNAL  TTrq    (PDUStruct);    /* token test request */
980   SIGNAL  TTcf    (PDUStruct);    /* token test confirm */

982   ENDBLOCK;
```

付録3　コントロール プロセスのＳＤＬ記述

（ＪＴ－Ｔ１２５に対する）

```
1   PROCESS  Control;

3          /* Type definitions */

5   NEWTYPE        Proc
6   LITERALS
7          Nil,
8          Receiving,            /*          Endpoint */
9          Responding,           /*          Endpoint */
10         Engaged,              /* Attachment Endpoint */
11         Quitting,             /* Attachment Endpoint */
12         Quit;                 /* Attachment Endpoint */
13  ENDNEWTYPE;

15  NEWTYPE        ProcByPri      Array(DataPriority, Proc);
16  ENDNEWTYPE;

18  NEWTYPE        CallSide
19  LITERALS
20         Calling,
21         Called;
22  ENDNEWTYPE;

24  NEWTYPE        PortalStruct
25  STRUCT
26         mcId           MCSConnectionId;       /* equals PortalId index */
27         ccId           Natural;              /* equals PortalId index */
28         kind           PortalKind;           /* (Attached,Downlink,Uplink) */
29         pids           PIdByPri;             /* processes comprising a portal */
30         proc           ProcByPri;            /* state of each created process */
31         label          Natural;              /* requester's label for confirm */
32         domain         DomainSelector;        /* domain selected by the portal */
33         opened         Boolean;              /* True if portal has been opened */
34         notify         Boolean;              /* True to notify when portal quit */
35         minParms       DomainParameters;      /* lower limit for negotiation */
36         maxParms       DomainParameters;      /* upper limit for negotiation */
37         parameters     DomainParameters;      /* values negotiated by portal */
38         callSide       CallSide;             /* portal is calling or called */
39         localTSAP      TSAPAddress;           /* local address for T.Connect */
40         remoteTSAP     TSAPAddress;           /* remote address for T.Connect */
41         targetQOSByPri  TransportQOSByPri;     /* desired quality of service */
42         minQOSByPri    TransportQOSByPri;     /* minimum that is acceptable */
43         userData       UserData;             /* of response, pending confirm */
44  ENDNEWTYPE;
45         /* Note:  In a practical implementation, the user data stored from
46         Connect.Response, awaiting establishment of additional TCs, need be
47         only one transport interface data unit, not a complete TSDU.  Any
48         excess can be left in the pipeline of the initial TC to be read out
49         when MCS.Connect.Provider.confirm is issued. */

51  NEWTYPE        Portal         Array(PortalId, PortalStruct);
52  ENDNEWTYPE;

54  NEWTYPE        DomainStruct
55  STRUCT
56         pid            PId;                  /* process for the domain or Null */
57         portals        Natural;               /* number of portals open to domain */
58         upward         portalId;              /* unique connection upward or zero */
59         minParms       DomainParameters;      /* lower limit of configuration */
60         maxParms       DomainParameters;      /* upper limit of configuration */
61         parameters     DomainParameters;      /* values established in domain */
62  ENDNEWTYPE;

64  NEWTYPE        Domain         Array(DomainSelector, DomainStruct);
65  ENDNEWTYPE;

67  NEWTYPE        DomainSelectorSet      SetOf(DomainSelector);
68  ENDNEWTYPE;

70         /* Data declarations */

72  DCL    domain         Domain,               /* resource arrays */
73         portal         Portal;
```

```
75          /* Note: The fields of a domain or portal array element
76          are undefined if the corresponding index is not in dUsed
77          or pUsed respectively. */

79    DCL     dUsed          DomainSelectorSet,     /* indexes used */
80            pUsed          PortalIdSet;

82    DCL     pFree          PortalIdSet;          /* indexes free */

84    DCL     nullParms      DomainParameters;      /* initializer */

86          /* Procedure decomposition */

88          /*      Initialize_resources
89                  Identify_sender               (p, dp)
90                  Min_parms                    (min, a, b)
91                  Max_parms                    (max, a, b)
92                  Test_parms                   (result, x, min, max)
93                  MCS_Connect_Provider_request   (...)
94                  T_Connect_indication          (...)
95                  Connect_Initial              (...)
96                  MCS_Connect_Provider_response  (...)
97                  Connect_Response             (...)
98                  Connect_Additional           (...)
99                  Connect_Result               (...)
100                 MCS_Disconnect_Provider_request (...)
101                 MCS_Attach_User_request       (...)
102                 Open_portal                  (p)
103                 Drop_portal                  (p, reason)
104                 Report_portal                (p, diagnostic)
105                 RJum                         (pdu)
106                 Quit_portal                  (p, reason)
107                 Quit
108                 Shut_portal                  (p)
109                 Exit_portal                  (p)     */

111                                                    /*-------------------------*/
112   PROCEDURE      Initialize_resources;                  /*  Initialize_resources
*/
113                                                    /*-------------------------*/
114         DCL     p             PortalId,
115                 ds            DomainSelector,
116                 dSet          DomainSelectorSet;
117         START
118         COMMENT'Initialize data structures during process start-up
119                 before accepting the first input signal.
120                 Note that each SetOf automatically defaults to Empty.
121                 Configure one hypothetical domain as an example.
122                 Some limits are determined by the implementation.
123                 ';
124         TASK    nullParms!maxChannelIds := 0,
125                 nullParms!maxUserIds := 0,
126                 nullParms!maxTokenIds := 0,
127                 nullParms!numPriorities := 0,
128                 nullParms!minThroughput := 0,
129                 nullParms!maxHeight := 0,
130                 nullParms!maxMCSPDUsize := 0,
131                 nullParms!protocolVersion := 0,
132                 p := maxPortalIds;
133          1b :   /* for p = ?..1 */
134         DECISION p > 0;
135         (True): TASK    portal(p)!mcId := p,
136                         portal(p)!ccId := p,
137                         pFree := Incl(p, pFree),
138                         p := p - 1;
139                 JOIN 1b;
140         ELSE:ENDDECISION;
141         TASK    ds := Mkstring(77) // Mkstring(67) // Mkstring(83),
142                 dUsed := Incl(ds, dUsed),
143                 domain(ds)!Pid := Null,
144                 domain(ds)!portals := 0,
145                 domain(ds)!upward := 0,
146                 dSet := dUsed;
147          2b :   /* for ds in dSet */
148         DECISION dSet = Empty;
149         (False):TASK    ds := Pick(dSet),
150                         dSet := Del(ds, dSet),
151                         domain(ds)!minParms!maxChannelIds := 0,
```

```
152                     domain(ds)!minParms!maxUserIds := 0,
153                     domain(ds)!minParms!maxTokenIds := 0,
154                     domain(ds)!minParms!numPriorities := 1,
155                     domain(ds)!minParms!minThroughput := 0,
156                     domain(ds)!minParms!maxHeight := 1,
157                     domain(ds)!minParms!maxMCSPDUsize := 35,
158                     domain(ds)!minParms!protocolVersion := 1,
159                     domain(ds)!maxParms!maxChannelIds := 65535,
160                     domain(ds)!maxParms!maxUserIds := 65535,
161                     domain(ds)!maxParms!maxTokenIds := 65535,
162                     domain(ds)!maxParms!numPriorities := 4,
163                     domain(ds)!maxParms!minThroughput := 1000000,
164                     domain(ds)!maxParms!maxHeight := 1000,
165                     domain(ds)!maxParms!maxMCSPDUsize := 32768,
166                     domain(ds)!maxParms!protocolVersion := 2;
167               JOIN 2b;
168         ELSE:ENDDECISION;
169         RETURN;
170         ENDPROCEDURE;

172                                                          /*--------------------*/
173   PROCEDURE      Identify_sender;                          /*  Identify_sender  */
174   FPAR    IN/OUT  p              PortalId,                 /*--------------------*/
175           IN/OUT  dp             DataPriority;

177         DCL     pSet           PortalIdSet;
178         START
179         COMMENT'An alternative would be to carry this
180                 information explicitly in SDL signals.
181                 ';
182         TASK    pSet := pUsed;
183          1b :   /* for p in pSet */
184         DECISION pSet = Empty;
185         (False):TASK    p := Pick(pSet),
186                         pSet := Del(p, pSet),
187                         dp := 0;
188                  2b :   /* for dp = 0..3 */
189                 DECISION dp < 4;
190                 (True): DECISION portal(p)!pids(dp) = SENDER;
191                         (True): RETURN;
192                         ELSE:ENDDECISION;
193                         TASK    dp := dp + 1;
194                         JOIN 2b;
195                 ELSE:ENDDECISION;
196                 JOIN 1b;
197         ELSE:ENDDECISION;
198         TASK    p := 0,
199                 dp := 0;
200         RETURN;
201         ENDPROCEDURE;

203                                                          /*--------------*/
204   PROCEDURE      Min_parms;                              /*  Min_parms   */
205   FPAR    IN/OUT  min            DomainParameters,        /*--------------*/
206                 a              DomainParameters,
207                 b              DomainParameters;

209         START
210         COMMENT'Return the minimum of two parameter sets.
211                 ';
212         TASK    min!maxChannelIds := IF a!maxChannelIds < b!maxChannelIds
213                                 THEN a!maxChannelIds ELSE b!maxChannelIds FI,
214                 min!maxUserIds := IF a!maxUserIds < b!maxUserIds
215                                 THEN a!maxUserIds ELSE b!maxUserIds FI,
216                 min!maxTokenIds := IF a!maxTokenIds < b!maxTokenIds
217                                 THEN a!maxTokenIds ELSE b!maxTokenIds FI,
218                 min!numPriorities := IF a!numPriorities < b!numPriorities
219                                 THEN a!numPriorities ELSE b!numPriorities FI,
220                 min!minThroughput := IF a!minThroughput < b!minThroughput
221                                 THEN a!minThroughput ELSE b!minThroughput FI,
222                 min!maxHeight := IF a!maxHeight < b!maxHeight
223                                 THEN a!maxHeight ELSE b!maxHeight FI,
224                 min!maxMCSPDUsize := IF a!maxMCSPDUsize < b!maxMCSPDUsize
225                                 THEN a!maxMCSPDUsize ELSE b!maxMCSPDUsize FI,
226                 min!protocolVersion := IF a!protocolVersion < b!protocolVersion
227                                 THEN a!protocolVersion ELSE b!protocolVersion FI;
228         RETURN;
229         ENDPROCEDURE;
```

```
                                                                  /*-------------*/
PROCEDURE      Max_parms;                                           /*  Max_parms  */
FPAR    IN/OUT  max            DomainParameters,                    /*-------------*/
                a              DomainParameters,
                b              DomainParameters;

        START
        COMMENT'Return the maximum of two parameter sets.
                ';
        TASK    max!maxChannelIds := IF a!maxChannelIds > b!maxChannelIds
                                  THEN a!maxChannelIds ELSE b!maxChannelIds FI,
                max!maxUserIds := IF a!maxUserIds > b!maxUserIds
                                  THEN a!maxUserIds ELSE b!maxUserIds FI,
                max!maxTokenIds := IF a!maxTokenIds > b!maxTokenIds
                                  THEN a!maxTokenIds ELSE b!maxTokenIds FI,
                max!numPriorities := IF a!numPriorities > b!numPriorities
                                  THEN a!numPriorities ELSE b!numPriorities FI,
                max!minThroughput := IF a!minThroughput > b!minThroughput
                                  THEN a!minThroughput ELSE b!minThroughput FI,
                max!maxHeight := IF a!maxHeight > b!maxHeight
                                  THEN a!maxHeight ELSE b!maxHeight FI,
                max!maxMCSPDUsize := IF a!maxMCSPDUsize > b!maxMCSPDUsize
                                  THEN a!maxMCSPDUsize ELSE b!maxMCSPDUsize FI,
                max!protocolVersion := IF a!protocolVersion > b!protocolVersion
                                  THEN a!protocolVersion ELSE b!protocolVersion FI;
        RETURN;
        ENDPROCEDURE;

                                                                  /*--------------*/
PROCEDURE      Test_parms;                                          /*  Test_parms  */
FPAR    IN/OUT  result         Result,                              /*--------------*/
                z              DomainParameters,
                min            DomainParameters,
                max            DomainParameters;

        START
        COMMENT'Check that the parameters lie between min and max.
                ';
        DECISION (z!maxChannelIds >= min!maxChannelIds)
            and (z!maxChannelIds <= max!maxChannelIds)
            and (z!maxUserIds >= min!maxUserIds)
            and (z!maxUserIds <= max!maxUserIds)
            and (z!maxTokenIds >= min!maxTokenIds)
            and (z!maxTokenIds <= max!maxTokenIds)
            and (z!numPriorities >= min!numPriorities)
            and (z!numPriorities <= max!numPriorities)
            and (z!minThroughput >= min!minThroughput)
            and (z!minThroughput <= max!minThroughput)
            and (z!maxHeight >= min!maxHeight)
            and (z!maxHeight <= max!maxHeight)
            and (z!maxMCSPDUsize >= min!maxMCSPDUsize)
            and (z!maxMCSPDUsize <= max!maxMCSPDUsize)
            and (z!protocolVersion >= min!protocolVersion)
            and (z!protocolVersion <= max!protocolVersion);
        (True): TASK    result := RT_successful;
        (False):TASK    result := RT_parameters_unacceptable;
        ENDDECISION;
        RETURN;
        ENDPROCEDURE;

                                                                  /*--------------------------------*/
PROCEDURE      MCS_Connect_Provider_request;       /*
MCS_Connect_Provider_request  */
FPAR           label          Natural,           /*--------------------------------
*/
               localTSAP      TSAPAddress,
               localDomain    DomainSelector,
               remoteTSAP     TSAPAddress,
               remoteDomain   DomainSelector,
               upward         Boolean,
               targetParms    DomainParameters,
               minParms       DomainParameters,
               maxParms       DomainParameters,
               targetQOSByPri TransportQOSByPri,
               minQOSByPri    TransportQOSByPri,
               userData       UserData;

        DCL    result         Result,
               p              PortalId,
```

```
308             dp              DataPriority,
309             ds              DomainSelector;
310         START
311         COMMENT'Process an MCS.Connect.Provider.request input signal.
312             Begin parameter negotiation and allocate a portal.
313             Create an endpoint process for the initial TC and
314             transmit Connect.Initial through it.
315             ';
316         DECISION pFree = Empty;
317         (True): TASK    result := RT_congested;
318             JOIN 1f;
319         ELSE:ENDDECISION;
320         TASK    ds := localDomain;
321         DECISION ds in dUsed;
322         (False):TASK    result := RT_no_such_domain;
323             JOIN 1f;
324         ELSE:ENDDECISION;
325         DECISION upward and domain(ds)!upward /= 0;
326         (True): TASK    result := RT_domain_not_hierarchical;
327             JOIN 1f;
328         ELSE:ENDDECISION;
329         CALL    Max_parms(minParms, minParms, domain(ds)!minParms);
330         CALL    Min_parms(maxParms, maxParms, domain(ds)!maxParms);
331         DECISION domain(ds)!portals > 0;
332         (True): TASK    targetParms := domain(ds)!parameters;
333         (False):CALL    Max_parms(targetParms, targetParms, minParms);
334             CALL    Min_parms(targetParms, targetParms, maxParms);
335         ENDDECISION;
336         CALL    Test_parms(result, targetParms, minParms, maxParms);
337         DECISION result = RT_successful;
338         (False): 1f :
339             OUTPUT  MCS.Connect.Provider.confirm
340                 (label, result, 0, nullParms, NullString);
341             RETURN;
342         ELSE:ENDDECISION;
343         DECISION domain(ds)!portals > 0;
344         (True): TASK    minParms := targetParms,
345                 maxParms := targetParms;
346         ELSE:ENDDECISION;
347         CREATE  Endpoint(Null, localTSAP, remoteTSAP,
348             targetQOSByPri(0), minQOSByPri(0),
349             nullParms);
350         OUTPUT  Connect.Initial(localDomain, remoteDomain, upward,
351             targetParms, minParms, maxParms, userData)
352             TO OFFSPRING;
353         TASK    p := Pick(pFree),
354             pFree := Del(p, pFree),
355             pUsed := Incl(p, pUsed),
356             portal(p)!kind := IF upward THEN Uplink ELSE Downlink FI,
357             portal(p)!label := label,
358             portal(p)!domain := ds,
359             portal(p)!opened := False,
360             portal(p)!notify := True,
361             portal(p)!minParms := minParms,
362             portal(p)!maxParms := maxParms,
363             portal(p)!parameters := nullParms,
364             portal(p)!callSide := Calling,
365             portal(p)!localTSAP := localTSAP,
366             portal(p)!remoteTSAP := remoteTSAP,
367             portal(p)!targetQOSByPri := targetQOSByPri,
368             portal(p)!minQOSByPri := minQOSByPri,
369             portal(p)!pids(0) := OFFSPRING,
370             portal(p)!proc(0) := Receiving,
371             dp := 1;
372          2b :   /* for dp = 1..3 */
373         DECISION dp < 4;
374         (True): TASK    portal(p)!pids(dp) := Null,
375                 portal(p)!proc(dp) := Nil,
376                 dp := dp + 1;
377             JOIN 2b;
378         ELSE:ENDDECISION;
379         RETURN;
380         ENDPROCEDURE;

382                                                 /*-------------------------*/
383     PROCEDURE       T_Connect_indication;           /*  T_Connect_indication
*/
384     FPAR            tcId            TCEndpointId,           /*-------------------------
*/
```

```
385                remoteTSAP      TSAPAddress,
386                localTSAP       TSAPAddress,
387                offeredQOS      TransportQOS,
388                minQOS          TransportQOS;

390          DCL    p              PortalId,
391                 dp             DataPriority;
392          START
393          COMMENT'Process a T.Connect.indication input signal.
394                 Allocate a portal, in case this is an initial TC.
395                 Create an endpoint process to receive Connect.Initial
396                 or Connect.Additional.
397                 ';
398          DECISION pFree = Empty
399                 or offeredQOS!throughput < minQOS!throughput
400                 or offeredQOS!transitDelay > minQOS!transitDelay
401                 or offeredQOS!dataPriority > minQOS!dataPriority;
402          (True): OUTPUT  T.Disconnect.request(tcId);
403                 RETURN;
404          ELSE:ENDDECISION;
405          CREATE  Endpoint(tcId, localTSAP, remoteTSAP,
406                 offeredQOS, minQOS,
407                 nullParms);
408          TASK    p := Pick(pFree),
409                 pFree := Del(p, pFree),
410                 pUsed := Incl(p, pUsed),
411                 portal(p)!kind := Downlink,
412                 portal(p)!opened := False,
413                 portal(p)!notify := False,
414                 portal(p)!parameters := nullParms,
415                 portal(p)!callSide := Called,
416                 portal(p)!localTSAP := localTSAP,
417                 portal(p)!remoteTSAP := remoteTSAP,
418                 portal(p)!pids(0) := OFFSPRING,
419                 portal(p)!proc(0) := Receiving,
420                 dp := 1;
421           1b :   /* for dp = 1..3 */
422          DECISION dp < 4;
423          (True): TASK    portal(p)!pids(dp) := Null,
424                        portal(p)!proc(dp) := Nil,
425                        dp := dp + 1;
426                 JOIN 1b;
427          ELSE:ENDDECISION;
428          RETURN;
429          ENDPROCEDURE;

431                                                          /*--------------------*/
432    PROCEDURE       Connect_Initial;                          /*  Connect_Initial  */
433    FPAR            remoteDomain    DomainSelector,            /*--------------------*/
434                    localDomain     DomainSelector,
435                    upward          Boolean,
436                    targetParms     DomainParameters,
437                    minParms        DomainParameters,
438                    maxParms        DomainParameters,
439                    userData        UserData;

441          DCL    result          Result,
442                 p               PortalId,
443                 dp              DataPriority,
444                 ds              DomainSelector;
445          START
446          COMMENT'Process a Connect.Initial input signal.
447                 Retain the portal and begin parameter negotiation.
448                 Indicate the connection to the controlling user.
449                 ';
450          CALL    Identify_sender(p, dp);
451          DECISION p in pUsed and portal(p)!callSide = Called
452                 and dp = 0 and portal(p)!proc(0) = Receiving;
453          (False):TASK    result := RT_unspecified_failure;
454                 JOIN 1f;
455          ELSE:ENDDECISION;
456          TASK    ds := localDomain;
457          DECISION ds in dUsed;
458          (False):TASK    result := RT_no_such_domain;
459                 JOIN 1f;
460          ELSE:ENDDECISION;
461          DECISION upward or domain(ds)!upward = 0;
462          (False): TASK   result := RT_domain_not_hierarchical;
463                 JOIN 1f;
```

```
464           ELSE:ENDDECISION;
465           CALL    Max_parms(minParms, minParms, domain(ds)!minParms);
466           CALL    Min_parms(maxParms, maxParms, domain(ds)!maxParms);
467           DECISION domain(ds)!portals > 0;
468           (True): TASK    targetParms := domain(ds)!parameters;
469           (False):CALL    Max_parms(targetParms, targetParms, minParms);
470                   CALL    Min_parms(targetParms, targetParms, maxParms);
471           ENDDECISION;
472           CALL    Test_parms(result, targetParms, minParms, maxParms);
473           DECISION result = RT_successful;
474           (False): 1f :
475                   OUTPUT  Connect.Response(result, 0, nullParms, NullString)
476                           TO SENDER;
477                   CALL    Quit_portal(p, RN_unspecified);
478                   RETURN;
479           ELSE:ENDDECISION;
480           DECISION domain(ds)!portals > 0;
481           (True): TASK    minParms := targetParms,
482                           maxParms := targetParms;
483           ELSE:ENDDECISION;
484           TASK    portal(p)!kind := IF upward THEN Downlink ELSE Uplink FI,
485                   portal(p)!domain := ds,
486                   portal(p)!notify := True,
487                   portal(p)!minParms := minParms,
488                   portal(p)!maxParms := maxParms,
489                   portal(p)!proc(0) := Responding;
490           OUTPUT  MCS.Connect.Provider.indication
491                   (portal(p)!mcId, portal(p)!localTSAP, localDomain,
portal(p)!remoteTSAP,
492                   remoteDomain, upward, targetParms, minParms, maxParms, userData);
493           RETURN;
494           ENDPROCEDURE;

496                                                   /*---------------------------------*/
497   PROCEDURE       MCS_Connect_Provider_response;      /*
MCS_Connect_Provider_response  */
498   FPAR            mcId            MCSConnectionId,    /*-------------------------------
-*/
499                   result          Result,
500                   parameters      DomainParameters,
501                   userData        UserData;

503           DCL     p               PortalId;
504           START
505           COMMENT'Process an MCS.Connect.Provider.response input signal.
506                   Check negotiated parameters and force a preference.
507                   Transmit Connect.Response.
508                   If there are no additional TCs, open the portal.
509                   ';
510           TASK    p := mcId;
511           DECISION p in pUsed and portal(p)!proc(0) = Responding;
512           (False):RETURN;
513           ELSE:ENDDECISION;
514           DECISION result = RT_successful;
515           (False):TASK    result := RT_user_rejected,
516                           portal(p)!notify := False;
517                   JOIN 1f;
518           ELSE:ENDDECISION;
519           CALL    Test_parms(result, parameters, portal(p)!minParms,
portal(p)!maxParms);
520           DECISION result = RT_successful;
521           (False): 1f :
522                   OUTPUT  Connect.Response(result, 0, parameters, userData)
523                           TO portal(p)!pids(0);
524                   CALL    Quit_portal(p, RN_parameters_unacceptable);
525                   RETURN;
526           ELSE:ENDDECISION;
527           CALL    Max_parms(parameters, parameters, portal(p)!minParms);
528           OUTPUT  Connect.Response(result, portal(p)!ccId, parameters, userData)
529                   TO portal(p)!pids(0);
530           TASK    portal(p)!parameters := parameters,
531                   portal(p)!proc(0) := Engaged;
532           CALL    Open_portal(p);
533           RETURN;
534           ENDPROCEDURE;

536                                                   /*--------------------*/
537   PROCEDURE       Connect_Response;                   /*  Connect_Response  */
538   FPAR            result          Result,             /*--------------------*/
```

```
539                 ccId            Natural,
540                 parameters      DomainParameters,
541                 userData        UserData;

543           DCL   p               PortalId,
544                 dp              DataPriority;
545           START
546           COMMENT'Process a Connect.Response input signal.
547                 Check the negotiated parameters.
548                 Create endpoint processes for additional TCs and
549                 transmit Connect.Additional through them.
550                 If there are no additional TCs, open the portal.
551                 ';
552           CALL  Identify_sender(p, dp);
553           DECISION p in pUsed and portal(p)!callSide = Calling
554                 and dp = 0 and portal(p)!proc(0) = Receiving;
555           (False):CALL  Quit_portal(p, RN_unspecified);
556                 RETURN;
557           ELSE:ENDDECISION;
558           TASK  portal(p)!parameters := parameters,
559                 portal(p)!userData := userData;
560           DECISION result = RT_successful;
561           (False):JOIN 1f;
562           ELSE:ENDDECISION;
563           CALL  Test_parms(result, parameters, portal(p)!minParms,
portal(p)!maxParms);
564           DECISION result = RT_successful;
565           (False): 1f :
566                 OUTPUT MCS.Connect.Provider.confirm
567                       (portal(p)!label, result, 0, parameters, userData);
568                 TASK  portal(p)!notify := False;
569                 CALL  Quit_portal(p, RN_unspecified);
570                 RETURN;
571           ELSE:ENDDECISION;
572           TASK  portal(p)!proc(0) := Engaged,
573                 dp := 1;
574            2b :  /* for dp = 1..? */
575           DECISION dp < parameters!numPriorities;
576           (True): CREATE Endpoint(Null, portal(p)!localTSAP, portal(p)!remoteTSAP,
577                       portal(p)!targetQOSByPri(dp), portal(p)!minQOSByPri(dp),
578                       parameters);
579                 OUTPUT Connect.Additional(ccId, dp)
580                       TO OFFSPRING;
581                 TASK  portal(p)!pids(dp) := OFFSPRING,
582                       portal(p)!proc(dp) := Receiving,
583                       dp := dp + 1;
584                 JOIN 2b;
585           ELSE:ENDDECISION;
586           CALL  Open_portal(p);
587           RETURN;
588           ENDPROCEDURE;

590                                                          /*----------------------*/
591     PROCEDURE     Connect_Additional;                        /*  Connect_Additional
*/
592     FPAR          ccId            Natural,                 /*----------------------*/
593                   dp              DataPriority;

595           DCL   p               PortalId,
596                 r               PortalId,
597                 x               DataPriority;
598           START
599           COMMENT'Process a Connect.Additional input signal.
600                 Release the allocated portal and piggyback onto
601                 the preceding Connect.Initial.
602                 Transmit Connect.Result.
603                 If all TCs are established, open the portal.
604                 ';
605           CALL  Identify_sender(r, x);
606           DECISION r in pUsed and portal(r)!callSide = Called
607                 and x = 0 and portal(r)!proc(0) = Receiving;
608           (False):JOIN 1f;
609           ELSE:ENDDECISION;
610           TASK  p := ccId;
611           DECISION p in pUsed and portal(p)!callSide = Called
612                 and dp > 0 and dp < portal(p)!parameters!numPriorities
613                 and portal(p)!proc(0) = Engaged and portal(p)!proc(dp) = Nil;
614           (False): 1f :
615                 OUTPUT Connect.Result(RT_unspecified_failure)
```

```
                        TO SENDER;
                CALL    Quit_portal(r, RN_unspecified);
                RETURN;
        ELSE:ENDDECISION;
        TASK    pUsed := Del(r, pUsed),
                pFree := Incl(r, pFree);
        OUTPUT  Connect.Result(RT_successful)
                TO SENDER;
        TASK    portal(p)!pids(dp) := SENDER,
                portal(p)!proc(dp) := Engaged;
        CALL    Open_portal(p);
        RETURN;
        ENDPROCEDURE;

                                                        /*------------------*/
PROCEDURE       Connect_Result;                            /* Connect_Result */
FPAR            result          Result;                   /*------------------*/

        DCL     p               PortalId,
                dp              DataPriority;
        START
        COMMENT'Process a Connect.Result input signal.
                If all TCs are established, open the portal.
                ';
        CALL    Identify_sender(p, dp);
        DECISION p in pUsed and portal(p)!callSide = Calling
                and dp > 0 and portal(p)!proc(dp) = Receiving;
        (False):CALL    Quit_portal(p, RN_unspecified);
                RETURN;
        ELSE:ENDDECISION;
        DECISION result = RT_successful;
        (False):OUTPUT  MCS.Connect.Provider.confirm
                        (portal(p)!label, result, 0,
                        portal(p)!parameters, portal(p)!userData);
                TASK    portal(p)!notify := False;
                CALL    Quit_portal(p, RN_unspecified);
                RETURN;
        ELSE:ENDDECISION;
        TASK    portal(p)!proc(dp) := Engaged;
        CALL    Open_portal(p);
        RETURN;
        ENDPROCEDURE;

                                                /*-----------------------------------*/
PROCEDURE       MCS_Disconnect_Provider_request;  /*
MCS_Disconnect_Provider_request  */
FPAR            mcId            MCSConnectionId;  /*-----------------------------------
-*/

        DCL     p               PortalId,
                ds              DomainSelector;
        START
        COMMENT'Process an MCS.Disconnect.Provider.request input signal.
                If the portal is open, this can be done gracefully.
                ';
        TASK    p := mcId;
        DECISION p in pUsed and portal(p)!notify;
        (True): TASK    portal(p)!notify := False,
                        ds := portal(p)!domain;
                DECISION portal(p)!opened and domain(ds)!portals > 0;
                (True): OUTPUT  Drop.portal(p, RN_user_requested) TO domain(ds)!pid;
                (False):CALL    Quit_portal(p, RN_unspecified);
                ENDDECISION;
        ELSE:ENDDECISION;
        RETURN;
        ENDPROCEDURE;

                                                        /*--------------------------*/
PROCEDURE       MCS_Attach_User_request;                    /* MCS_Attach_User_request
*/
FPAR            label           Natural,                  /*--------------------------
*/
                localDomain     DomainSelector;

        DCL     p               PortalId,
                dp              DataPriority,
                ds              DomainSelector;
        START
        COMMENT'Process an MCS.Attach.User.request input signal.
```

```
691             Allocate a portal, expecting to create an attachment,
692             and open the portal.
693             ';
694         DECISION pFree = Empty;
695         (True): TASK    result := RT_congested;
696                 JOIN 1f;
697         ELSE:ENDDECISION;
698         TASK    ds := localDomain;
699         DECISION ds in dUsed;
700         (False):TASK    result := RT_no_such_domain;
701                 1f :
702                 OUTPUT  MCS.Attach.User.confirm
703                         (label, result, Null, 0);
704                 RETURN;
705         ELSE:ENDDECISION;
706         TASK    p := Pick(pFree),
707                 pFree := Del(p, pFree),
708                 pUsed := Incl(p, pUsed),
709                 portal(p)!kind := Attached,
710                 portal(p)!label := label,
711                 portal(p)!domain := ds,
712                 portal(p)!opened := False,
713                 portal(p)!notify := False,
714                 portal(p)!pids(0) := Null,
715                 portal(p)!proc(0) := Engaged,
716                 dp := 1;
717          2b :   /* for dp = 1..3 */
718         DECISION dp < 4;
719         (True): TASK    portal(p)!pids(dp) := Null,
720                         portal(p)!proc(dp) := Nil,
721                         dp := dp + 1;
722                 JOIN 2b;
723         ELSE:ENDDECISION;
724         CALL    Open_portal(p);
725         RETURN;
726         ENDPROCEDURE;

728                                                         /*---------------*/
729     PROCEDURE       Open_portal;                           /*  Open_portal  */
730     FPAR            p               PortalId;             /*---------------*/

732         DCL     dp              DataPriority,
733                 ds              DomainSelector,
734                 numP            Natural,
735                 parameters      DomainParameters;
736         START
737         COMMENT'When all TCs have been established, or if this
738                 is a user attachment, open the portal to a domain.
739                 If the domain process is stopping, try again later.
740                 Attachments must wait until domain parameters are set.
741                 ';
742         TASK    ds := portal(p)!domain,
743                 dp := 0;
744         DECISION portal(p)!kind = Attached;
745         (True): TASK    parameters := domain(ds)!minParms,
746                         numP := 1;
747         (False):TASK    parameters := portal(p)!parameters,
748                         numP := parameters!numPriorities;
749         ENDDECISION;
750          1b :   /* for dp = 0..? */
751         DECISION dp < numP;
752         (True): DECISION portal(p)!proc(dp) = Engaged;
753                 (False):RETURN;
754                 ELSE:ENDDECISION;
755                 TASK    dp := dp + 1;
756                 JOIN 1b;
757         ELSE:ENDDECISION;
758         DECISION domain(ds)!pid = Null;
759         (False):DECISION domain(ds)!portals = 0;
760                 (True): RETURN;
761                 ELSE:ENDDECISION;
762         (True): CREATE  Domain(parameters);
763                 TASK    domain(ds)!pid := OFFSPRING,
764                         domain(ds)!portals := 0,
765                         domain(ds)!upward := 0,
766                         domain(ds)!parameters := parameters;
767         ENDDECISION;
768         DECISION portal(p)!kind = Attached;
769         (True): CREATE  Attachment(portal(p)!label, domain(ds)!parameters);
```

```
770                 TASK    portal(p)!pids(0) := OFFSPRING;
771           (False):DECISION domain(ds)!parameters = parameters;
772                 (False):CALL    Quit_portal(p, RN_parameters_unacceptable);
773                       RETURN;
774                 ELSE:ENDDECISION;
775                 DECISION portal(p)!kind = Uplink;
776                 (True): DECISION domain(ds)!upward = 0;
777                       (False):CALL    Quit_portal(p, RN_domain_not_hierarchical);
778                             RETURN;
779                       (True): TASK    domain(ds)!upward := p;
780                       ENDDECISION;
781                 ELSE:ENDDECISION;
782                 DECISION portal(p)!callSide = Called;
783                 (False):OUTPUT  MCS.Connect.Provider.confirm
784                             (portal(p)!label, RT_successful, portal(p)!mcId,
785                             portal(p)!parameters, portal(p)!userData);
786                 ELSE:ENDDECISION;
787           ENDDECISION;
788           OUTPUT  Open.portal(p, portal(p)!kind, portal(p)!pids) TO domain(ds)!pid;
789           TASK    domain(ds)!portals := domain(ds)!portals + 1,
790                 portal(p)!opened := True;
791           RETURN;
792           ENDPROCEDURE;

794                                                                 /*---------------*/
795     PROCEDURE       Drop_portal;                                  /*  Drop_portal  */
796     FPAR            p               PortalId,                   /*---------------*/
797                     reason          Reason;

799           START
800           COMMENT'Process a Drop.portal input signal.
801                 ';
802           CALL    Quit_portal(p, reason);
803           RETURN;
804           ENDPROCEDURE;

806                                                                 /*-----------------*/
807     PROCEDURE       Report_portal;                                /*  Report_portal  */
808     FPAR            p               PortalId,                   /*-----------------*/
809                     diagnostic      Diagnostic;

811           DCL     reason          Reason;
812           START
813           COMMENT'Process a Report.portal input signal.
814                 For testing, local diagnostic could be logged.
815                 ';
816           TASK    reason := RN_unspecified;
817           DECISION diagnostic;
818           (DC_throughput_inadequate):
819                 TASK    reason := RN_provider_initiated;
820           (DC_height_limit_exceeded):
821                 TASK    reason := RN_domain_not_hierarchical;
822           ELSE:ENDDECISION;
823           CALL    Quit_portal(p, reason);
824           RETURN;
825           ENDPROCEDURE;

827                                                                 /*--------*/
828     PROCEDURE       RJum;                                         /*  RJum  */
829     FPAR            pdu             PDUStruct;                  /*--------*/

831           DCL     p               PortalId,
832                   dp              DataPriority;
833           START
834           COMMENT'Process an RJum input signal.
835                 For testing, remote pdu!diagnostic could be logged.
836                 ';
837           CALL    Identify_sender(p, dp);
838           CALL    Quit_portal(p, RN_unspecified);
839           RETURN;
840           ENDPROCEDURE;

842                                                                 /*---------------*/
843     PROCEDURE       Quit_portal;                                  /*  Quit_portal  */
844     FPAR            p               PortalId,                   /*---------------*/
845                     reason          Reason;

847           DCL     result          Result,
848                   dp              DataPriority;
```

```
849         START
850         COMMENT'If necessary, notify the controlling user.
851               Quiesce the portal processes.
852               ';
853         DECISION p in pUsed;
854         (False):RETURN;
855         ELSE:ENDDECISION;
856         DECISION portal(p)!notify;
857         (True): DECISION portal(p)!callSide = Called or portal(p)!opened;
858               (True): OUTPUT  MCS.Disconnect.Provider.indication
859                           (portal(p)!mcId, reason);
860               (False):DECISION reason;
861                     (RN_domain_not_hierarchical):
862                           TASK    result := RT_domain_not_hierarchical;
863                     (RN_parameters_unacceptable):
864                           TASK    result := RT_parameters_unacceptable;
865                     ELSE:
866                           TASK    result := RT_unspecified_failure;
867                     ENDDECISION;
868                     OUTPUT  MCS.Connect.Provider.confirm
869                           (portal(p)!label, result, 0, nullParms, NullString);
870               ENDDECISION;
871         ELSE:ENDDECISION;
872         TASK    portal(p)!notify := False,
873               dp := 0;
874          1b :   /* for dp = 0..3 */
875         DECISION dp < 4;
876         (True): DECISION portal(p)!proc(dp);
877               (Receiving, Responding, Engaged):
878                     OUTPUT  Quit TO portal(p)!pids(dp);
879                     TASK    portal(p)!proc(dp) := Quitting;
880               ELSE:ENDDECISION;
881               TASK    dp := dp + 1;
882               JOIN 1b;
883         ELSE:ENDDECISION;
884         RETURN;
885         ENDPROCEDURE;

887                                                                 /*--------*/
888   PROCEDURE      Quit;                                          /*  Quit  */
889                                                                 /*--------*/
890         DCL     p             PortalId,
891               pSet            PortalIdSet,
892               dp              DataPriority,
893               ds              DomainSelector;
894         START
895         COMMENT'Process a Quit input signal.
896               When all processes are quiesced, it is safe
897               to shut this portal on the domain.
898               If an upward portal, quiesce all others too.
899               ';
900         CALL    Identify_sender(p, dp);
901         DECISION p in pUsed;
902         (False):RETURN;
903         ELSE:ENDDECISION;
904         TASK    portal(p)!proc(dp) := Quit;
905         CALL    Quit_portal(p, RN_unspecified);
906         TASK    dp := 0;
907          1b :   /* for dp = 0..3 */
908         DECISION dp < 4;
909         (True): DECISION portal(p)!proc(dp);
910               (Receiving, Responding, Engaged, Quitting):
911                     RETURN;
912               ELSE:ENDDECISION;
913               TASK    dp := dp + 1;
914               JOIN 1b;
915         ELSE:ENDDECISION;
916         DECISION portal(p)!opened;
917         (False):CALL    Exit_portal(p);
918               RETURN;
919         ELSE:ENDDECISION;
920         TASK    ds := portal(p)!domain,
921               domain(ds)!portals := domain(ds)!portals - 1;
922         OUTPUT  Shut.portal(p) TO domain(ds)!pid;
923         DECISION portal(p)!kind = Uplink;
924         (True): TASK    domain(ds)!upward := 0,
925                     pSet := pUsed;
926                2b :   /* for p in pSet */
927               DECISION pSet = Empty;
```

```
928                 (False):TASK    p := Pick(pSet),
929                                 pSet := Del(p, pSet);
930                       DECISION portal(p)!opened and portal(p)!domain = ds;
931                       (True): CALL    Quit_portal(p, RN_domain_disconnected);
932                       ELSE:ENDDECISION;
933                       JOIN 2b;
934                 ELSE:ENDDECISION;
935           ELSE:ENDDECISION;
936           RETURN;
937           ENDPROCEDURE;

939                                                                  /*---------------*/
940     PROCEDURE       Shut_portal;                                  /*  Shut_portal  */
941     FPAR            p               PortalId;                    /*---------------*/

943           DCL     ds              DomainSelector,
944                   pSet            PortalIdSet;
945           START
946           COMMENT'Process a Shut.portal input signal.
947                   It is now safe to stop the portal processes.
948                   If this was the last portal, the domain process stops too
949                   so that it can be recreated with different parameters.
950                   ';
951           TASK    ds := portal(p)!domain;
952           CALL    Exit_portal(p);
953           DECISION domain(ds)!portals = 0;
954           (True): TASK    domain(ds)!pid := Null,
955                           pSet := pUsed;
956                   1b :    /* for p in pSet */
957                   DECISION pSet = Empty;
958                   (False):TASK    p := Pick(pSet),
959                                   pSet := Del(p, pSet);
960                           DECISION portal(p)!domain = ds;
961                           (True): CALL    Open_portal(p);
962                           ELSE:ENDDECISION;
963                           JOIN 1b;
964                   ELSE:ENDDECISION;
965           ELSE:ENDDECISION;
966           RETURN;
967           ENDPROCEDURE;

969                                                                  /*---------------*/
970     PROCEDURE       Exit_portal;                                  /*  Exit_portal  */
971     FPAR            p               PortalId;                    /*---------------*/

973           DCL     dp              DataPriority;
974           START
975           COMMENT'Release the portal and stop its processes.
976                   ';
977           TASK    pUsed := Del(p, pUsed),
978                   pFree := Incl(p, pFree),
979                   dp := 0;
980            1b :   /* for dp = 0..3 */
981           DECISION dp < 4;
982           (True): DECISION portal(p)!proc(dp);
983                   (Quit):
984                           OUTPUT  Exit TO portal(p)!pids(dp);
985                           TASK    portal(p)!proc(dp) := Nil;
986                   ELSE:ENDDECISION;
987                   TASK    dp := dp + 1;
988                   JOIN 1b;
989           ELSE:ENDDECISION;
990           RETURN;
991           ENDPROCEDURE;

993           /* Input transitions */

995           DCL     p               PortalId,
996                   dp              DataPriority,
997                   pdu             PDUStruct,
998                   mcId            MCSConnectionId,
999                   ccId            Natural,
1000                  tcId            TCEndpointId,
1001                  reason          Reason,
1002                  result          Result,
1003                  diagnostic      Diagnostic,
1004                  label           Natural,
1005                  localTSAP       TSAPAddress,
1006                  localDomain     DomainSelector,
```

```
1007                remoteTSAP      TSAPAddress,
1008                remoteDomain    DomainSelector,
1009                upward          Boolean,
1010                targetParms     DomainParameters,
1011                minParms        DomainParameters,
1012                maxParms        DomainParameters,
1013                parameters      DomainParameters,
1014                targetQOSByPri  TransportQOSByPri,
1015                minQOSByPri     TransportQOSByPri,
1016                offeredQOS      TransportQOS,
1017                minQOS          TransportQOS,
1018                userData        UserData;
1019          START
1020          COMMENT'The state machine contains a single state.
1021                ';
1022          CALL    Initialize_resources;
1023          NEXTSTATE ~;

1025    STATE ~;INPUT   MCS.Connect.Provider.request(label, localTSAP, localDomain,
1026                remoteTSAP, remoteDomain, upward, targetParms, minParms, maxParms,
1027                targetQOSByPri, minQOSByPri, userData);
1028          CALL    MCS_Connect_Provider_request(label, localTSAP, localDomain,
1029                remoteTSAP, remoteDomain, upward, targetParms, minParms, maxParms,
1030                targetQOSByPri, minQOSByPri, userData);
1031          NEXTSTATE -;

1033    STATE ~;INPUT   MCS.Connect.Provider.response(mcId, result, parameters, userData);
1034          CALL    MCS_Connect_Provider_response(mcId, result, parameters, userData);
1035          NEXTSTATE -;

1037    STATE ~;INPUT   MCS.Disconnect.Provider.request(mcId);
1038          CALL    MCS_Disconnect_Provider_request(mcId);
1039          NEXTSTATE -;

1041    STATE ~;INPUT   MCS.Attach.User.request(label, localDomain);
1042          CALL    MCS_Attach_User_request(label, localDomain);
1043          NEXTSTATE -;

1045    STATE ~;INPUT   T.Connect.indication(tcId, remoteTSAP, localTSAP, offeredQOS,
minQOS);
1046          CALL    T_Connect_indication(tcId, remoteTSAP, localTSAP, offeredQOS, minQOS);
1047          NEXTSTATE -;

1049    STATE ~;INPUT   Connect.Initial(remoteDomain, localDomain, upward,
1050                targetParms, minParms, maxParms, userData);
1051          CALL    Connect_Initial(remoteDomain, localDomain, upward,
1052                targetParms, minParms, maxParms, userData);
1053          NEXTSTATE -;

1055    STATE ~;INPUT   Connect.Response(result, ccId, parameters, userData);
1056          CALL    Connect_Response(result, ccId, parameters, userData);
1057          NEXTSTATE -;

1059    STATE ~;INPUT   Connect.Additional(ccId, dp);
1060          CALL    Connect_Additional(ccId, dp);
1061          NEXTSTATE -;

1063    STATE ~;INPUT   Connect.Result(result);
1064          CALL    Connect_Result(result);
1065          NEXTSTATE -;

1067    STATE ~;INPUT   Drop.portal(p, reason);
1068          CALL    Drop_portal(p, reason);
1069          NEXTSTATE -;

1071    STATE ~;INPUT   Report.portal(p, diagnostic);
1072          CALL    Report_portal(p, diagnostic);
1073          NEXTSTATE -;

1075    STATE ~;INPUT   Shut.portal(p);
1076          CALL    Shut_portal(p);
1077          NEXTSTATE -;

1079    STATE ~;INPUT   RJum(pdu);
1080          CALL    RJum(pdu);
1081          NEXTSTATE -;

1083    STATE ~;INPUT   Quit;
1084          CALL    Quit;
```

```
1085          NEXTSTATE -;

1087    ENDPROCESS;
```

付録４　ドメイン プロセスのＳＤＬ記述

（ＪＴ－Ｔ１２５に対する）

```
1   PROCESS Domain;
  2   FPAR   parameters     DomainParameters;      /* values established in domain */

  4   SYNONYM maxBufferIds   Natural = EXTERNAL;    /* an implementation limit */

  6   TIMER  Time.portal(PortalId);               /* at most one timer per portal */

  8         /* Type definitions */

 10   NEWTYPE        ChannelStruct
 11   STRUCT
 12         kind           ChannelKind;    /* (Static,UserId,Private,Assigned) */
 13         joined         PortalIdSet;    /* directions where channel is joined */
 14         portal         PortalId;       /* if (UserId): the direction to it */
 15         manager        UserId;         /* if (Private): channel's manager */
 16         admitted       UserIdSet;      /* if (Private): zero or more users */
 17         uMerge         UserIdSet;      /* if (Private): still to be merged */
 18   ENDNEWTYPE;

 20   NEWTYPE        Chan              Array(ChannelId, ChannelStruct);
 21   ENDNEWTYPE;

 23   NEWTYPE        TokenStruct
 24   STRUCT
 25         kind           TokenKind;      /* (Grabbed,Inhibited,Giving,Ungivable,Given)
*/
 26         grabber        UserId;         /* if (Grabbed,Giving,Ungivable): user */
 27         recipient      UserId;         /* if (Giving,Given): an intended user */
 28         inhibitors     UserIdSet;      /* if (Inhibited): one or more users */
 29         uMerge         UserIdSet;      /* if (Inhibited): still to be merged */
 30   ENDNEWTYPE;

 32   NEWTYPE        Token             Array(TokenId, TokenStruct);
 33   ENDNEWTYPE;

 35   NEWTYPE        BooleanByPri       Array(DataPriority, Boolean);
 36   ENDNEWTYPE;
 37   NEWTYPE        NaturalByPri       Array(DataPriority, Natural);
 38   ENDNEWTYPE;
 39   NEWTYPE        BufferIdByPri      Array(DataPriority, BufferId);
 40   ENDNEWTYPE;
 41   NEWTYPE        BufferIdQueueByPri  Array(DataPriority, BufferIdQueue);
 42   ENDNEWTYPE;
 43   NEWTYPE        PortalIdSetByPri    Array(DataPriority, PortalIdSet);
 44   ENDNEWTYPE;

 46   NEWTYPE        NaturalByPriByKind  Array(PortalKind, NaturalByPri);
 47   ENDNEWTYPE;

 49   NEWTYPE        PortalStruct
 50   STRUCT
 51         kind           PortalKind;        /* (Attached,Downlink,Uplink) */
 52         pids           PIdByPri;          /* processes constituting a portal */
 53         inCredit       NaturalByPri;       /* permission to allocate inBuffer */
 54         inBuffer       BufferIdByPri;      /* PDU input coming from a process */
 55         outReady       BooleanByPri;       /* True if process allows an output */
 56         outQueue       BufferIdQueueByPri; /* PDUs awaiting output to process */
 57         outCount       Natural;           /* number queued for all priorities */
 58         outFlow        Natural;           /* number output since timer was set */
 59         elapsed        Duration;          /* interval set for portal timer */
 60         interval       Duration;          /* new interval to set for timer */
 61         subHeight      Natural;           /* subordinate's height or zero */
 62         subInterval    Duration;          /* subordinate's interval or zero */
 63   ENDNEWTYPE;

 65   NEWTYPE        Portal            Array(PortalId, PortalStruct);
 66   ENDNEWTYPE;

 68   NEWTYPE        PortalIdQueue      Queue(PortalId);
 69   ENDNEWTYPE;

 71   SYNTYPE        BufferId          = Integer CONSTANTS 0:maxBufferIds
 72   ENDSYNTYPE;
```

```
74   NEWTYPE        BufferIdSet        SetOf(BufferId);
75   ENDNEWTYPE;

77   NEWTYPE        BufferStruct
78   STRUCT
79         receiver       PortalId;      /* source of inCredit and input PDU */
80         dataPriority   DataPriority;  /* index into inBuffer and outQueue */
81         portals        Natural;       /* number of outQueues buffer is in */
82         pdu            PDUStruct;     /* the content of one domain MCSPDU */
83   ENDNEWTYPE;

85   NEWTYPE        Buffer             Array(BufferId, BufferStruct);
86   ENDNEWTYPE;

88   NEWTYPE        BufferIdQueue      Queue(BufferId);
89   ENDNEWTYPE;

91   GENERATOR      Queue (TYPE ItemType)   /* a first-in first-out queue */
92   LITERALS
93         EmptyQueue;
94   OPERATORS
95         Push:   ItemType, Queue -> Queue;      /* appends next item */
96         Next:   Queue           -> ItemType;   /* reveals first item */
97         Pull:   Queue           -> Queue;      /* deletes first item */
98   AXIOMS
99         Next(EmptyQueue) == ERROR!;
100        Pull(EmptyQueue) == ERROR!;
101        FOR ALL q IN Queue (
102        FOR ALL item IN ItemType
103        (
104              Next(Push(item,q)) == IF q = EmptyQueue THEN item
105                                    ELSE Next(q) FI;
106              Pull(Push(item,q)) == IF q = EmptyQueue THEN q
107                                    ELSE Push(item,Pull(q)) FI;
108        ));
109  DEFAULT
110        EmptyQueue;
111  ENDGENERATOR;

113        /* Data declarations */

115  DCL   control        PId;              /* the Control process */

117  DCL   upward         PortalId;         /* unique MCS Connection upward or */
118                                         /* zero if this provider is the top */

120  DCL   merging        Boolean,          /* True if domain is merging upward */
121        pdSend         Boolean,          /* True if PDin to be sent downward */
122        edSend         Boolean;          /* True if EDrq to be sent upward */

124  DCL   uMerge         UserIdSet,        /* users still to be merged */
125        uConfirm       UserIdSet,        /* users with merge confirmed */
126        cMerge         ChannelIdSet,     /* channels still to be merged */
127        cConfirm       ChannelIdSet,     /* channels with merge confirmed */
128        tMerge         TokenIdSet,       /* tokens still to be merged */
129        tConfirm       TokenIdSet,       /* tokens with merge confirmed */

131  DCL   mcrqQueue      PortalIdQueue,    /* origin of each pending MCrq */
132        mtrqQueue      PortalIdQueue,    /* origin of each pending MTrq */
133        aurqQueue      PortalIdQueue;    /* origin of each pending AUrq */

135  DCL   pDrop          PortalIdSet,      /* dropped portals needing DPum */
136        uDetach        UserIdSet,        /* disconnected users needing DUrq */
137        uRevoke        UserIdSet,        /* revoked token users needing DUrq */
138        cLeave         ChannelIdSet,     /* unjoined channels needing CLrq */
139        cDisband       ChannelIdSet,     /* unmanaged channels needing CDin */
140        tReject        TokenIdSet;       /* ungivable tokens needing TVcf */

142  DCL   pBufferWait    PortalIdSetByPri, /* portals requiring an inBuffer */
143        pInputWait     PortalIdSetByPri; /* inputs suspended during merge */

145  DCL   chan           Chan,             /* resource arrays */
146        token          Token,
147        portal         Portal,
148        buffer         Buffer;

150        /* Note: The fields of a chan, token, portal, or buffer
151        array element are undefined if the corresponding index
```

```
152         is not in cUsed, tUsed, pUsed, or bUsed, respectively. */

154   DCL   cUsed          ChannelIdSet,      /* indexes used */
155         tUsed          TokenIdSet,
156         pUsed          PortalIdSet,
157         bUsed          BufferIdSet;

159   DCL   cFree          ChannelIdSet,      /* indexes free */
160         tFree          TokenIdSet,
161         pFree          PortalIdSet,
162         bFree          BufferIdSet;

164   DCL   uUsed          UserIdSet;         /* subset of cUsed */

166   DCL   numChannelIds  Natural,           /* number in use */
167         numUserIds     Natural,
168         numTokenIds    Natural;

170   DCL   height         Natural,           /* current height of provider */
171         interval       Duration,          /* min throughput of one MCSPDU */
172         maxInterval    Duration;          /* maximum of portal intervals */

174   DCL   inCredit       NaturalByPriByKind; /* initializer */

176         /* Procedure decomposition */

178         /*     Initialize_resources
179               Take_user             (c, diagnostic)
180               Take_channel          (c, diagnostic)
181               Take_token            (t, diagnostic)
182               New_user              (u, result)
183               New_channel           (c, result)
184               Open_portal           (p, pKind, pids)
185               Time_portal           (p)
186               Drop_portal           (p, reason)
187               Shut_portal           (p)
188               Clean_queue           (p, queue)
189               Erect_domain
190               Identify_sender       (p, dp)
191               PDU_ready             (dp)
192               Input_PDU             (pdu)
193               Allocate_buffer       (b)
194               Cast_buffer           (b, p)
195               Release_buffer        (b)
196               Route_user            (u, p)
197               Multicast_buffer      (b, uSet)
198               Broadcast_buffer      (b)
199               Crank_engine
200               Drop_portals
201               Merge_users
202               Merge_channels
203               Merge_tokens
204               Detach_users
205               Leave_channels
206               Disband_channels
207               Reject_tokens
208               Process_PDU           (r, dp)
209               Validate_input        (r, b, diagnostic)
210               Top_provider          (r, b)
211               Apply_PDU             (r, b, diagnostic)
212               Token_status          (pdu)
213               Token_route           (u, x, p)
214               Delete_user           (u)
215               Delete_channel        (c)
216               Delete_token          (t)
217               Purge_users           (uSet)
218               Purge_channels        (cSet)
219               Purge_tokens          (tSet)
220               Output_buffer         (b, p)        */

222                                                   /*------------------------*/
223   PROCEDURE     Initialize_resources;                 /*  Initialize_resources
*/
224                                                   /*------------------------*/
225         DCL   c             ChannelId,
226               t             TokenId,
227               p             PortalId,
228               b             BufferId,
229               n             NaturalByPri;
```

```
230         START
231         COMMENT'Initialize data structures during process start-up
232               before accepting the first input signal.
233               Note that each SetOf automatically defaults to Empty
234               and each Queue to EmptyQueue.
235               Fixed buffer credits are an example; other values could
236               be computed from maxPortalIds and maxBufferIds.
237               ';
238         TASK    control := PARENT,
239               upward := 0,
240               merging := False,
241               pdSend := False,
242               edSend := False,
243               numChannelIds := 0,
244               numUserIds := 0,
245               numTokenIds := 0,
246               height := 1,
247               interval := IF parameters!minThroughput = 0 THEN 0 ELSE oneSecond *
248                (Float(parameters!maxMCSPDUsize) / Float(parameters!minThroughput))
FI,
249               maxInterval := 0,
250               c := 65535,
251               t := 65535,
252               p := maxPortalIds,
253               b := maxBufferIds;
254          1b :   /* for c = ?..1 */
255         DECISION c > 0;
256         (True): TASK    cFree := Incl(c, cFree),
257                       c := c - 1;
258               JOIN 1b;
259         ELSE:ENDDECISION;
260          2b :   /* for t = ?..1 */
261         TASK    tFree := Incl(t, tFree);
262         DECISION t > 1;
263         (True): TASK    t := t - 1;
264               JOIN 2b;
265         ELSE:ENDDECISION;
266          3b :   /* for p = ?..1 */
267         DECISION p > 0;
268         (True): TASK    pFree := Incl(p, pFree),
269                       p := p - 1;
270               JOIN 3b;
271         ELSE:ENDDECISION;
272          4b :   /* for b = ?..1 */
273         DECISION b > 0;
274         (True): TASK    bFree := Incl(b, bFree),
275                       b := b - 1;
276               JOIN 4b;
277         ELSE:ENDDECISION;
278         TASK    n(0) := 2, n(1) := 1, n(2) := 1, n(3) := 1,
279               inCredit(Attached) := n,
280               n(0) := 3, n(1) := 2, n(2) := 1, n(3) := 1,
281               inCredit(Downlink) := n,
282               n(0) := 4, n(1) := 3, n(2) := 2, n(3) := 1,
283               inCredit(Uplink) := n;
284         RETURN;
285         ENDPROCEDURE;

287                                                          /*-------------*/
288   PROCEDURE      Take_user;                              /*  Take_user  */
289   FPAR           c             ChannelId,                /*-------------*/
290         IN/OUT  diagnostic     Diagnostic;

292         DCL     u             UserId;
293         START
294         COMMENT'Put the free channel id to use as a user id.
295               ';
296         DECISION numUserIds < parameters!maxUserIds;
297         (False):TASK    diagnostic := DC_too_many_users;
298               RETURN;
299         ELSE:ENDDECISION;
300         CALL    Take_channel(c, diagnostic);
301         DECISION diagnostic = DC_OK;
302         (True): TASK    u := UserId(c),
303                       numUserIds := numUserIds + 1,
304                       uUsed := Incl(u, uUsed),
305                       chan(c)!kind := UserId;
306         ELSE:ENDDECISION;
307         RETURN;
```

```
308         ENDPROCEDURE;

310                                                                 /*----------------*/
311     PROCEDURE       Take_channel;                                  /*  Take_channel  */
312     FPAR           c            ChannelId,                        /*----------------*/
313           IN/OUT  diagnostic     Diagnostic;

315         START
316         COMMENT'Put the free channel id to unspecified use.
317                 ';
318         DECISION c in cFree;
319         (False):TASK    diagnostic := DC_channel_id_conflict;
320               RETURN;
321         ELSE:ENDDECISION;
322         DECISION numChannelIds < parameters!maxChannelIds;
323         (False):TASK    diagnostic := DC_too_many_channels;
324               RETURN;
325         ELSE:ENDDECISION;
326         TASK    diagnostic := DC_OK,
327               numChannelIds := numChannelIds + 1,
328               cFree := Del(c, cFree),
329               cUsed := Incl(c, cUsed),
330               chan(c)!joined := Empty,
331               chan(c)!admitted := Empty;
332         RETURN;
333         ENDPROCEDURE;

335                                                                 /*--------------*/
336     PROCEDURE       Take_token;                                    /*  Take_token  */
337     FPAR           t            TokenId,                          /*--------------*/
338           IN/OUT  diagnostic     Diagnostic;

340         START
341         COMMENT'Put the free token id to unspecified use.
342                 ';
343         DECISION t in tFree;
344         (False):TASK    diagnostic := DC_token_id_conflict;
345               RETURN;
346         ELSE:ENDDECISION;
347         DECISION numTokenIds < parameters!maxTokenIds;
348         (False):TASK    diagnostic := DC_too_many_tokens;
349               RETURN;
350         ELSE:ENDDECISION;
351         TASK    diagnostic := DC_OK,
352               numTokenIds := numTokenIds + 1,
353               tFree := Del(t, tFree),
354               tUsed := Incl(t, tUsed),
355               token(t)!inhibitors := Empty;
356         RETURN;
357         ENDPROCEDURE;

359                                                                 /*------------*/
360     PROCEDURE       New_user;                                      /*  New_user  */
361     FPAR    IN/OUT  u            UserId,                           /*------------*/
362           IN/OUT  result         Result;

364         DCL     c              ChannelId;
365         START
366         COMMENT'Allocate a new user id or fail and return 0.
367                 ';
368         TASK    u := 0;
369         DECISION numUserIds < parameters!maxUserIds;
370         (False):TASK    result := RT_too_many_users;
371               RETURN;
372         ELSE:ENDDECISION;
373         CALL    New_channel(c, result);
374         DECISION result = RT_successful;
375         (True): TASK    u := UserId(c),
376                         numUserIds := numUserIds + 1,
377                         uUsed := Incl(u, uUsed),
378                         chan(c)!kind := UserId;
379         ELSE:ENDDECISION;
380         RETURN;
381         ENDPROCEDURE;

383                                                                 /*---------------*/
384     PROCEDURE       New_channel;                                   /*  New_channel  */
385     FPAR    IN/OUT  c            ChannelId,                        /*---------------*/
386           IN/OUT  result         Result;
```

```
388         DCL     diagnostic      Diagnostic;
389         START
390         COMMENT'Allocate a new channel id or fail and return 0.
391                 ';
392         TASK    c := 0;
393         DECISION numChannelIds < parameters!maxChannelIds;
394         (False):TASK    result := RT_too_many_channels;
395                 RETURN;
396         ELSE:ENDDECISION;
397          1b :   /* keep trying */
398         TASK    c := ANY(ChannelId);    /* randomize */
399         DECISION c >= 1001 and c in cFree;
400         (False):JOIN 1b;
401         ELSE:ENDDECISION;
402         CALL    Take_channel(c, diagnostic);
403         TASK    result := RT_successful;
404         RETURN;
405         ENDPROCEDURE;

407                                                          /*---------------*/
408   PROCEDURE       Open_portal;                             /*  Open_portal  */
409   FPAR            p               PortalId,               /*---------------*/
410                   pKind           PortalKind,
411                   pids            PIdByPri;

413         DCL     pid             PId,
414                 dp              DataPriority,
415                 c               ChannelId,
416                 cSet            ChannelIdSet,
417                 t               TokenId,
418                 tSet            TokenIdSet;
419         START
420         COMMENT'Process an Open.portal input signal.
421                 Accept a new MCS connection or attachment to the domain.
422                 If this is an upward connection, prepare for merge.
423                 ';
424         DECISION pKind = Attached;
425         (True): TASK    pid := pids(0),
426                         pids(1) := pid,
427                         pids(2) := pid,
428                         pids(3) := pid;
429         ELSE:ENDDECISION;
430         TASK    pFree := Del(p, pFree),
431                 pUsed := Incl(p, pUsed),
432                 portal(p)!kind := pKind,
433                 portal(p)!pids := pids,
434                 portal(p)!inCredit := inCredit(pKind),
435                 portal(p)!outCount := 0,
436                 portal(p)!interval := IF pKind = Uplink THEN 0 ELSE interval FI,
437                 portal(p)!subHeight := 0,
438                 portal(p)!subInterval := 0,
439                 dp := 0;
440          1b :   /* for dp = 0..? */
441         DECISION dp < parameters!numPriorities;
442         (True): TASK    portal(p)!inBuffer(dp) := 0,
443                         portal(p)!outReady(dp) := False,
444                         portal(p)!outQueue(dp) := EmptyQueue,
445                         pBufferWait(dp) := Incl(p, pBufferWait(dp)),
446                         dp := dp + 1;
447                 JOIN 1b;
448         ELSE:ENDDECISION;
449         DECISION portal(p)!kind = Uplink;
450         (True): TASK    upward := p,
451                         merging := True,
452                         pdSend := True,
453                         edSend := True,
454                         uMerge := uUsed,
455                         uConfirm := Empty,
456                         cMerge := cUsed,
457                         cConfirm := Empty,
458                         tMerge := tUsed,
459                         tConfirm := Empty,
460                         cLeave := Empty,
461                         cDisband := Empty,
462                         tReject := Empty,
463                         cSet := cMerge;
464                  2b :   /* for c in cSet */
465                 DECISION cSet = Empty;
```

```
466                 (False):TASK    c := Pick(cSet),
467                                 cSet := Del(c, cSet),
468                                 chan(c)!uMerge := chan(c)!admitted;
469                       JOIN 2b;
470                 ELSE:ENDDECISION;
471                 TASK    tSet := tMerge;
472                  3b :   /* for t in tSet */
473                 DECISION tSet = Empty;
474                 (False):TASK    t := Pick(tSet),
475                                 tSet := Del(t, tSet),
476                                 token(t)!uMerge := token(t)!inhibitors;
477                       JOIN 3b;
478                 ELSE:ENDDECISION;
479           ELSE:ENDDECISION;
480           CALL    Erect_domain;
481           CALL    Crank_engine;
482           RETURN;
483           ENDPROCEDURE;

485                                                          /*---------------*/
486   PROCEDURE       Time_portal;                             /*  Time_portal  */
487   FPAR            p             PortalId;                /*---------------*/

489           START
490           COMMENT'Process a Time.portal input signal.
491                 Ensure that minimum throughput is maintained.
492                 Allow for the fact that outFlow takes integer steps.
493                 ';
494           DECISION portal(p)!elapsed > interval * (Float(portal(p)!outFlow) + 0.99);
495           (True): OUTPUT  Report.portal(p, DC_throughput_inadequate) TO control;
496           (False):TASK    portal(p)!outFlow := 0,
497                           portal(p)!elapsed := portal(p)!interval;
498                 SET(NOW + portal(p)!interval, Time.portal(p));
499           ENDDECISION;
500           CALL    Crank_engine;
501           RETURN;
502           ENDPROCEDURE;

504                                                          /*---------------*/
505   PROCEDURE       Drop_portal;                             /*  Drop_portal  */
506   FPAR            p             PortalId,                /*---------------*/
507                   reason        Reason;

509           START
510           COMMENT'Process a Drop.portal input signal,
511                 knowing the only reason is user-requested.
512                 ';
513           DECISION p in pUsed and portal(p)!kind /= Attached;
514           (True): TASK    pDrop := Incl(p, pDrop);
515           ELSE:ENDDECISION;
516           CALL    Crank_engine;
517           RETURN;
518           ENDPROCEDURE;

520                                                          /*---------------*/
521   PROCEDURE       Shut_portal;                             /*  Shut_portal  */
522   FPAR            p             PortalId;                /*---------------*/

524           DCL     dp            DataPriority,
525                   b             BufferId,
526                   bSet          BufferIdSet,
527                   c             ChannelId,
528                   cSet          ChannelIdSet,
529                   u             UserId;
530           START
531           COMMENT'Process a Shut.portal input signal.
532                 Remove the corresponding MCS connection or attachment.
533                 When the last one is gone, stop the domain process.
534                 ';
535           RESET(Time.portal(p));
536           TASK    pUsed := Del(p, pUsed),
537                   pFree := Incl(p, pFree),
538                   pDrop := Del(p, pDrop),
539                   dp := 0;
540            1b :   /* for dp = 0..? */
541           DECISION dp < parameters!numPriorities;
542           (True):  2b :   /* for b in outQueue(dp) */
543                   DECISION portal(p)!outQueue(dp) = EmptyQueue;
544                   (False):TASK    b := Next(portal(p)!outQueue(dp)),
```

```
545                         portal(p)!outQueue(dp) := Pull(portal(p)!outQueue(dp)),
546                         buffer(b)!portals := buffer(b)!portals - 1;
547                   CALL   Release_buffer(b);
548                   JOIN 2b;
549             ELSE:ENDDECISION;
550             TASK   b := portal(p)!inBuffer(dp),
551                    portal(p)!inBuffer(dp) := 0;
552             CALL   Release_buffer(b);
553             TASK   pBufferWait(dp) := Del(p, pBufferWait(dp)),
554                    pInputWait(dp) := Del(p, pInputWait(dp)),
555                    dp := dp + 1;
556             JOIN 1b;
557       ELSE:ENDDECISION;
558       TASK   bSet := bUsed;
559        3b :  /* for b in bSet */
560       DECISION bSet = Empty;
561       (False):TASK   b := Pick(bSet),
562                      bSet := Del(b, bSet);
563             DECISION buffer(b)!receiver = p;
564             (True): TASK   buffer(b)!receiver := 0;
565             ELSE:ENDDECISION;
566             JOIN 3b;
567       ELSE:ENDDECISION;
568       TASK   cSet := cUsed;
569        4b :  /* for c in cSet */
570       DECISION cSet = Empty;
571       (False):TASK   c := Pick(cSet),
572                      cSet := Del(c, cSet);
573             DECISION p in chan(c)!joined;
574             (True): TASK   chan(c)!joined := Del(p, chan(c)!joined);
575                    DECISION chan(c)!joined = Empty;
576                    (True): TASK   cLeave := Incl(c, cLeave);
577                    ELSE:ENDDECISION;
578             ELSE:ENDDECISION;
579             DECISION chan(c)!kind = UserId and chan(c)!portal = p;
580             (True): TASK   chan(c)!portal := 0,
581                            u := UserId(c),
582                            uDetach := Incl(u, uDetach),
583                            cLeave := Del(c, cLeave);
584             ELSE:ENDDECISION;
585             JOIN 4b;
586       ELSE:ENDDECISION;
587       DECISION portal(p)!kind = Uplink;
588       (True): TASK   upward := 0,
589                      merging := False,
590                      mcrqQueue := EmptyQueue,
591                      mtrqQueue := EmptyQueue,
592                      aurqQueue := EmptyQueue;
593       (False):CALL   Clean_queue(p, mcrqQueue);
594             CALL   Clean_queue(p, mtrqQueue);
595             CALL   Clean_queue(p, aurqQueue);
596       ENDDECISION;
597       OUTPUT  Shut.portal(p) TO control;
598       CALL   Erect_domain;
599       CALL   Crank_engine;
600       RETURN;
601       ENDPROCEDURE;

603                                                          /*---------------*/
604 PROCEDURE      Clean_queue;                                 /*  Clean_queue  */
605 FPAR           p              PortalId,                     /*---------------*/
606       IN/OUT   queue          PortalIdQueue;

608       DCL      x              PortalId,
609                clean          PortalIdQueue;
610       START
611       COMMENT'Remove a shut portal id from a queue.
612               ';
613       TASK   clean := EmptyQueue;
614        1b :  /* for x in queue */
615       DECISION queue = EmptyQueue;
616       (False):TASK   x := Next(queue),
617                      queue := Pull(queue),
618                      x := IF x = p THEN 0 ELSE x FI,
619                      clean := Push(x, clean);
620             JOIN 1b;
621       ELSE:ENDDECISION;
622       TASK   queue := clean;
623       RETURN;
```

```
624         ENDPROCEDURE;

626                                                          /*----------------*/
627   PROCEDURE      Erect_domain;                             /*  Erect_domain  */
628                                                          /*----------------*/

630         DCL     p             PortalId,
631                 pSet          PortalIdSet,
632                 h             Natural,
633                 hmax          Natural,
634                 i             Duration,
635                 imax          Duration;
636         START
637         COMMENT'Recalculate the provider height and maxInterval.
638                 If either changed, communicate them upward.
639                 ';
640         TASK    hmax := 1,
641                 imax := 0,
642                 pSet := pUsed;
643          1b :   /* for p in pSet */
644         DECISION pSet = Empty;
645         (False):TASK    p := Pick(pSet),
646                         pSet := Del(p, pSet),
647                         h := portal(p)!subHeight + 1,
648                         hmax := IF h > hmax THEN h ELSE hmax FI,
649                         i := portal(p)!interval,
650                         imax := IF i > imax THEN i ELSE imax FI;
651                 JOIN 1b;
652         ELSE:ENDDECISION;
653         DECISION height = hmax and maxInterval = imax;
654         (False):TASK    height := hmax,
655                         maxInterval := imax,
656                         edSend := True;
657         ELSE:ENDDECISION;
658         RETURN;
659         ENDPROCEDURE;

661                                                          /*--------------------*/
662   PROCEDURE      Identify_sender;                          /*  Identify_sender  */
663   FPAR    IN/OUT  p             PortalId,                  /*--------------------*/
664           IN/OUT  dp            DataPriority;

666         DCL     pSet          PortalIdSet;
667         START
668         COMMENT'An alternative would be to carry this
669                 information explicitly in SDL signals.
670                 ';
671         TASK    pSet := pUsed;
672          1b :   /* for p in pSet */
673         DECISION pSet = Empty;
674         (False):TASK    p := Pick(pSet),
675                         pSet := Del(p, pSet),
676                         dp := 0;
677                  2b :   /* for dp = 0..? */
678                 DECISION dp < parameters!numPriorities;
679                 (True): DECISION portal(p)!pids(dp) = SENDER;
680                         (True): RETURN;
681                         ELSE:ENDDECISION;
682                         TASK    dp := dp + 1;
683                         JOIN 2b;
684                 ELSE:ENDDECISION;
685                 JOIN 1b;
686         ELSE:ENDDECISION;
687         TASK    p := 0,
688                 dp := 0;
689         RETURN;
690         ENDPROCEDURE;

692                                                          /*--------------*/
693   PROCEDURE      PDU_ready;                                /*  PDU_ready  */
694   FPAR           dp            DataPriority;             /*--------------*/
695         DCL     p             PortalId,
696                 x             DataPriority,
697                 b             BufferId;
698         START
699         COMMENT'Process a PDU.ready input signal.
700                 If a buffer is waiting, it can be output.
701                 ';
702         CALL    Identify_sender(p, x);
```

```
703         DECISION p in pUsed;
704         (False):RETURN;
705         ELSE:ENDDECISION;
706         DECISION portal(p)!outQueue(dp) = EmptyQueue;
707         (True): TASK    portal(p)!outReady(dp) := True;
708         (False):TASK    b := Next(portal(p)!outQueue(dp)),
709                         portal(p)!outQueue(dp) := Pull(portal(p)!outQueue(dp)),
710                         buffer(b)!portals := buffer(b)!portals - 1;
711                 CALL    Output_buffer(b, p);
712                 TASK    portal(p)!outReady(dp) := False;
713                 CALL    Release_buffer(b);
714                 TASK    portal(p)!outFlow := portal(p)!outFlow + 1,
715                         portal(p)!outCount := portal(p)!outCount - 1;
716                 DECISION portal(p)!outCount = 0;
717                 (True): RESET(Time.portal(p));
718                 ELSE:ENDDECISION;
719         ENDDECISION;
720         CALL    Crank_engine;
721         RETURN;
722         ENDPROCEDURE;

724                                                                 /*-------------*/
725   PROCEDURE       Input_PDU;                                      /* Input_PDU */
726   FPAR            pdu             PDUStruct;                      /*-------------*/

728         DCL     p               PortalId,
729                 dp              DataPriority,
730                 b               BufferId;
731         START
732         COMMENT'Process an MCSPDU input signal.
733                 If merging, requests and responses must wait.
734                 ';
735         CALL    Identify_sender(p, dp);
736         DECISION p in pUsed;
737         (False):RETURN;
738         ELSE:ENDDECISION;
739         DECISION portal(p)!kind = Attached;
740         (True): DECISION pdu!kind;
741                 (SDrq, SDin, USrq, USin):
742                         TASK    dp := pdu!dataPriority,
743                                 dp := IF dp < parameters!numPriorities THEN dp
744                                                 ELSE parameters!numPriorities - 1 FI;
745                 ELSE:ENDDECISION;
746         ELSE:ENDDECISION;
747         TASK    b := portal(p)!inBuffer(dp),
748                 buffer(b)!pdu := pdu;
749         DECISION p = upward or not merging;
750         (True): CALL    Process_PDU(p, dp);
751         (False):TASK    pInputWait(dp) := Incl(p, pInputWait(dp));
752         ENDDECISION;
753         CALL    Crank_engine;
754         RETURN;
755         ENDPROCEDURE;

757                                                                 /*-------------------*/
758   PROCEDURE       Allocate_buffer;                                /* Allocate_buffer */
759   FPAR    IN/OUT  b               BufferId;                       /*-------------------*/

761         START
762         COMMENT'Allocate a free buffer id or fail and return 0.
763                 ';
764         DECISION b /= 0 and buffer(b)!portals = 0;
765         (True): RETURN;
766         ELSE:ENDDECISION;
767         TASK    b := 0;
768         DECISION bFree = Empty;
769         (False):TASK    b := Pick(bFree),
770                         bFree := Del(b, bFree),
771                         bUsed := Incl(b, bUsed),
772                         buffer(b)!receiver := 0,
773                         buffer(b)!dataPriority := 0,
774                         buffer(b)!portals := 0;
775         ELSE:ENDDECISION;
776         RETURN;
777         ENDPROCEDURE;

779                                                                 /*---------------*/
780   PROCEDURE       Cast_buffer;                                    /* Cast_buffer */
781   FPAR            b               BufferId,                       /*---------------*/
```

```
782                 p               PortalId;

784         DCL     dp              DataPriority;
785         START
786         COMMENT'Send buffer containing an MCSPDU to a single output.
787                 ';
788         DECISION p = 0;
789         (True): RETURN;
790         ELSE:ENDDECISION;
791         TASK    dp := buffer(b)!dataPriority;
792         DECISION portal(p)!outReady(dp);
793         (True): CALL    Output_buffer(b, p);
794                 TASK    portal(p)!outReady(dp) := False;
795         (False):DECISION portal(p)!outCount = 0 and portal(p)!interval > 0;
796                 (True): TASK    portal(p)!outFlow := 0,
797                                 portal(p)!elapsed := portal(p)!interval;
798                         SET(NOW + portal(p)!interval, Time.portal(p));
799                 ELSE:ENDDECISION;
800                 TASK    portal(p)!outQueue(dp) := Push(b, portal(p)!outQueue(dp)),
801                         buffer(b)!portals := buffer(b)!portals + 1,
802                         portal(p)!outCount := portal(p)!outCount + 1;
803         ENDDECISION;
804         RETURN;
805         ENDPROCEDURE;

807                                                         /*------------------*/
808 PROCEDURE       Release_buffer;                           /* Release_buffer */
809 FPAR            b               BufferId;                /*------------------*/

811         DCL     p               PortalId,
812                 dp              DataPriority;
813         START
814         COMMENT'Release a buffer that is no longer needed.
815                 ';
816         DECISION b = 0 or buffer(b)!portals > 0;
817         (True): RETURN;
818         ELSE:ENDDECISION;
819         TASK    bUsed := Del(b, bUsed),
820                 bFree := Incl(b, bFree),
821                 p := buffer(b)!receiver,
822                 dp := buffer(b)!dataPriority;
823         DECISION p = 0;
824         (False):TASK    portal(p)!inCredit(dp) := portal(p)!inCredit(dp) + 1;
825                 DECISION portal(p)!inBuffer(dp) = 0;
826                 (True): TASK    pBufferWait(dp) := Incl(p, pBufferWait(dp));
827                 ELSE:ENDDECISION;
828         ELSE:ENDDECISION;
829         RETURN;
830         ENDPROCEDURE;

832                                                         /*--------------*/
833 PROCEDURE       Route_user;                               /* Route_user */
834 FPAR            u               UserId,                  /*--------------*/
835         IN/OUT  p               PortalId;

837         DCL     c               ChannelId;
838         START
839         COMMENT'Return the portal id that leads toward a user.
840                 ';
841         TASK    p := 0;
842         DECISION u in uUsed;
843         (True): TASK    c := ChannelId(u),
844                         p := chan(c)!portal;
845         ELSE:ENDDECISION;
846         RETURN;
847         ENDPROCEDURE;

849                                                         /*--------------------*/
850 PROCEDURE       Multicast_buffer;                         /* Multicast_buffer */
851 FPAR            b               BufferId,                /*--------------------*/
852                 uSet            UserIdSet;

854         DCL     u               UserId,
855                 p               PortalId,
856                 pSet            PortalIdSet;
857         START
858         COMMENT'Send buffer containing an MCSPDU to multiple users.
859                 ';
860         TASK    pSet := Empty;
```

```
861          1b :   /* for u in uSet */
862          DECISION uSet = Empty;
863          (False):TASK    u := Pick(uSet),
864                          uSet := Del(u, uSet);
865                CALL    Route_user(u, p);
866                TASK    pSet := Incl(p, pSet);
867                JOIN 1b;
868          ELSE:ENDDECISION;
869           2b :   /* for p in pSet */
870          DECISION pSet = Empty;
871          (False):TASK    p := Pick(pSet),
872                          pSet := Del(p, pSet);
873                CALL    Cast_buffer(b, p);
874                JOIN 2b;
875          ELSE:ENDDECISION;
876          RETURN;
877          ENDPROCEDURE;

879                                                              /*--------------------*/
880    PROCEDURE      Broadcast_buffer;                            /* Broadcast_buffer */
881    FPAR           b              BufferId;                   /*--------------------*/

883          DCL     pdu             PDUStruct,
884                  p               PortalId,
885                  pSet            PortalIdSet;
886          START
887          COMMENT'Send buffer containing an MCSPDU to the whole subtree.
888                  ';
889          TASK    pdu := buffer(b)!pdu;
890          DECISION (pdu!kind = PCin and pdu!detachUserIds = Empty
891                                    and pdu!purgeChannelIds = Empty)
892                or (pdu!kind = PTin and pdu!purgeTokenIds = Empty)
893                or (pdu!kind = DUin and pdu!userIds = Empty);
894          (True): RETURN;
895          ELSE:ENDDECISION;
896          TASK    pSet := pUsed;
897           1b :   /* for p in pSet */
898          DECISION pSet = Empty;
899          (False):TASK    p := Pick(pSet),
900                          pSet := Del(p, pSet);
901                DECISION portal(p)!kind;
902                (Attached):
903                      DECISION pdu!kind = PDin;
904                      (False):CALL    Cast_buffer(b, p);
905                      ELSE:ENDDECISION;
906                (Downlink):
907                      CALL    Cast_buffer(b, p);
908                ELSE:ENDDECISION;
909                JOIN 1b;
910          ELSE:ENDDECISION;
911          RETURN;
912          ENDPROCEDURE;

914                                                              /*----------------*/
915    PROCEDURE      Crank_engine;                                /* Crank_engine */
916                                                              /*----------------*/
917          DCL     b               BufferId,
918                  p               PortalId,
919                  dp              DataPriority;
920          START
921          COMMENT'After individual processing of each input signal,
922                  look for the most deserving work to do next.
923                  This advances incrementally through stages of a merge;
924                  else it cleans up users, channels, tokens left behind.
925                  Buffers are assigned to accept new inputs, with preference
926                  to PDUs flowing downward and to higher priorities first.
927                  ';
928          TASK    b := 0;
929          DECISION pdSend;
930          (True): CALL    Allocate_buffer(b);
931                DECISION b = 0;
932                (True): RETURN;
933                ELSE:ENDDECISION;
934                TASK    pdSend := False,
935                        buffer(b)!pdu!kind := PDin,
936                        buffer(b)!pdu!heightLimit := parameters!maxHeight - 1;
937                CALL    Broadcast_buffer(b);
938          ELSE:ENDDECISION;
939          DECISION edSend;
```

```
940         (True): CALL    Allocate_buffer(b);
941                 DECISION b = 0;
942                 (True): RETURN;
943                 ELSE:ENDDECISION;
944                 TASK    edSend := False,
945                         buffer(b)!pdu!kind := EDrq,
946                         buffer(b)!pdu!subHeight := height,
947                         buffer(b)!pdu!subInterval := maxInterval;
948                 CALL    Cast_buffer(b, upward);
949         ELSE:ENDDECISION;
950         CALL    Release_buffer(b);
951         TASK    dp := 0;
952          1b :   /* for dp = 0..? */
953         DECISION dp < parameters!numPriorities;
954         (True): DECISION upward in pBufferWait(dp);
955                 (False):TASK    dp := dp + 1;
956                         JOIN 1b;
957                 ELSE:ENDDECISION;
958                 TASK    p := upward,
959                         b := 0;
960                 CALL    Allocate_buffer(b);
961                 DECISION b = 0;
962                 (True): RETURN;
963                 ELSE:ENDDECISION;
964                 TASK    pBufferWait(dp) := Del(p, pBufferWait(dp)),
965                         buffer(b)!receiver := p,
966                         buffer(b)!dataPriority := dp,
967                         portal(p)!inCredit(dp) := portal(p)!inCredit(dp) - 1,
968                         portal(p)!inBuffer(dp) := b;
969                 OUTPUT  PDU.ready(dp) TO portal(p)!pids(dp);
970                 JOIN 1b;
971         ELSE:ENDDECISION;
972         CALL    Drop_portals;
973         DECISION merging;
974         (True): CALL    Merge_users;
975                 DECISION uConfirm = uUsed;
976                 (True): CALL    Merge_tokens;
977                         DECISION tConfirm = tUsed;
978                         (True): CALL    Merge_channels;
979                                 DECISION cConfirm = cUsed;
980                                 (True): TASK    merging := False;
981                                 ELSE:ENDDECISION;
982                         ELSE:ENDDECISION;
983                 ELSE:ENDDECISION;
984                 DECISION merging;
985                 (True): RETURN;
986                 (False):TASK    dp := 0;
987                          2b :   /* for dp = 0..? */
988                         DECISION dp < parameters!numPriorities;
989                         (True): /* for p in pInputWait(dp) */
990                                 DECISION pInputWait(dp) = Empty;
991                                 (True): TASK    dp := dp + 1;
992                                         JOIN 2b;
993                                 ELSE:ENDDECISION;
994                                 TASK    p := Pick(pInputWait(dp)),
995                                         pInputWait(dp) := Del(p, pInputWait(dp));
996                                 CALL    Process_PDU(p, dp);
997                                 JOIN 2b;
998                         ELSE:ENDDECISION;
999                 ENDDECISION;
1000        ELSE:ENDDECISION;
1001        CALL    Detach_users;
1002        CALL    Leave_channels;
1003        CALL    Disband_channels;
1004        CALL    Reject_tokens;
1005        TASK    dp := 0;
1006         3b :   /* for dp = 0..? */
1007        DECISION dp < parameters!numPriorities;
1008        (True): /* for p in pBufferWait(dp) */
1009                DECISION pBufferWait(dp) = Empty;
1010                (True): TASK    dp := dp + 1;
1011                        JOIN 3b;
1012                ELSE:ENDDECISION;
1013                TASK    p := Pick(pBufferWait(dp)),
1014                        b := 0;
1015                CALL    Allocate_buffer(b);
1016                DECISION b = 0;
1017                (True): RETURN;
1018                ELSE:ENDDECISION;
```

```
1019                TASK   pBufferWait(dp) := Del(p, pBufferWait(dp)),
1020                       buffer(b)!receiver := p,
1021                       buffer(b)!dataPriority := dp,
1022                       portal(p)!inCredit(dp) := portal(p)!inCredit(dp) - 1,
1023                       portal(p)!inBuffer(dp) := b;
1024                OUTPUT PDU.ready(dp) TO portal(p)!pids(dp);
1025                JOIN 3b;
1026          ELSE:ENDDECISION;
1027          RETURN;
1028          ENDPROCEDURE;

1030                                                                  /*----------------*/
1031    PROCEDURE      Drop_portals;                                     /* Drop_portals */
1032                                                                  /*----------------*/
1033          DCL    b             BufferId,
1034                 p             PortalId,
1035                 pdu           PDUStruct;
1036          START
1037          COMMENT'Generate DPum MCSPDUs requested by controller.
1038                 ';
1039          TASK   b := 0,
1040                 pdu!kind := DPum,
1041                 pdu!reason := RN_user_requested;
1042           1b :  /* for p in pDrop */
1043          DECISION pDrop = Empty;
1044          (False):CALL   Allocate_buffer(b);
1045                 DECISION b = 0;
1046                 (True): RETURN;
1047                 ELSE:ENDDECISION;
1048                 TASK   p := Pick(pDrop),
1049                        pDrop := Del(p, pDrop),
1050                        buffer(b)!pdu := pdu;
1051                 CALL   Cast_buffer(b, p);
1052                 JOIN 1b;
1053          ELSE:ENDDECISION;
1054          CALL   Release_buffer(b);
1055          RETURN;
1056          ENDPROCEDURE;

1058                                                                  /*---------------*/
1059    PROCEDURE      Merge_users;                                      /* Merge_users */
1060                                                                  /*---------------*/
1061          DCL    b             BufferId,
1062                 u             UserId,
1063                 c             ChannelId,
1064                 pdu           PDUStruct,
1065                 cAttr         ChannelAttributes;
1066          START
1067          COMMENT'Generate MCrq MCSPDUs to merge user ids upward.
1068                 Fill them to maximum size that encoding allows.
1069                 ';
1070          TASK   b := 0,
1071                 pdu!kind := MCrq,
1072                 pdu!mergeChannels := Empty;
1073           1b :  /* for u in uMerge */
1074          DECISION uMerge = Empty;
1075          (False):CALL   Allocate_buffer(b);
1076                 DECISION b = 0;
1077                 (True): RETURN;
1078                 ELSE:ENDDECISION;
1079                 TASK   u := Pick(uMerge),
1080                        uMerge := Del(u, uMerge),
1081                        c := ChannelId(u),
1082                        cMerge := Del(c, cMerge),
1083                        cAttr!channelId := c,
1084                        cAttr!kind := UserId,
1085                        cAttr!joined := (chan(c)!joined /= Empty),
1086                        pdu!mergeChannels := Incl(cAttr, pdu!mergeChannels);
1087                 DECISION'encoding of pdu + 9 <= maxMCSPDUsize';
1088                 ('True'):JOIN 1b;
1089                 ELSE:ENDDECISION;
1090          ELSE:ENDDECISION;
1091          DECISION pdu!mergeChannels = Empty;
1092          (False):TASK   buffer(b)!pdu := pdu,
1093                         pdu!mergeChannels := Empty;
1094                 CALL   Cast_buffer(b, upward);
1095                 JOIN 1b;
1096          ELSE:ENDDECISION;
1097          CALL   Release_buffer(b);
```

```
1098        RETURN;
1099        ENDPROCEDURE;

1101                                                          /*------------------*/
1102  PROCEDURE      Merge_channels;                             /* Merge_channels  */
1103                                                          /*------------------*/
1104        DCL     b             BufferId,
1105                c             ChannelId,
1106                u             UserId,
1107                pdu           PDUStruct,
1108                cAttr         ChannelAttributes;
1109        START
1110        COMMENT'Generate MCrq MCSPDUs to merge channel ids upward.
1111                Fill them to maximum size that encoding allows.
1112                ';
1113        TASK    b := 0,
1114                pdu!kind := MCrq,
1115                pdu!mergeChannels := Empty,
1116                pdu!purgeChannelIds := Empty;
1117         1b :   /* for c in cMerge */
1118        DECISION cMerge = Empty;
1119        (False):CALL    Allocate_buffer(b);
1120                DECISION b = 0;
1121                (True): RETURN;
1122                ELSE:ENDDECISION;
1123                TASK    c := Pick(cMerge),
1124                        cMerge := Del(c, cMerge),
1125                        cAttr!channelId := c,
1126                        cAttr!kind := chan(c)!kind,
1127                        cAttr!manager := chan(c)!manager,
1128                        cAttr!admitted := Empty,
1129                        cAttr!joined := (chan(c)!joined /= Empty);
1130                DECISION (chan(c)!kind = Static or chan(c)!kind = Assigned)
1131                        and chan(c)!joined = Empty;
1132                (True): CALL    Delete_channel(c);
1133                        JOIN 1b;
1134                ELSE:ENDDECISION;
1135                DECISION chan(c)!kind = Private;
1136                (True): DECISION chan(c)!manager in uUsed;
1137                        (False):TASK    pdu!purgeChannelIds :=
1138                                                Incl(c, pdu!purgeChannelIds);
1139                                JOIN 3f;
1140                        ELSE:ENDDECISION;
1141                         2b :   /* for u in chan(c)!uMerge */
1142                        DECISION chan(c)!uMerge = Empty;
1143                        (False):TASK    u := Pick(chan(c)!uMerge),
1144                                        chan(c)!uMerge := Del(u, chan(c)!uMerge),
1145                                        cAttr!admitted := Incl(u, cAttr!admitted),
1146                                        pdu!mergeChannels :=      /* try this many */
1147                                                Incl(cAttr, pdu!mergeChannels);
1148                                DECISION'encoding of pdu + 4 <= maxMCSPDUsize';
1149                                ('True'):TASK   pdu!mergeChannels := /* try another */
1150                                                Del(cAttr, pdu!mergeChannels);
1151                                        JOIN 2b;
1152                                ELSE:ENDDECISION;
1153                        ELSE:ENDDECISION;
1154                ELSE:ENDDECISION;
1155                TASK    pdu!mergeChannels := Incl(cAttr, pdu!mergeChannels);
1156                 3f :
1157                DECISION'encoding of pdu + 23 <= maxMCSPDUsize';
1158                ('True'):JOIN 1b;
1159                ELSE:ENDDECISION;
1160        ELSE:ENDDECISION;
1161        DECISION pdu!mergeChannels = Empty and pdu!purgeChannelIds = Empty;
1162        (False):TASK    buffer(b)!pdu := pdu,
1163                        pdu!mergeChannels := Empty,
1164                        pdu!purgeChannelIds := Empty;
1165                CALL    Cast_buffer(b, upward);
1166                JOIN 1b;
1167        ELSE:ENDDECISION;
1168        CALL    Release_buffer(b);
1169        RETURN;
1170        ENDPROCEDURE;

1172                                                          /*----------------*/
1173  PROCEDURE      Merge_tokens;                               /* Merge_tokens  */
1174                                                          /*----------------*/
1175        DCL     b             BufferId,
1176                t             TokenId,
```

```
1177                u              UserId,
1178                pdu            PDUStruct,
1179                tAttr          TokenAttributes;
1180          START
1181          COMMENT'Generate MTrq MCSPDUs to merge token ids upward.
1182                Fill them to maximum size that encoding allows.
1183                ';
1184          TASK    b := 0,
1185                pdu!kind := MTrq,
1186                pdu!mergeTokens := Empty,
1187                pdu!purgeTokenIds := Empty,
1188           1b :   /* for t in tMerge */
1189          DECISION tMerge = Empty;
1190          (False):CALL    Allocate_buffer(b);
1191                DECISION b = 0;
1192                (True): RETURN;
1193                ELSE:ENDDECISION;
1194                TASK    t := Pick(tMerge),
1195                      tMerge := Del(t, tMerge),
1196                      tAttr!tokenId := t,
1197                      tAttr!kind := token(t)!kind,
1198                      tAttr!grabber := token(t)!grabber,
1199                      tAttr!recipient := token(t)!recipient,
1200                      tAttr!inhibitors := Empty;
1201                DECISION token(t)!kind = Inhibited;
1202                (True):  2b :   /* for u in token(t)!uMerge */
1203                      DECISION token(t)!uMerge = Empty;
1204                      (False):TASK    u := Pick(token(t)!uMerge),
1205                                  token(t)!uMerge := Del(u, token(t)!uMerge),
1206                                  tAttr!inhibitors := Incl(u, tAttr!inhibitors),
1207                                  pdu!mergeTokens :=      /* try this many */
1208                                        Incl(tAttr, pdu!mergeTokens);
1209                            DECISION'encoding of pdu + 4 <= maxMCSPDUsize';
1210                            ('True'):TASK   pdu!mergeTokens := /* try another */
1211                                        Del(tAttr, pdu!mergeTokens);
1212                                  JOIN 2b;
1213                            ELSE:ENDDECISION;
1214                      ELSE:ENDDECISION;
1215                ELSE:ENDDECISION;
1216                TASK    pdu!mergeTokens := Incl(tAttr, pdu!mergeTokens);
1217                DECISION'encoding of pdu + 16 <= maxMCSPDUsize';
1218                ('True'):JOIN 1b;
1219                ELSE:ENDDECISION;
1220          ELSE:ENDDECISION;
1221          DECISION pdu!mergeTokens = Empty;
1222          (False):TASK    buffer(b)!pdu := pdu,
1223                      pdu!mergeTokens := Empty;
1224                CALL    Cast_buffer(b, upward);
1225                JOIN 1b;
1226          ELSE:ENDDECISION;
1227          CALL    Release_buffer(b);
1228          RETURN;
1229          ENDPROCEDURE;

1231                                                          /*----------------*/
1232    PROCEDURE      Detach_users;                           /*  Detach_users  */
1233                                                          /*----------------*/
1234          DCL     b              BufferId,
1235                u              UserId,
1236                pdu            PDUStruct,
1237                diagnostic     Diagnostic;
1238          START
1239          COMMENT'Generate DUrq MCSPDUs to detach users.
1240                Fill them to maximum size that encoding allows.
1241                ';
1242          TASK    b := 0,
1243                pdu!kind := DUrq,
1244                pdu!userIds := Empty;
1245           1b :   /* for u in uDetach */
1246          DECISION uDetach = Empty;
1247          (False):CALL    Allocate_buffer(b);
1248                DECISION b = 0;
1249                (True): RETURN;
1250                ELSE:ENDDECISION;
1251                TASK    u := Pick(uDetach),
1252                      uDetach := Del(u, uDetach),
1253                      uRevoke := Del(u, uRevoke),
1254                      pdu!reason := RN_domain_disconnected,
1255                      pdu!userIds := Incl(u, pdu!userIds);
```

```
1256                DECISION'encoding of pdu + 4 <= maxMCSPDUsize';
1257                ('True'):JOIN 1b;
1258                ELSE:ENDDECISION;
1259          ELSE:ENDDECISION;
1260          DECISION pdu!userIds = Empty;
1261          (False):JOIN 3f;
1262          ELSE:ENDDECISION;
1263           2b :   /* for u in uRevoke */
1264          DECISION uRevoke = Empty;
1265          (False):CALL    Allocate_buffer(b);
1266                DECISION b = 0;
1267                (True): RETURN;
1268                ELSE:ENDDECISION;
1269                TASK    u := Pick(uRevoke),
1270                      uRevoke := Del(u, uRevoke),
1271                      pdu!reason := RN_channel_purged,
1272                      pdu!userIds := Incl(u, pdu!userIds);
1273                DECISION'encoding of pdu + 4 <= maxMCSPDUsize';
1274                ('True'):JOIN 1b;
1275                ELSE:ENDDECISION;
1276          ELSE:ENDDECISION;
1277           3f :
1278          DECISION pdu!userIds = Empty;
1279          (False):TASK    buffer(b)!pdu := pdu,
1280                      pdu!userIds := Empty;
1281                DECISION upward = 0;
1282                (False):CALL    Cast_buffer(b, upward);
1283                (True): TASK    buffer(b)!pdu!kind := DUin;
1284                      CALL    Apply_PDU(0, b, diagnostic);
1285                ENDDECISION;
1286                JOIN 1b;
1287          ELSE:ENDDECISION;
1288          CALL    Release_buffer(b);
1289          RETURN;
1290          ENDPROCEDURE;

1292                                                          /*------------------*/
1293    PROCEDURE       Leave_channels;                          /*  Leave_channels  */
1294                                                          /*------------------*/
1295          DCL     b             BufferId,
1296                  c             ChannelId,
1297                  pdu           PDUStruct;
1298          START
1299          COMMENT'Generate CLrq MCSPDUs to leave channels.
1300                Fill them to maximum size that encoding allows.
1301                ';
1302          TASK    b := 0,
1303                pdu!kind := CLrq,
1304                pdu!channelIds := Empty;
1305           1b :   /* for c in cLeave */
1306          DECISION cLeave = Empty;
1307          (False):CALL    Allocate_buffer(b);
1308                DECISION b = 0;
1309                (True): RETURN;
1310                ELSE:ENDDECISION;
1311                TASK    c := Pick(cLeave),
1312                      cLeave := Del(c, cLeave);
1313                DECISION chan(c)!kind;
1314                (Static, Assigned):
1315                      CALL    Delete_channel(c);
1316                ELSE:ENDDECISION;
1317                TASK    pdu!channelIds := Incl(c, pdu!channelIds);
1318                DECISION'encoding of pdu + 4 <= maxMCSPDUsize';
1319                ('True'):JOIN 1b;
1320                ELSE:ENDDECISION;
1321          ELSE:ENDDECISION;
1322          DECISION pdu!channelIds = Empty;
1323          (False):TASK    buffer(b)!pdu := pdu,
1324                      pdu!channelIds := Empty;
1325                CALL    Cast_buffer(b, upward);
1326                JOIN 1b;
1327          ELSE:ENDDECISION;
1328          CALL    Release_buffer(b);
1329          RETURN;
1330          ENDPROCEDURE;

1332                                                          /*--------------------*/
1333    PROCEDURE       Disband_channels;                        /*  Disband_channels  */
1334                                                          /*--------------------*/
```

```
1335        DCL     b               BufferId,
1336                c               ChannelId,
1337                pdu             PDUStruct,
1338                diagnostic      Diagnostic;
1339        START
1340        COMMENT'Generate CDrq MCSPDUs to disband channels.
1341                ';
1342        TASK    b := 0,
1343                pdu!kind := CDrq;
1344         1b :   /* for c in cDisband */
1345        DECISION cDisband = Empty;
1346        (False):CALL    Allocate_buffer(b);
1347                DECISION b = 0;
1348                (True): RETURN;
1349                ELSE:ENDDECISION;
1350                TASK    c := Pick(cDisband),
1351                        cDisband := Del(c, cDisband),
1352                        pdu!channelId := c,
1353                        buffer(b)!pdu := pdu;
1354                DECISION upward = 0;
1355                (False):CALL    Cast_buffer(b, upward);
1356                (True): TASK    buffer(b)!pdu!kind := CDin;
1357                        CALL    Apply_PDU(0, b, diagnostic);
1358                ENDDECISION;
1359                JOIN 1b;
1360        ELSE:ENDDECISION;
1361        CALL    Release_buffer(b);
1362        RETURN;
1363        ENDPROCEDURE;

1365                                                        /*-----------------*/
1366    PROCEDURE       Reject_tokens;                        /* Reject_tokens  */
1367                                                        /*-----------------*/
1368        DCL     b               BufferId,
1369                t               TokenId,
1370                u               UserId,
1371                p               PortalId,
1372                pdu             PDUStruct;
1373        START
1374        COMMENT'Generate TVcf MCSPDUs to reject tokens.
1375                ';
1376        TASK    b := 0,
1377                pdu!kind := TVcf,
1378                pdu!result := RT_no_such_user;
1379         1b :   /* for t in tReject */
1380        DECISION tReject = Empty;
1381        (False):CALL    Allocate_buffer(b);
1382                DECISION b = 0;
1383                (True): RETURN;
1384                ELSE:ENDDECISION;
1385                TASK    t := Pick(tReject),
1386                        tReject := Del(t, tReject),
1387                        token(t)!kind := Grabbed,
1388                        u := token(t)!grabber,
1389                        pdu!initiator := u,
1390                        pdu!tokenId := t;
1391                CALL    Token_status(pdu);
1392                TASK    buffer(b)!pdu := pdu;
1393                CALL    Route_user(u, p);
1394                CALL    Cast_buffer(b, p);
1395                JOIN 1b;
1396        ELSE:ENDDECISION;
1397        CALL    Release_buffer(b);
1398        RETURN;
1399        ENDPROCEDURE;

1401                                                        /*---------------*/
1402    PROCEDURE       Process_PDU;                          /* Process_PDU  */
1403    FPAR            r               PortalId,           /*---------------*/
1404                    dp              DataPriority;

1406        DCL     b               BufferId,
1407                pdu             PDUStruct,
1408                diagnostic      Diagnostic;
1409        START
1410        COMMENT'This is the main line of processing an MCSPDU.
1411                If its content is invalid, ignore or reject it.
1412                Else if this is the top provider, take the special
1413                actions of Top_provider.  Then whether or not this
```

```
1414                is the top provider, call Apply_PDU.
1415                ';
1416          TASK    b := portal(r)!inBuffer(dp),
1417                pdu := buffer(b)!pdu;
1418          DECISION pdu!kind = RJum;
1419          (True): TASK    diagnostic := pdu!diagnostic;
1420          (False):CALL    Validate_input(r, b, diagnostic);
1421                DECISION diagnostic = DC_OK;
1422                (True): DECISION buffer(b)!pdu!kind;
1423                      (MCrq): TASK    mcrqQueue := Push(r, mcrqQueue);
1424                      (MTrq): TASK    mtrqQueue := Push(r, mtrqQueue);
1425                      (AUrq): TASK    aurqQueue := Push(r, aurqQueue);
1426                      ELSE:ENDDECISION;
1427                      DECISION upward = 0;
1428                      (True): CALL    Top_provider(r, b);
1429                      ELSE:ENDDECISION;
1430                      CALL    Apply_PDU(r, b, diagnostic);
1431                ELSE:ENDDECISION;
1432          ENDDECISION;
1433          DECISION diagnostic;
1434          (DC_OK, DC_ignore):
1435                /* no action */
1436          ELSE:
1437                OUTPUT  Report.portal(r, diagnostic) TO control;
1438                DECISION pdu!kind = RJum;
1439                (False):TASK'pdu!initialOctets := truncate pdu';
1440                ELSE:ENDDECISION;
1441                TASK    pdu!kind := RJum,
1442                      pdu!diagnostic := diagnostic,
1443                      buffer(b)!pdu := pdu,
1444                      dp := buffer(b)!dataPriority,
1445                      buffer(b)!dataPriority := 0;
1446                CALL    Cast_buffer(b, r);
1447                TASK    buffer(b)!dataPriority := dp;
1448          ENDDECISION;
1449          CALL    Release_buffer(b);
1450          TASK    portal(r)!inBuffer(dp) := 0;
1451          DECISION portal(r)!inCredit(dp) > 0;
1452          (True): TASK    pBufferWait(dp) := Incl(r, pBufferWait(dp));
1453          ELSE:ENDDECISION;
1454          RETURN;
1455          ENDPROCEDURE;

1457                                                          /*------------------*/
1458    PROCEDURE       Validate_input;                          /*  Validate_input  */
1459    FPAR            r             PortalId,                /*------------------*/
1460                    b             BufferId,
1461          IN/OUT  diagnostic      Diagnostic;

1463          DCL     pdu           PDUStruct,
1464                  p             PortalId,
1465                  dp            DataPriority,
1466                  u             UserId,
1467                  uSet          UserIdSet,
1468                  c             ChannelId,
1469                  cSet          ChannelIdSet,
1470                  cAttr         ChannelAttributes,
1471                  cAttrSet      ChannelAttributesSet,
1472                  t             TokenId,
1473                  tSet          TokenIdSet,
1474                  tAttr         TokenAttributes,
1475                  tAttrSet      TokenAttributesSet;
1476          START
1477          COMMENT'Validate the direction and user ids of an MCSPDU
1478                  and perform other checks, depending on its type.
1479                  Top_provider and Apply_PDU trust this procedure
1480                  to catch important anomalies and ease their logic.
1481                  ';
1482          TASK    pdu := buffer(b)!pdu,
1483                  diagnostic := DC_OK;
1484          DECISION portal(r)!kind;
1485          (Downlink):
1486                  DECISION pdu!kind;
1487                  (EDrq, MCrq, MTrq, DPum, RJum,
1488                   AUrq, DUrq, CJrq, CLrq, CCrq,
1489                   CDrq, CArq, CErq, SDrq, USrq,
1490                   TGrq, TIrq, TVrq, TVrs, TPrq,
1491                   TRrq, TTrq):
1492                        /* OK */
```

```
1493                ELSE:
1494                      TASK    diagnostic := DC_forbidden_PDU_upward;
1495                      RETURN;
1496                ENDDECISION;
1497          (Uplink):
1498                DECISION pdu!kind;
1499                (PDin, MCcf, PCin, MTcf, PTin,
1500                 DPum, RJum, AUcf, DUin, CJcf,
1501                 CCcf, CDin, CAin, CEin, SDin,
1502                 USin, TGcf, TIcf, TVin, TVcf,
1503                 TPin, TRcf, TTcf):
1504                      /* OK */
1505                ELSE:
1506                      TASK    diagnostic := DC_forbidden_PDU_downward;
1507                      RETURN;
1508                ENDDECISION;
1509          ELSE:ENDDECISION;
1510          TASK    dp := 0;
1511          DECISION pdu!kind;
1512          (SDrq, SDin, USrq, USin):
1513                TASK    dp := pdu!dataPriority,
1514                      dp := IF dp < parameters!numPriorities THEN dp
1515                                  ELSE parameters!numPriorities - 1 FI;
1516          ELSE:ENDDECISION;
1517          DECISION buffer(b)!dataPriority = dp;
1518          (False):TASK    diagnostic := DC_wrong_transport_priority;
1519                RETURN;
1520          ELSE:ENDDECISION;
1521          DECISION pdu!kind;
1522          (MCrq): TASK    cAttrSet := pdu!mergeChannels,
1523                      cSet := pdu!purgeChannelIds;
1524                 1b :   /* for cAttr in cAttrSet */
1525                DECISION cAttrSet = Empty;
1526                (False):TASK    cAttr := Pick(cAttrSet),
1527                            cAttrSet := Del(cAttr, cAttrSet),
1528                            c := cAttr!channelId;
1529                      DECISION c in cSet;
1530                      (True): TASK    diagnostic := DC_inconsistent_merge;
1531                            RETURN;
1532                      ELSE:ENDDECISION;
1533                      TASK    cSet := Incl(c, cSet);
1534                      DECISION cAttr!kind = Private;
1535                      (False):JOIN 1b;
1536                      ELSE:ENDDECISION;
1537                      TASK    pdu!mergeChannels := Del(cAttr, pdu!mergeChannels);
1538                      CALL    Route_user(cAttr!manager, p);
1539                      DECISION p = r;
1540                      (False):TASK    pdu!purgeChannelIds :=
1541                                                Incl(c, pdu!purgeChannelIds);
1542                            JOIN 1b;
1543                      ELSE:ENDDECISION;
1544                      TASK    uSet := cAttr!admitted;
1545                       2b :   /* for u in uSet */
1546                      DECISION uSet = Empty;
1547                      (False):TASK    u := Pick(uSet),
1548                                  uSet := Del(u, uSet);
1549                            CALL    Route_user(u, p);
1550                            DECISION p = r;
1551                            (False):TASK    cAttr!admitted :=
1552                                                Del(u, cAttr!admitted);
1553                            ELSE:ENDDECISION;
1554                            JOIN 2b;
1555                      ELSE:ENDDECISION;
1556                      TASK    pdu!mergeChannels := Incl(cAttr, pdu!mergeChannels);
1557                      JOIN 1b;
1558                ELSE:ENDDECISION;
1559          (MTrq): TASK    tAttrSet := pdu!mergeTokens,
1560                      tSet := pdu!purgeTokenIds;
1561                 3b :   /* for tAttr in tAttrSet */
1562                DECISION tAttrSet = Empty;
1563                (False):TASK    tAttr := Pick(tAttrSet),
1564                            tAttrSet := Del(tAttr, tAttrSet),
1565                            t := tAttr!tokenId;
1566                      DECISION t in tSet;
1567                      (True): TASK    diagnostic := DC_inconsistent_merge;
1568                            RETURN;
1569                      ELSE:ENDDECISION;
1570                      TASK    tSet := Incl(t, tSet),
1571                            pdu!mergeTokens := Del(tAttr, pdu!mergeTokens);
```

```
1572                    DECISION tAttr!kind;
1573                    (Grabbed, Ungivable):
1574                          CALL    Route_user(tAttr!grabber, p);
1575                          DECISION p = r;
1576                          (False):JOIN 4f;
1577                          ELSE:ENDDECISION;
1578                    (Given):
1579                          CALL    Route_user(tAttr!recipient, p);
1580                          DECISION p = r;
1581                          (False):JOIN 4f;
1582                          ELSE:ENDDECISION;
1583                    (Giving):
1584                          CALL    Route_user(tAttr!recipient, p);
1585                          DECISION p = r;
1586                          (True): CALL    Route_user(tAttr!grabber, p);
1587                                DECISION p = r;
1588                                (False):TASK    tAttr!kind := Given;
1589                                ELSE:ENDDECISION;
1590                          (False):TASK    tAttr!kind := Ungivable;
1591                                CALL    Route_user(tAttr!grabber, p);
1592                                DECISION p = r;
1593                                (False): 4f :
1594                                      TASK    pdu!purgeTokenIds :=
1595                                                Incl(t, pdu!purgeTokenIds);
1596                                      JOIN 3b;
1597                                ELSE:ENDDECISION;
1598                          ENDDECISION;
1599                    (Inhibited):
1600                          TASK    uSet := tAttr!inhibitors;
1601                           5b :   /* for u in uSet */
1602                          DECISION uSet = Empty;
1603                          (False):TASK    u := Pick(uSet),
1604                                      uSet := Del(u, uSet);
1605                                CALL    Route_user(u, p);
1606                                DECISION p = r;
1607                                (False):TASK    tAttr!inhibitors :=
1608                                                Del(u, tAttr!inhibitors);
1609                                ELSE:ENDDECISION;
1610                                JOIN 5b;
1611                          ELSE:ENDDECISION;
1612                    ENDDECISION;
1613                    TASK    pdu!mergeTokens := Incl(tAttr, pdu!mergeTokens);
1614                    JOIN 3b;
1615              ELSE:ENDDECISION;
1616        (DUrq): TASK    uSet := pdu!userIds;
1617                 6b :   /* for u in uSet */
1618                DECISION uSet = Empty;
1619                (False):TASK    u := Pick(uSet),
1620                              uSet := Del(u, uSet);
1621                       CALL    Route_user(u, p);
1622                       DECISION p = r;
1623                       (False):TASK    pdu!userIds := Del(u, pdu!userIds);
1624                       ELSE:ENDDECISION;
1625                       JOIN 6b;
1626                ELSE:ENDDECISION;
1627                DECISION pdu!userIds = Empty;
1628                (True): TASK    diagnostic := DC_ignore;
1629                       RETURN;
1630                ELSE:ENDDECISION;
1631        (CJrq, CCrq, CDrq, CArq, CErq,
1632         SDrq, USrq, TGrq, TIrq, TVrq,
1633         TPrq, TRrq, TTrq):
1634                CALL    Route_user(pdu!initiator, p);
1635                DECISION p = r;
1636                (False):TASK    diagnostic := DC_ignore;
1637                       RETURN;
1638                ELSE:ENDDECISION;
1639        (TVin): DECISION pdu!recipient in uUsed;
1640                (False):TASK    diagnostic := DC_misrouted_user;
1641                       RETURN;
1642                ELSE:ENDDECISION;
1643        (TVrs): CALL    Route_user(pdu!recipient, p);
1644                DECISION p = r;
1645                (False):TASK    diagnostic := DC_ignore;
1646                       RETURN;
1647                ELSE:ENDDECISION;
1648        (CJcf, CCcf, TGcf, TIcf, TVcf,
1649         TRcf, TTcf):
1650                DECISION pdu!initiator in uUsed;
```

```
1651                (False):TASK    diagnostic := DC_misrouted_user;
1652                      RETURN;
1653               ELSE:ENDDECISION;
1654         ELSE:ENDDECISION;
1655         DECISION pdu!kind;
1656         (CDrq, CArq, CErq):
1657               TASK    c := pdu!channelId;
1658               DECISION c in cUsed and chan(c)!kind = Private
1659                     and pdu!initiator = chan(c)!manager;
1660               (False):TASK    diagnostic := DC_ignore;
1661                     RETURN;
1662               ELSE:ENDDECISION;
1663         (SDrq, USrq):
1664               TASK    c := pdu!channelId;
1665               DECISION c in cUsed and chan(c)!kind = Private;
1666               (True): DECISION pdu!initiator in chan(c)!admitted;
1667                     (False):TASK    diagnostic := DC_ignore;
1668                           RETURN;
1669                     ELSE:ENDDECISION;
1670               ELSE:ENDDECISION;
1671         (MCcf): DECISION not merging and mcrqQueue = EmptyQueue;
1672               (True): TASK    diagnostic := DC_unrequested_confirm;
1673                     RETURN;
1674               ELSE:ENDDECISION;
1675         (MTcf): DECISION not merging and mtrqQueue = EmptyQueue;
1676               (True): TASK    diagnostic := DC_unrequested_confirm;
1677                     RETURN;
1678               ELSE:ENDDECISION;
1679         (AUcf): DECISION aurqQueue = EmptyQueue;
1680               (True): TASK    diagnostic := DC_unrequested_confirm;
1681                     RETURN;
1682               ELSE:ENDDECISION;
1683         (CJcf, CCcf, TGcf, TIcf, TVcf,
1684          TRcf, TTcf):
1685               DECISION merging;
1686               (True): TASK    diagnostic := DC_unrequested_confirm;
1687                     RETURN;
1688               ELSE:ENDDECISION;
1689         ELSE:ENDDECISION;
1690         TASK    buffer(b)!pdu := pdu;
1691         RETURN
1692         COMMENT'Additional tests might be applied to check
1693               that MCSPDUs flowing downward are consistent
1694               with user or channel states already recorded.
1695               But stronger assertions could contain flaws,
1696               and such defenses are not necessary for the
1697               continued functioning of this MCS provider.
1698               The value of an MCS domain involves trust in
1699               the correct operation of superior providers.
1700               ';
1701         ENDPROCEDURE;

1703                                                          /*----------------*/
1704   PROCEDURE      Top_provider;                            /*  Top_provider  */
1705   FPAR           r            PortalId,                  /*----------------*/
1706                  b            BufferId;

1708         DCL     pdu           PDUStruct,
1709                 new           PDUStruct,
1710                 u             UserId,
1711                 c             ChannelId,
1712                 cAttr         ChannelAttributes,
1713                 t             TokenId,
1714                 tAttr         TokenAttributes,
1715                 result        Result,
1716                 diagnostic    Diagnostic;
1717         START
1718         COMMENT'The buffer containing an MCSPDU will be processed next
1719               by Apply_PDU.  Its contents may first be modified here.
1720               Changing pdu!kind results in "turning the corner".
1721               ';
1722         TASK    pdu := buffer(b)!pdu;
1723         DECISION pdu!kind;
1724         (EDrq): DECISION pdu!subHeight < parameters!maxHeight;
1725               (False):TASK    pdSend := True;
1726               ELSE:ENDDECISION;
1727         (MCrq): TASK    new!kind := MCcf,
1728                       new!mergeChannels := Empty,
1729                       new!purgeChannelIds := pdu!purgeChannelIds;
```

```
1730                1b :   /* for cAttr in mergeChannels */
1731               DECISION pdu!mergeChannels = Empty;
1732               (False):TASK    cAttr := Pick(pdu!mergeChannels),
1733                             pdu!mergeChannels := Del(cAttr, pdu!mergeChannels),
1734                             c := cAttr!channelId;
1735                      DECISION c in cUsed;
1736                      (True): DECISION chan(c)!kind;
1737                             (Static):
1738                                    JOIN 2f;
1739                             (Private):
1740                                    DECISION cAttr!kind = Private
1741                                           and cAttr!manager = chan(c)!manager;
1742                                    (True): JOIN 2f;
1743                                    ELSE:ENDDECISION;
1744                             ELSE:ENDDECISION;
1745                      (False):DECISION cAttr!kind = UserId;
1746                             (True): CALL    Take_user(c, diagnostic);
1747                             (False):CALL    Take_channel(c, diagnostic);
1748                             ENDDECISION;
1749                             DECISION diagnostic = DC_OK;
1750                             (True): TASK    chan(c)!kind := cAttr!kind;
1751                                     2f :
1752                                    TASK    new!mergeChannels :=
1753                                                  Incl(cAttr, new!mergeChannels);
1754                                    JOIN 1b;
1755                             ELSE:ENDDECISION;
1756                      ENDDECISION;
1757                      TASK    new!purgeChannelIds := Incl(c, new!purgeChannelIds);
1758                      JOIN 1b;
1759               ELSE:ENDDECISION;
1760               TASK    pdu := new;
1761        (MTrq): TASK    new!kind := MTcf,
1762                      new!mergeTokens := Empty,
1763                      new!purgeTokenIds := pdu!purgeTokenIds;
1764                3b :   /* for tAttr in mergeTokens */
1765               DECISION pdu!mergeTokens = Empty;
1766               (False):TASK    tAttr := Pick(pdu!mergeTokens),
1767                             pdu!mergeTokens := Del(tAttr, pdu!mergeTokens),
1768                             t := tAttr!tokenId;
1769                      DECISION t in tUsed;
1770                      (True): DECISION token(t)!kind;
1771                             (Inhibited):
1772                                    DECISION tAttr!kind = Inhibited;
1773                                    (True): JOIN 4f;
1774                                    ELSE:ENDDECISION;
1775                             ELSE:ENDDECISION;
1776                      (False):CALL    Take_token(t, diagnostic);
1777                             DECISION diagnostic = DC_OK;
1778                             (True):  4f :
1779                                    TASK    new!mergeTokens :=
1780                                                  Incl(tAttr, new!mergeTokens);
1781                                    JOIN 3b;
1782                             ELSE:ENDDECISION;
1783                      ENDDECISION;
1784                      TASK    new!purgeTokenIds := Incl(t, new!purgeTokenIds);
1785                      JOIN 3b;
1786               ELSE:ENDDECISION;
1787               TASK    pdu := new;
1788        (AUrq): CALL    New_user(u, result);
1789               TASK    pdu!kind := AUcf,
1790                      pdu!result := result,
1791                      pdu!initiator := u;
1792        (DUrq): TASK    pdu!kind := DUin;
1793        (CJrq): TASK    pdu!kind := CJcf,
1794                      c := pdu!channelId,
1795                      pdu!requested := c,
1796                      result := RT_successful;
1797               DECISION c = 0;
1798               (True): CALL    New_channel(c, result);
1799                      DECISION result = RT_successful;
1800                      (True): TASK    chan(c)!kind := Assigned;
1801                      ELSE:ENDDECISION;
1802                      TASK    pdu!channelId := c;
1803               (False):DECISION c in cUsed;
1804                      (False):DECISION c < 1001;
1805                             (False):TASK    result := RT_no_such_channel;
1806                             (True): CALL    Take_chanel(c, diagnostic);
1807                                    DECISION diagnostic = DC_OK;
1808                                    (False):TASK    result := RT_too_many_channels;
```

```
1809                                    (True): TASK    chan(c)!kind := Static;
1810                                    ENDDECISION;
1811                              ENDDECISION;
1812                        (True): DECISION chan(c)!kind;
1813                              (UserId):
1814                                    TASK    u := UserId(c);
1815                                    DECISION pdu!initiator = u;
1816                                    (False):TASK    result := RT_other_user_id;
1817                                    ELSE:ENDDECISION;
1818                              (Private):
1819                                    DECISION pdu!initiator in chan(c)!admitted;
1820                                    (False):TASK    result := RT_not_admitted;
1821                                    ELSE:ENDDECISION;
1822                              ELSE:ENDDECISION;
1823                        ENDDECISION;
1824                        DECISION result = RT_successful;
1825                        (False):TASK    pdu!channelId := 0;
1826                        ELSE:ENDDECISION;
1827                  ENDDECISION;
1828                  TASK    pdu!result := result;
1829            (CLrq): /* no special action */
1830            (CCrq): CALL    New_channel(c, result);
1831                  TASK    pdu!kind := CCcf,
1832                        pdu!result := result,
1833                        pdu!channelId := c;
1834            (CDrq): TASK    pdu!kind := CDin;
1835            (CArq): TASK    pdu!kind := CAin;
1836            (CErq): TASK    pdu!kind := CEin;
1837            (SDrq): TASK    pdu!kind := SDin;
1838            (USrq): TASK    pdu!kind := USin;
1839            (TGrq): TASK    pdu!kind := TGcf,
1840                        pdu!result := RT_successful,
1841                        t := pdu!tokenId,
1842                        u := pdu!initiator;
1843                  DECISION t in tUsed;
1844                  (False):CALL    Take_token(t, diagnostic);
1845                        DECISION diagnostic = DC_OK;
1846                        (False):TASK    pdu!result := RT_too_many_tokens;
1847                        (True): TASK    token(t)!kind := Grabbed,
1848                                    token(t)!grabber := u;
1849                        ENDDECISION;
1850                  (True): DECISION token(t)!kind = Inhibited
1851                              and token(t)!inhibitors = Incl(u, Empty);
1852                        (False):TASK    pdu!result := RT_token_not_available;
1853                        (True): TASK    token(t)!kind := Grabbed,
1854                                    token(t)!grabber := u;
1855                        ENDDECISION;
1856                  ENDDECISION;
1857                  CALL    Token_status(pdu);
1858            (TIrq): TASK    pdu!kind := TIcf,
1859                        pdu!result := RT_successful,
1860                        t := pdu!tokenId,
1861                        u := pdu!initiator;
1862                  DECISION t in tUsed;
1863                  (False):CALL    Take_token(t, diagnostic);
1864                        DECISION diagnostic = DC_OK;
1865                        (False):TASK    pdu!result := RT_too_many_tokens;
1866                        (True): TASK    token(t)!kind := Inhibited,
1867                                    token(t)!inhibitors := Incl(u, Empty);
1868                        ENDDECISION;
1869                  (True): DECISION token(t)!kind = Grabbed and token(t)!grabber = u;
1870                        (True): TASK    token(t)!kind := Inhibited,
1871                                    token(t)!inhibitors := Incl(u, Empty);
1872                        ELSE:ENDDECISION;
1873                        DECISION token(t)!kind = Inhibited;
1874                        (False):TASK    pdu!result := RT_token_not_available;
1875                        (True): TASK    token(t)!inhibitors :=
1876                                          Incl(u, token(t)!inhibitors);
1877                        ENDDECISION;
1878                  ENDDECISION;
1879                  CALL    Token_status(pdu);
1880            (TVrq): TASK    pdu!kind := TVcf,
1881                        pdu!result := RT_token_not_possessed,
1882                        t := pdu!tokenId,
1883                        u := pdu!initiator;
1884                  DECISION t in tUsed and token(t)!kind = Grabbed and token(t)!grabber =
u;
1885                  (True): DECISION pdu!recipient in uUsed;
1886                        (False):TASK    pdu!result := RT_no_such_user;
```

```
                        (True): TASK    pdu!kind := TVin,
                                        token(t)!kind := Giving,
                                        token(t)!recipient := pdu!recipient;
                        ENDDECISION;
                ELSE:ENDDECISION;
                CALL    Token_status(pdu);
        (TPrq): TASK    pdu!kind := TPin;
        (TRrq): TASK    pdu!kind := TRcf,
                        pdu!result := RT_token_not_possessed,
                        t := pdu!tokenId,
                        u := pdu!initiator;
                DECISION t in tUsed;
                (True): DECISION token(t)!kind;
                        (Grabbed, Ungivable):
                                DECISION token(t)!grabber = u;
                                (True): TASK    pdu!result := RT_successful;
                                        CALL    Delete_token(t);
                                ELSE:ENDDECISION;
                        (Giving):
                                DECISION token(t)!grabber = u;
                                (True): TASK    pdu!result := RT_successful,
                                                token(t)!kind := Given;
                                ELSE:ENDDECISION;
                        (Inhibited):
                                DECISION u in token(t)!inhibitors;
                                (True): TASK    pdu!result := RT_successful,
                                                token(t)!inhibitors :=
                                                        Del(u, token(t)!inhibitors);
                                ELSE:ENDDECISION;
                                DECISION token(t)!inhibitors = Empty;
                                (True): CALL    Delete_token(t);
                                ELSE:ENDDECISION;
                        ENDDECISION;
                ELSE:ENDDECISION;
                CALL    Token_status(pdu);
        (TTrq): TASK    pdu!kind := TTcf;
                CALL    Token_status(pdu);
        (TVrs): TASK    t := pdu!tokenId,
                        u := pdu!recipient;
                DECISION t in tUsed and token(t)!recipient = u;
                (True): DECISION token(t)!kind;
                        (Giving):
                                TASK    token(t)!kind := Grabbed,
                                        pdu!kind := TVcf,
                                        pdu!initiator := token(t)!grabber;
                                DECISION pdu!result = RT_successful;
                                (True): TASK    token(t)!grabber := u;
                                ELSE:ENDDECISION;
                                CALL    Token_status(pdu);
                        ELSE:ENDDECISION;
                ELSE:ENDDECISION;
        ELSE:ENDDECISION;
        TASK    buffer(b)!pdu := pdu;
        RETURN;
        ENDPROCEDURE;

                                                        /*-------------*/
PROCEDURE       Apply_PDU;                                /*  Apply_PDU  */
FPAR            r               PortalId,                 /*-------------*/
                b               BufferId,
        IN/OUT  diagnostic      Diagnostic;

        DCL     pdu             PDUStruct,
                new             PDUStruct,
                p               PortalId,
                pSet            PortalIdSet,
                x               PortalId,
                u               UserId,
                uSet            UserIdSet,
                c               ChannelId,
                cSet            ChannelIdSet,
                cAttr           ChannelAttributes,
                cAttrSet        ChannelAttributesSet,
                t               TokenId,
                tSet            TokenIdSet,
                tAttr           TokenAttributes,
                tAttrSet        TokenAttributesSet,
                result          Result;
        START
```

```
COMMENT'This is a large case selection based on MCSPDU kind.
       These actions and updates to the information base
       take place in both the top and subordinate providers.
       ';
TASK    pdu := buffer(b)!pdu;
DECISION pdu!kind;
(MCrq, MTrq, AUrq, DUrq, CCrq,
 CDrq, CArq, CErq, USrq, TGrq,
 TIrq, TVrq, TPrq, TRrq, TTrq):
       CALL    Cast_buffer(b, upward);
(PDin): DECISION pdu!heightLimit > 0;
       (True):TASK     buffer(b)!pdu!heightLimit := pdu!heightLimit - 1;
       (False):OUTPUT  Report.portal(r, DC_height_limit_exceeded) TO control;
       ENDDECISION;
       CALL    Broadcast_buffer(b);
(EDrq): TASK    portal(r)!subHeight := pdu!subHeight,
               portal(r)!subInterval := pdu!subInterval,
               portal(r)!interval := IF pdu!subInterval = 0 THEN interval
                                     ELSE oneSecond + (pdu!subInterval * 3) FI;
       CALL    Erect_domain;
(MCcf): DECISION merging;
       (False):TASK    p := Next(mcrqQueue),
                       mcrqQueue := Pull(mcrqQueue);
       ELSE:ENDDECISION;
       TASK    cAttrSet := pdu!mergeChannels;
        1b :   /* for cAttr in cAttrSet */
       DECISION cAttrSet = Empty;
       (False):TASK    cAttr := Pick(cAttrSet),
                       cAttrSet := Del(cAttr, cAttrSet),
                       c := cAttr!channelId;
              DECISION c in cUsed and chan(c)!kind = cAttr!kind;
              (False):DECISION cAttr!kind = UserId;
                     (True): CALL    Take_user(c, diagnostic);
                     (False):CALL    Take_channel(c, diagnostic);
                     ENDDECISION;
                     DECISION diagnostic = DC_OK;
                     (False):RETURN;
                     ELSE:ENDDECISION;
                     TASK    chan(c)!kind := cAttr!kind;
              ELSE:ENDDECISION;
              DECISION merging;
              (True): DECISION chan(c)!kind = UserId;
                     (True): TASK    u := UserId(c),
                                     uConfirm := Incl(u, uConfirm);
                     ELSE:ENDDECISION;
                     DECISION chan(c)!kind = Private
                            and chan(c)!uMerge /= Empty;
                     (True): TASK    cMerge := Incl(c, cMerge);
                     (False):TASK    cConfirm := Incl(c, cConfirm);
                     ENDDECISION;
                     JOIN 1b;
              ELSE:ENDDECISION;
              TASK    chan(c)!portal := p;
              DECISION cAttr!kind = Static or cAttr!kind = Assigned
                     or cAttr!joined;
              (True): DECISION p = 0;
                     (False):TASK    chan(c)!joined :=
                                            Incl(p, chan(c)!joined),
                                     cLeave := Del(c, cLeave);
                     (True): DECISION chan(c)!joined = Empty;
                            (True): TASK    cLeave := Incl(c, cLeave);
                            ELSE:ENDDECISION;
                     ENDDECISION;
              ELSE:ENDDECISION;
              DECISION cAttr!kind = UserId and p = 0;
              (True): TASK    u := UserId(c),
                              uDetach := Incl(u, uDetach),
                              cLeave := Del(c, cLeave);
              ELSE:ENDDECISION;
              DECISION cAttr!kind = Private;
              (False):JOIN 1b;
              ELSE:ENDDECISION;
              CALL    Route_user(cAttr!manager, x);
              DECISION x = p;
              (False):TASK    chan(c)!manager := 0,
                              buffer(b)!pdu!mergeChannels :=
                                     Del(cAttr, buffer(b)!pdu!mergeChannels),
                              buffer(b)!pdu!purgeChannelIds :=
                                     Incl(c, buffer(b)!pdu!purgeChannelIds),
```

```
                                    cDisband := Incl(c, cDisband);
                        (True): TASK   chan(c)!manager := cAttr!manager,
                                    uSet := cAttr!admitted and uUsed;
                                 2b :   /* for u in uSet */
                                DECISION uSet = Empty;
                                (False):TASK    u := Pick(uSet),
                                            uSet := Del(u, uSet);
                                    CALL    Route_user(u, x);
                                    DECISION x = p;
                                    (True): TASK    chan(c)!admitted :=
                                                        Incl(u, chan(c)!admitted);
                                    ELSE:ENDDECISION;
                                    JOIN 2b;
                                ELSE:ENDDECISION;
                            ENDDECISION;
                            JOIN 1b;
                        ELSE:ENDDECISION;
                        DECISION merging;
                        (False):CALL    Cast_buffer(b, p);
                        (True): TASK    new!kind := PCin,
                                    new!detachUserIds := Empty,
                                    new!purgeChannelIds := Empty,
                                    cSet := pdu!purgeChannelIds and cUsed;
                                 3b :   /* for c in cSet */
                                DECISION cSet = Empty;
                                (False):TASK    c := Pick(cSet),
                                            cSet := Del(c, cSet);
                                    DECISION chan(c)!kind;
                                    (UserId):
                                            TASK    u := UserId(c),
                                                    new!detachUserIds :=
                                                        Incl(u, new!detachUserIds);
                                    (Static, Assigned, Private):
                                            TASK    new!purgeChannelIds :=
                                                        Incl(c, new!purgeChannelIds);
                                    ENDDECISION;
                                    JOIN 3b;
                            ELSE:ENDDECISION;
                            CALL    Purge_users(new!detachUserIds);
                            CALL    Purge_channels(new!purgeChannelIds);
                            TASK    buffer(b)!pdu := new;
                            CALL    Broadcast_buffer(b);
                        ENDDECISION;
(PCin): TASK    cSet := pdu!purgeChannelIds and cUsed,
                    buffer(b)!pdu!purgeChannelIds := cSet;
                DECISION merging;
                (True): TASK    buffer(b)!pdu!detachUserIds := pdu!detachUserIds
                                                            and uConfirm,
                            buffer(b)!pdu!purgeChannelIds := cSet and cConfirm;
                         4b :   /* for c in cSet */
                        DECISION cSet = Empty;
                        (False):TASK    c := Pick(cSet),
                                    cSet := Del(c, cSet);
                            DECISION c in cConfirm;
                            (False):TASK    chan(c)!uMerge := chan(c)!admitted;
                            ELSE:ENDDECISION;
                            JOIN 4b;
                        ELSE:ENDDECISION;
                ELSE:ENDDECISION;
                CALL    Purge_users(buffer(b)!pdu!detachUserIds);
                CALL    Purge_channels(buffer(b)!pdu!purgeChannelIds);
                CALL    Broadcast_buffer(b);
(MTcf): DECISION merging;
                (False):TASK    p := Next(mtrqQueue),
                            mtrqQueue := Pull(mtrqQueue);
                ELSE:ENDDECISION;
                TASK    tAttrSet := pdu!mergeTokens;
                 5b :   /* for tAttr in tAttrSet */
                DECISION tAttrSet = Empty;
                (False):TASK    tAttr := Pick(tAttrSet),
                            tAttrSet := Del(tAttr, tAttrSet),
                            t := tAttr!tokenId;
                        DECISION t in tUsed;
                        (False):CALL    Take_token(t, diagnostic);
                            DECISION diagnostic = DC_OK;
                            (False):RETURN;
                            ELSE:ENDDECISION;
                        ELSE:ENDDECISION;
                        TASK    token(t)!kind := tAttr!kind,
```

```
2124                            token(t)!grabber := tAttr!grabber,
2125                            token(t)!recipient := tAttr!recipient;
2126                      DECISION merging;
2127                      (True): DECISION token(t)!kind = Inhibited
2128                                  and token(t)!uMerge /= Empty;
2129                              (True): TASK    tMerge := Incl(t, tMerge);
2130                              (False):TASK    tConfirm := Incl(t, tConfirm);
2131                                      DECISION token(t)!kind = Inhibited
2132                                                  and token(t)!inhibitors = Empty;
2133                                      (True): CALL    Delete_token(t);
2134                                      ELSE:ENDDECISION;
2135                              ENDDECISION;
2136                              JOIN 5b;
2137                      ELSE:ENDDECISION;
2138                      DECISION upward = 0 and token(t)!kind = Ungivable;
2139                      (True): TASK    tReject := Incl(t, tReject);
2140                      ELSE:ENDDECISION;
2141                      DECISION tAttr!kind;
2142                      (Grabbed, Ungivable):
2143                              CALL    Token_route(tAttr!grabber, x, p);
2144                              DECISION x = p;
2145                              (False):JOIN 6f;
2146                              ELSE:ENDDECISION;
2147                      (Given):
2148                              CALL    Token_route(tAttr!recipient, x, p);
2149                              DECISION x = p;
2150                              (False):JOIN 6f;
2151                              ELSE:ENDDECISION;
2152                      (Giving):
2153                              CALL    Token_route(tAttr!recipient, x, p);
2154                              DECISION x = p;
2155                              (True): CALL    Token_route(tAttr!grabber, x, p);
2156                                      DECISION x = p;
2157                                      (False):TASK    token(t)!kind := Given;
2158                                      ELSE:ENDDECISION;
2159                              (False):CALL    Token_route(tAttr!grabber, x, p);
2160                                      DECISION x = p;
2161                                      (True): TASK    token(t)!kind := Ungivable;
2162                                      (False): 6f :
2163                                              TASK    buffer(b)!pdu!mergeTokens :=
2164                                                 Del(tAttr, buffer(b)!pdu!mergeTokens),
2165                                                      buffer(b)!pdu!purgeTokenIds :=
2166                                                 Incl(t, buffer(b)!pdu!purgeTokenIds);
2167                                              CALL    Delete_token(t);
2168                                      ENDDECISION;
2169                              ENDDECISION;
2170                      (Inhibited):
2171                              TASK    uSet := tAttr!inhibitors and uUsed;
2172                               7b :   /* for u in uSet */
2173                              DECISION uSet = Empty;
2174                              (False):TASK    u := Pick(uSet),
2175                                              uSet := Del(u, uSet);
2176                                      CALL    Token_route(u, x, p);
2177                                      DECISION x = p;
2178                                      (True): TASK    token(t)!inhibitors :=
2179                                                       Incl(u, token(t)!inhibitors);
2180                                      ELSE:ENDDECISION;
2181                                      JOIN 7b;
2182                              ELSE:ENDDECISION;
2183                              DECISION token(t)!inhibitors = Empty;
2184                              (True): CALL    Delete_token(t);
2185                              ELSE:ENDDECISION;
2186                      ENDDECISION;
2187                      JOIN 5b;
2188              ELSE:ENDDECISION;
2189              DECISION merging;
2190              (False):CALL    Cast_buffer(b, p);
2191              (True): CALL    Purge_tokens(pdu!purgeTokenIds);
2192                      TASK    buffer(b)!pdu!kind := PTin;
2193                      CALL    Broadcast_buffer(b);
2194              ENDDECISION;
2195      (PTin): TASK    tSet := pdu!purgeTokenIds and tUsed,
2196                      buffer(b)!pdu!purgeTokenIds := tSet;
2197              DECISION merging;
2198              (True): TASK    buffer(b)!pdu!purgeTokenIds := tSet and tConfirm;
2199                       8b :   /* for t in tSet */
2200                      DECISION tSet = Empty;
2201                      (False):TASK    t := Pick(tSet),
2202                                      tSet := Del(t, tSet);
```

```
                        DECISION t in tConfirm;
                        (False):TASK    token(t)!uMerge := token(t)!inhibitors;
                        ELSE:ENDDECISION;
                        JOIN 8b;
                  ELSE:ENDDECISION;
            ELSE:ENDDECISION;
            CALL    Purge_tokens(buffer(b)!pdu!purgeTokenIds);
            CALL    Broadcast_buffer(b);
      (DPum): OUTPUT  Drop.portal(r, pdu!reason) TO control;
      (AUcf): TASK    p := Next(aurqQueue),
                  aurqQueue := Pull(aurqQueue),
                  c := ChannelId(u),
                  u := pdu!initiator;
            DECISION pdu!result = RT_successful;
            (True): DECISION upward = 0;
                  (False):CALL    Take_user(c, diagnostic);
                        DECISION diagnostic = DC_OK;
                        (False):RETURN;
                        ELSE:ENDDECISION;
                  ELSE:ENDDECISION;
                  TASK    chan(c)!portal := p;
                  DECISION p = 0;
                  (True): TASK    uDetach := Incl(u, uDetach),
                              cLeave := Del(c, cLeave);
                  ELSE:ENDDECISION;
            ELSE:ENDDECISION;
            CALL    Cast_buffer(b, p);
      (DUin): DECISION merging;
            (True): TASK    buffer(b)!pdu!userIds := pdu!userIds and uConfirm;
            ELSE:ENDDECISION;
            CALL    Purge_users(buffer(b)!pdu!userIds);
            CALL    Broadcast_buffer(b);
      (CJrq): TASK    c := pdu!channelId,
                  result := RT_successful,
                  p := upward;
            DECISION c in cUsed;
            (True): DECISION chan(c)!kind;
                  (UserId):
                        TASK    u := UserId(c);
                        DECISION pdu!initiator = u;
                        (False):TASK    result := RT_other_user_id;
                        ELSE:ENDDECISION;
                  (Private):
                        DECISION pdu!initiator in chan(c)!admitted;
                        (False):TASK    result := RT_not_admitted;
                        ELSE:ENDDECISION;
                  ELSE:ENDDECISION;
                  DECISION result = RT_successful;
                  (False):TASK    buffer(b)!pdu!kind := CJcf,
                              buffer(b)!pdu!requested := c,
                              buffer(b)!pdu!result := result,
                              buffer(b)!pdu!channelId := 0,
                              p := r;
                  (True): DECISION chan(c)!joined /= Empty or c in cLeave;
                        (True): TASK    chan(c)!joined :=
                                          Incl(r, chan(c)!joined),
                                    cLeave := Del(c, cLeave),
                                    buffer(b)!pdu!kind := CJcf,
                                    buffer(b)!pdu!requested := c,
                                    buffer(b)!pdu!result := result,
                                    p := r;
                        ELSE:ENDDECISION;
                  ENDDECISION;
            ELSE:ENDDECISION;
            CALL    Cast_buffer(b, p);
      (CJcf): TASK    c := pdu!channelId,
                  u := pdu!initiator;
            CALL    Route_user(u, p);
            DECISION pdu!result = RT_successful;
            (True): DECISION c in cUsed;
                  (False):CALL    Take_channel(c, diagnostic);
                        DECISION diagnostic = DC_OK;
                        (False):RETURN;
                        ELSE:ENDDECISION;
                        TASK    chan(c)!kind := IF c < 1001 THEN Static
                                                ELSE Assigned FI;
                  ELSE:ENDDECISION;
                  DECISION p = 0;
                  (False):TASK    chan(c)!joined := Incl(p, chan(c)!joined),
```

```
                                cLeave := Del(c, cLeave);
                        (True): DECISION chan(c)!joined = Empty;
                                (True): TASK    cLeave := Incl(c, cLeave);
                                ELSE:ENDDECISION;
                        ENDDECISION;
                ELSE:ENDDECISION;
                CALL    Cast_buffer(b, p);
        (CLrq): TASK    cSet := pdu!channelIds and cUsed;
                9b :    /* for c in cSet */
                DECISION cSet = Empty;
                (False):TASK    c := Pick(cSet),
                                cSet := Del(c, cSet);
                        DECISION r in chan(c)!joined;
                        (True): TASK    chan(c)!joined := Del(r, chan(c)!joined);
                                DECISION chan(c)!joined = Empty;
                                (True): TASK    cLeave := Incl(c, cLeave);
                                ELSE:ENDDECISION;
                        ELSE:ENDDECISION;
                        JOIN 9b;
                ELSE:ENDDECISION;
        (CCcf): TASK    c := pdu!channelId,
                        u := pdu!initiator;
                DECISION pdu!result = RT_successful;
                (True): DECISION upward = 0;
                        (False):CALL    Take_channel(c, diagnostic);
                                DECISION diagnostic = DC_OK;
                                (False):RETURN;
                                ELSE:ENDDECISION;
                        ELSE:ENDDECISION;
                        TASK    chan(c)!kind := Private,
                                chan(c)!manager := u,
                                chan(c)!admitted := Incl(u, Empty);
                ELSE:ENDDECISION;
                CALL    Route_user(u, p);
                CALL    Cast_buffer(b, p);
        (CDin): TASK    c := pdu!channelId;
                DECISION c in cUsed;
                (False):RETURN;
                (True): DECISION merging and not c in cConfirm;
                        (True): TASK    chan(c)!uMerge := chan(c)!admitted;
                                RETURN;
                        ELSE:ENDDECISION;
                        DECISION chan(c)!kind = Private;
                        (False):TASK    diagnostic := DC_channel_id_conflict;
                                RETURN;
                        ELSE:ENDDECISION;
                ENDDECISION;
                TASK    uSet := Incl(chan(c)!manager, chan(c)!admitted);
                CALL    Delete_channel(c);
                CALL    Multicast_buffer(b, uSet);
        (CAin): TASK    c := pdu!channelId,
                        uSet := pdu!userIds and uUsed,
                        buffer(b)!pdu!userIds := uSet;
                DECISION merging;
                (True): TASK    uSet := uSet and uConfirm,
                                buffer(b)!pdu!userIds := uSet;
                        10b :   /* for u in uSet */
                        DECISION uSet = Empty;
                        (False):TASK    u := Pick(uSet),
                                        uSet := Del(u, uSet),
                                        c := ChannelId(u),
                                        chan(c)!portal = 0;
                                JOIN 10b;
                        ELSE:ENDDECISION;
                        TASK    buffer(b)!pdu!kind := DUrq,
                                buffer(b)!pdu!reason := RN_channel_purged;
                        CALL    Cast_buffer(b, r);
                        RETURN;
                ELSE:ENDDECISION;
                DECISION uSet = Empty;
                (False):DECISION c in cUsed and chan(c)!kind = Private;
                        (False):CALL    Take_channel(c, diagnostic);
                                DECISION diagnostic = DC_OK;
                                (False):RETURN;
                                ELSE:ENDDECISION;
                        ELSE:ENDDECISION;
                        TASK    chan(c)!kind := Private,
                                chan(c)!manager := pdu!initiator,
                                chan(c)!admitted := chan(c)!admitted or uSet;
```

```
                    CALL    Multicast_buffer(b, uSet);
              ELSE:ENDDECISION;
       (CEin): TASK    c := pdu!channelId;
              DECISION c in cUsed;
              (False):RETURN;
              (True): DECISION merging and not c in cConfirm;
                     (True): TASK    chan(c)!uMerge := chan(c)!admitted;
                            RETURN;
                     ELSE:ENDDECISION;
                     DECISION chan(c)!kind = Private;
                     (False):TASK    diagnostic := DC_channel_id_conflict;
                            RETURN;
                     ELSE:ENDDECISION;
              ENDDECISION;
              TASK    uSet := chan(c)!admitted,
                     pSet := Empty;
               11b :  /* for u in uSet */
              DECISION uSet = Empty;
              (False):TASK    u := Pick(uSet),
                            uSet := Del(u, uSet);
                     DECISION u in pdu!userIds;
                     (False):CALL    Route_user(u, p);
                            TASK    pSet := Incl(p, pSet);
                     ELSE:ENDDECISION;
                     JOIN 11b;
              ELSE:ENDDECISION;
              TASK    uSet := pdu!userIds and chan(c)!admitted,
                     buffer(b)!pdu!userIds := uSet;
               12b :  /* for u in uSet */
              DECISION uSet = Empty;
              (False):TASK    u := Pick(uSet),
                            uSet := Del(u, uSet),
                            chan(c)!admitted := Del(u, chan(c)!admitted);
                     CALL    Route_user(u, p);
                     DECISION p in chan(c)!joined and not p in pSet;
                     (False):TASK    chan(c)!joined := Del(p, chan(c)!joined);
                            DECISION chan(c)!joined = Empty;
                            (True): TASK    cLeave := Incl(c, cLeave);
                            ELSE:ENDDECISION;
                     ELSE:ENDDECISION;
                     JOIN 12b;
              ELSE:ENDDECISION;
              DECISION chan(c)!admitted /= Empty or chan(c)!manager in uUsed;
              (False):CALL    Delete_channel(c);
              ELSE:ENDDECISION;
              TASK    uSet := buffer(b)!pdu!userIds;
              CALL    Multicast_buffer(b, uSet);
       (SDrq): TASK    buffer(b)!pdu!kind := SDin,
                     pSet := Incl(upward, Empty);
              JOIN 13f;
       (SDin): TASK    pSet := Empty;
              JOIN 13f;
       (USin): TASK    pSet := Empty;
               13f :
              TASK    c := pdu!channelId;
              DECISION c in cUsed and not merging;
              (True): TASK    pSet := pSet or chan(c)!joined;
                     DECISION chan(c)!kind = UserId;
                     (True): TASK    pSet := Del(upward, pSet);
                     ELSE:ENDDECISION;
              ELSE:ENDDECISION;
              DECISION buffer(b)!pdu!kind = SDin;
              (True): TASK    pSet := Del(r, pSet);
              ELSE:ENDDECISION;
               14b :  /* for p in pSet */
              DECISION pSet = Empty;
              (False):TASK    p := Pick(pSet),
                            pSet := Del(p, pSet);
                     CALL    Cast_buffer(b, p);
                     JOIN 14b;
              ELSE:ENDDECISION;
       (TGcf, TIcf, TVcf, TRcf, TTcf):
              TASK    t := pdu!tokenId,
                     u := pdu!initiator;
              DECISION pdu!tokenStatus;
              (SelfGrabbed):
                     DECISION t in tUsed;
                     (False):CALL    Take_token(t, diagnostic);
                            DECISION diagnostic = DC_OK;
```

```
                        (False):RETURN;
                        ELSE:ENDDECISION;
                  ELSE:ENDDECISION;
                  TASK    token(t)!kind := Grabbed,
                        token(t)!grabber := u;
            (SelfInhibited):
                  DECISION t in tUsed;
                  (False):CALL    Take_token(t, diagnostic);
                        DECISION diagnostic = DC_OK;
                        (False):RETURN;
                        ELSE:ENDDECISION;
                  ELSE:ENDDECISION;
                  TASK    token(t)!kind := Inhibited,
                        token(t)!inhibitors := Incl(u, token(t)!inhibitors);
            (SelfRecipient):
                  TASK    token(t)!recipient := u;
            (SelfGiving):
                  TASK    token(t)!grabber := u;
            (NotInUse):
                  CALL    Delete_token(t);
            ELSE:
                  DECISION token(t)!kind;
                  (Grabbed, Ungivable):
                        DECISION token(t)!grabber = u;
                        (True): CALL    Delete_token(t);
                        ELSE:ENDDECISION;
                  (Giving, Given):
                        DECISION token(t)!grabber = u;
                        (True): TASK    token(t)!kind := Given;
                        ELSE:ENDDECISION;
                        DECISION token(t)!recipient = u;
                        (True): CALL    Delete_token(t);
                        ELSE:ENDDECISION;
                  (Inhibited):
                        TASK    token(t)!inhibitors :=
                                    Del(u, token(t)!inhibitors);
                        DECISION token(t)!inhibitors = Empty;
                        (True): CALL    Delete_token(t);
                        ELSE:ENDDECISION;
                  ENDDECISION;
            ENDDECISION;
            CALL    Route_user(u, p);
            CALL    Cast_buffer(b, p);
      (TVin): TASK    t := pdu!tokenId,
                  u := pdu!recipient;
            DECISION merging;
            (True): TASK    buffer(b)!pdu!kind := TVrs,
                        buffer(b)!pdu!result := RT_domain_merging;
                  CALL    Cast_buffer(b, r);
                  RETURN;
            ELSE:ENDDECISION;
            DECISION t in tUsed;
            (False):CALL    Take_token(t, diagnostic);
                  DECISION diagnostic = DC_OK;
                  (False):RETURN;
                  ELSE:ENDDECISION;
            ELSE:ENDDECISION;
            TASK    token(t)!kind := Giving,
                  token(t)!grabber := pdu!initiator,
                  token(t)!recipient := u;
            CALL    Route_user(u, p);
            CALL    Cast_buffer(b, p);
      (TVrs): TASK    t := pdu!tokenId,
                  u := pdu!recipient;
            DECISION t in tUsed and token(t)!recipient = u;
            (True): DECISION token(t)!kind;
                  (Giving):
                        TASK    token(t)!kind := Grabbed;
                        DECISION pdu!result = RT_successful;
                        (True): TASK    token(t)!grabber := u;
                        (False):DECISION token(t)!grabber in uUsed;
                              (False):CALL    Delete_token(t);
                              ELSE:ENDDECISION;
                        ENDDECISION;
                        CALL    Cast_buffer(b, upward);
                  (Given):
                        TASK    token(t)!kind := Grabbed,
                              token(t)!grabber := u;
                        DECISION pdu!result = RT_successful;
```

```
2519                                (False):CALL    Delete_token(t);
2520                                ELSE:ENDDECISION;
2521                                CALL    Cast_buffer(b, upward);
2522                          ELSE:ENDDECISION;
2523                    ELSE:ENDDECISION;
2524              (TPin): TASK    t := pdu!tokenId;
2525                    DECISION t in tUsed and not merging;
2526                    (True): DECISION token(t)!kind;
2527                          (Grabbed, Ungivable):
2528                                CALL    Route_user(token(t)!grabber, p);
2529                                CALL    Cast_buffer(b, p);
2530                          (Given):
2531                                CALL    Route_user(token(t)!recipient, p);
2532                                CALL    Cast_buffer(b, p);
2533                          (Giving):
2534                                TASK    uSet := Incl(token(t)!grabber, Empty);
2535                                TASK    uSet := Incl(token(t)!recipient, uSet);
2536                                CALL    Multicast_buffer(b, uSet);
2537                          (Inhibited):
2538                                TASK    uSet := token(t)!inhibitors;
2539                                CALL    Multicast_buffer(b, uSet);
2540                          ENDDECISION;
2541                    ELSE:ENDDECISION;
2542              ENDDECISION;
2543              RETURN;
2544              ENDPROCEDURE;

2546                                                              /*----------------*/
2547    PROCEDURE       Token_status;                               /*  Token_status  */
2548    FPAR    IN/OUT  pdu             PDUStruct;                  /*----------------*/

2550              DCL     t               TokenId,
2551                      u               UserId,
2552                      status          TokenStatus;
2553              START
2554              COMMENT'Calculate for an MCSPDU the relationship between
2555                      the initiator user id and token id it contains.
2556                      ';
2557              TASK    t := pdu!tokenId,
2558                      u := pdu!initiator;
2559              DECISION t in tUsed;
2560              (False):TASK    status := NotInUse;
2561              (True): DECISION token(t)!kind;
2562                      (Grabbed):
2563                              DECISION token(t)!grabber = u;
2564                              (True): TASK    status := SelfGrabbed;
2565                              (False):TASK    status := OtherGrabbed;
2566                              ENDDECISION;
2567                      (Ungivable):
2568                              DECISION token(t)!grabber = u;
2569                              (True): TASK    status := SelfGiving;
2570                              (False):TASK    status := OtherGiving;
2571                              ENDDECISION;
2572                      (Given):
2573                              DECISION token(t)!recipient = u;
2574                              (True): TASK    status := SelfRecipient;
2575                              (False):TASK    status := OtherGiving;
2576                              ENDDECISION;
2577                      (Giving):
2578                              DECISION token(t)!recipient = u;
2579                              (True): TASK    status := SelfRecipient;
2580                              (False):DECISION token(t)!grabber = u;
2581                                      (True): TASK    status := SelfGiving;
2582                                      (False):TASK    status := OtherGiving;
2583                                      ENDDECISION;
2584                              ENDDECISION;
2585                      (Inhibited):
2586                              DECISION u in token(t)!inhibitors;
2587                              (True): TASK    status := SelfInhibited;
2588                              (False):TASK    status := OtherInhibited;
2589                              ENDDECISION;
2590                      ENDDECISION;
2591              ENDDECISION;
2592              TASK    pdu!tokenStatus := status;
2593              RETURN;
2594              ENDPROCEDURE;

2596                                                              /*----------------*/
2597    PROCEDURE       Token_route;                                /*  Token_route   */
```

```
2598    FPAR          u             UserId,                    /*---------------*/
2599          IN/OUT  x             PortalId,
2600                  p             PortalId;

2602          START
2603          COMMENT'Revoke token user if its route x is no longer via p.
2604                  ';
2605          CALL    Route_user(u, x);
2606          DECISION x = p;
2607          (False):DECISION u in uUsed;
2608                  (True): TASK    uRevoke := Incl(u, uRevoke);
2609                  ELSE:ENDDECISION;
2610          ELSE:ENDDECISION;
2611          RETURN;
2612          ENDPROCEDURE;

2614                                                          /*---------------*/
2615    PROCEDURE      Delete_user;                            /*  Delete_user  */
2616    FPAR           u             UserId;                  /*---------------*/

2618          DCL     p             PortalId,
2619                  c             ChannelId,
2620                  cSet          ChannelIdSet,
2621                  a             UserId,
2622                  aSet          UserIdSet,
2623                  q             PortalId,
2624                  t             TokenId,
2625                  tSet          TokenIdSet;
2626          START
2627          COMMENT'Update the information base to delete a user id.
2628                  This has consequences for its channels and tokens.
2629                  Any data still in transit is left undisturbed.
2630                  ';
2631          DECISION u in uUsed;
2632          (False):RETURN;
2633          ELSE:ENDDECISION;
2634          TASK    c := ChannelId(u),
2635                  p := chan(c)!portal,
2636                  uUsed := Del(u, uUsed),
2637                  numUserIds := numUserIds - 1,
2638                  cUsed := Del(c, cUsed),
2639                  cFree := Incl(c, cFree),
2640                  numChannelIds := numChannelIds - 1,
2641                  uConfirm := Del(u, uConfirm),
2642                  cConfirm := Del(c, cConfirm),
2643                  uDetach := Del(u, uDetach),
2644                  uRevoke := Del(u, uRevoke),
2645                  cLeave := Del(c, cLeave);
2646          TASK    cSet := cUsed;
2647           1b :   /* for c in cSet */
2648          DECISION cSet = Empty;
2649          (False):TASK    c := Pick(cSet),
2650                          cSet := Del(c, cSet);
2651                  DECISION portal(p)!kind = Attached and p in chan(c)!joined;
2652                  (True): TASK    chan(c)!joined := Del(p, chan(c)!joined);
2653                          DECISION chan(c)!joined = Empty;
2654                          (True): TASK    cLeave := Incl(c, cLeave);
2655                          ELSE:ENDDECISION;
2656                  ELSE:ENDDECISION;
2657                  DECISION chan(c)!kind = Private;
2658                  (True): DECISION chan(c)!manager = u;
2659                          (True): TASK    chan(c)!manager := 0;
2660                                  DECISION upward = 0;
2661                                  (True): TASK    cDisband := Incl(c, cDisband);
2662                                  ELSE:ENDDECISION;
2663                          ELSE:ENDDECISION;
2664                          TASK    chan(c)!admitted := Del(u, chan(c)!admitted),
2665                                  chan(c)!uMerge := Del(u, chan(c)!uMerge);
2666                          DECISION chan(c)!admitted /= Empty or chan(c)!manager in uUsed;
2667                          (False):DECISION not merging or c in cConfirm;
2668                                  (True): CALL    Delete_channel(c);
2669                                  ELSE:ENDDECISION;
2670                          (True): DECISION p in chan(c)!joined;
2671                                  (True): TASK    aSet := chan(c)!admitted;
2672                                           2b :   /* for a in aSet */
2673                                          DECISION aSet = Empty;
2674                                          (False):TASK    a := Pick(aSet),
2675                                                          aSet := Del(a, aSet);
2676                                                  CALL    Route_user(a, q);
```

```
2677                                        DECISION q = p;
2678                                        (False):JOIN 2b;
2679                                        (True): JOIN 1b;
2680                                        ENDDECISION;
2681                                  ELSE:ENDDECISION;
2682                                  TASK    chan(c)!joined := Del(p, chan(c)!joined);
2683                                  DECISION chan(c)!joined = Empty;
2684                                  (True): TASK    cLeave := Incl(c, cLeave);
2685                                  ELSE:ENDDECISION;
2686                            ELSE:ENDDECISION;
2687                      ENDDECISION;
2688                ELSE:ENDDECISION;
2689                JOIN 1b;
2690          ELSE:ENDDECISION;
2691          TASK    tSet := tUsed;
2692           3b :   /* for t in tSet */
2693          DECISION tSet = Empty;
2694          (False):TASK    t := Pick(tSet),
2695                          tSet := Del(t, tSet);
2696                DECISION token(t)!kind;
2697                (Grabbed, Ungivable):
2698                      DECISION token(t)!grabber = u;
2699                      (True): JOIN 3f;
2700                      ELSE:ENDDECISION;
2701                (Given):
2702                      DECISION token(t)!recipient = u;
2703                      (True): JOIN 3f;
2704                      ELSE:ENDDECISION;
2705                (Giving):
2706                      DECISION token(t)!recipient = u;
2707                      (True): TASK    token(t)!kind := Ungivable;
2708                            DECISION token(t)!grabber in uUsed;
2709                            (False):JOIN 3f;
2710                            (True): DECISION upward = 0;
2711                                  (True): TASK    tReject := Incl(t, tReject);
2712                                  ELSE:ENDDECISION;
2713                            ENDDECISION;
2714                      (False):DECISION token(t)!grabber = u;
2715                            (True): TASK    token(t)!kind := Given;
2716                            ELSE:ENDDECISION;
2717                      ELSE:ENDDECISION;
2718                (Inhibited):
2719                      TASK    token(t)!inhibitors := Del(u, token(t)!inhibitors),
2720                              token(t)!uMerge := Del(u, token(t)!uMerge);
2721                      DECISION token(t)!inhibitors = Empty;
2722                      (True):  3f :
2723                            DECISION not merging or t in tConfirm;
2724                            (True): CALL    Delete_token(t);
2725                            (False):TASK    token(t)!kind := Inhibited,
2726                                            token(t)!inhibitors := Empty;
2727                            ENDDECISION;
2728                      ELSE:ENDDECISION;
2729                ENDDECISION;
2730                JOIN 3b;
2731          ELSE:ENDDECISION;
2732          RETURN;
2733          ENDPROCEDURE;

2735                                                      /*------------------*/
2736    PROCEDURE      Delete_channel;                        /* Delete_channel */
2737    FPAR           c            ChannelId;               /*------------------*/

2739          DCL     u             UserId;
2740          START
2741          COMMENT'Update the information base to delete a channel id.
2742                  ';
2743          DECISION c in cUsed;
2744          (False):RETURN;
2745          ELSE:ENDDECISION;
2746          DECISION chan(c)!kind = UserId;
2747          (True): TASK    u := UserId(c);
2748                CALL    Delete_user(u);
2749          (False):TASK    cUsed := Del(c, cUsed),
2750                          cFree := Incl(c, cFree),
2751                          numChannelIds := numChannelIds - 1,
2752                          cConfirm := Del(c, cConfirm),
2753                          cLeave := Del(c, cLeave),
2754                          cDisband := Del(c, cDisband);
2755          ENDDECISION;
```

```
2756          RETURN;
2757          ENDPROCEDURE;

2759                                                                  /*----------------*/
2760    PROCEDURE       Delete_token;                                    /*  Delete_token  */
2761    FPAR            t               TokenId;                       /*----------------*/

2763          START
2764          COMMENT'Update the information base to delete a token id.
2765                  ';
2766          DECISION t in tUsed;
2767          (False):RETURN;
2768          ELSE:ENDDECISION;
2769          TASK    tUsed := Del(t, tUsed),
2770                  tFree := Incl(t, tFree),
2771                  numTokenIds := numTokenIds - 1,
2772                  tConfirm := Del(t, tConfirm),
2773                  tReject := Del(t, tReject);
2774          RETURN;
2775          ENDPROCEDURE;

2777                                                                  /*---------------*/
2778    PROCEDURE       Purge_users;                                     /*  Purge_users  */
2779    FPAR            uSet            UserIdSet;                     /*---------------*/

2781          DCL     u               UserId;
2782          START
2783          COMMENT'Delete a set of user ids.
2784                  ';
2785           lb :   /* for u in uSet */
2786          DECISION uSet = Empty;
2787          (False):TASK    u := Pick(uSet),
2788                          uSet := Del(u, uSet);
2789                  CALL    Delete_user(u);
2790                  JOIN lb;
2791          ELSE:ENDDECISION;
2792          RETURN;
2793          ENDPROCEDURE;

2795                                                                  /*------------------*/
2796    PROCEDURE       Purge_channels;                                  /*  Purge_channels  */
2797    FPAR            cSet            ChannelIdSet;                  /*------------------*/

2799          DCL     c               ChannelId;
2800          START
2801          COMMENT'Delete a set of channel ids.
2802                  ';
2803           lb :   /* for c in cSet */
2804          DECISION cSet = Empty;
2805          (False):TASK    c := Pick(cSet),
2806                          cSet := Del(c, cSet);
2807                  CALL    Delete_channel(c);
2808                  JOIN lb;
2809          ELSE:ENDDECISION;
2810          RETURN;
2811          ENDPROCEDURE;

2813                                                                  /*----------------*/
2814    PROCEDURE       Purge_tokens;                                    /*  Purge_tokens  */
2815    FPAR            tSet            TokenIdSet;                    /*----------------*/

2817          DCL     t               TokenId;
2818          START
2819          COMMENT'Delete a set of token ids.
2820                  ';
2821           lb :   /* for t in tSet */
2822          DECISION tSet = Empty;
2823          (False):TASK    t := Pick(tSet),
2824                          tSet := Del(t, tSet);
2825                  CALL    Delete_token(t);
2826                  JOIN lb;
2827          ELSE:ENDDECISION;
2828          RETURN;
2829          ENDPROCEDURE;

2831                                                                  /*-----------------*/
2832    PROCEDURE       Output_buffer;                                   /*  Output_buffer  */
2833    FPAR            b               BufferId,                      /*-----------------*/
2834                    p               PortalId;
```

```
2836          DCL     dp              DataPriority,
2837                  pdu             PDUStruct,
2838                  pid             PId;
2839          START
2840          COMMENT'Send the output signal representing an MCSPDU.
2841                  The next must wait until PDU.ready is received.
2842                  ';
2843          TASK    dp := buffer(b)!dataPriority,
2844                  pdu := buffer(b)!pdu,
2845                  pid := portal(p)!pids(dp);
2846          DECISION p = upward and pdu!kind = SDin;
2847          (True): TASK    pdu!kind := SDrq;
2848          ELSE:ENDDECISION;
2849          DECISION pdu!kind;
2850          (PDin): OUTPUT  PDin(pdu) TO pid;
2851          (EDrq): OUTPUT  EDrq(pdu) TO pid;
2852          (MCrq): OUTPUT  MCrq(pdu) TO pid;
2853          (MCcf): OUTPUT  MCcf(pdu) TO pid;
2854          (PCin): OUTPUT  PCin(pdu) TO pid;
2855          (MTrq): OUTPUT  MTrq(pdu) TO pid;
2856          (MTcf): OUTPUT  MTcf(pdu) TO pid;
2857          (PTin): OUTPUT  PTin(pdu) TO pid;
2858          (DPum): OUTPUT  DPum(pdu) TO pid;
2859          (RJum): OUTPUT  RJum(pdu) TO pid;
2860          (AUrq): OUTPUT  AUrq(pdu) TO pid;
2861          (AUcf): OUTPUT  AUcf(pdu) TO pid;
2862          (DUrq): OUTPUT  DUrq(pdu) TO pid;
2863          (DUin): OUTPUT  DUin(pdu) TO pid;
2864          (CJrq): OUTPUT  CJrq(pdu) TO pid;
2865          (CJcf): OUTPUT  CJcf(pdu) TO pid;
2866          (CLrq): OUTPUT  CLrq(pdu) TO pid;
2867          (CCrq): OUTPUT  CCrq(pdu) TO pid;
2868          (CCcf): OUTPUT  CCcf(pdu) TO pid;
2869          (CDrq): OUTPUT  CDrq(pdu) TO pid;
2870          (CDin): OUTPUT  CDin(pdu) TO pid;
2871          (CArq): OUTPUT  CArq(pdu) TO pid;
2872          (CAin): OUTPUT  CAin(pdu) TO pid;
2873          (CErq): OUTPUT  CErq(pdu) TO pid;
2874          (CEin): OUTPUT  CEin(pdu) TO pid;
2875          (SDrq): OUTPUT  SDrq(pdu) TO pid;
2876          (SDin): OUTPUT  SDin(pdu) TO pid;
2877          (USrq): OUTPUT  USrq(pdu) TO pid;
2878          (USin): OUTPUT  USin(pdu) TO pid;
2879          (TGrq): OUTPUT  TGrq(pdu) TO pid;
2880          (TGcf): OUTPUT  TGcf(pdu) TO pid;
2881          (TIrq): OUTPUT  TIrq(pdu) TO pid;
2882          (TIcf): OUTPUT  TIcf(pdu) TO pid;
2883          (TVrq): OUTPUT  TVrq(pdu) TO pid;
2884          (TVin): OUTPUT  TVin(pdu) TO pid;
2885          (TVrs): OUTPUT  TVrs(pdu) TO pid;
2886          (TVcf): OUTPUT  TVcf(pdu) TO pid;
2887          (TPrq): OUTPUT  TPrq(pdu) TO pid;
2888          (TPin): OUTPUT  TPin(pdu) TO pid;
2889          (TRrq): OUTPUT  TRrq(pdu) TO pid;
2890          (TRcf): OUTPUT  TRcf(pdu) TO pid;
2891          (TTrq): OUTPUT  TTrq(pdu) TO pid;
2892          (TTcf): OUTPUT  TTcf(pdu) TO pid;
2893          ENDDECISION;
2894          RETURN;
2895          ENDPROCEDURE;

2897          /* Input transitions */

2899          DCL     p               PortalId,
2900                  dp              DataPriority,
2901                  pdu             PDUStruct,
2902                  pKind           PortalKind,
2903                  pids            PIdByPri,
2904                  reason          Reason;
2905          START
2906          COMMENT'The state machine contains a single state.
2907                  ';
2908          CALL    Initialize_resources;
2909          NEXTSTATE ~;

2911    STATE ~;INPUT   Open.portal(p, pKind, pids);
2912          CALL    Open_portal(p, pKind, pids);
2913          NEXTSTATE -;
```

```
2915    STATE ~;INPUT   Time.portal(p);
2916          CALL    Time_portal(p);
2917          NEXTSTATE -;

2919    STATE ~;INPUT   Drop.portal(p, reason);
2920          CALL    Drop_portal(p, reason);
2921          NEXTSTATE -;

2923    STATE ~;INPUT   Shut.portal(p);
2924          CALL    Shut_portal(p);
2925          DECISION pUsed = Empty;
2926          (False):NEXTSTATE -;
2927          (True): STOP;
2928          ENDDECISION;

2930    STATE ~;INPUT   PDU.ready(dp);
2931          CALL    PDU_ready(dp);
2932          NEXTSTATE -;

2934    STATE ~;INPUT   PDin(pdu); TASK pdu!kind := PDin; CALL Input_PDU(pdu); NEXTSTATE -;
2935    STATE ~;INPUT   EDrq(pdu); TASK pdu!kind := EDrq; CALL Input_PDU(pdu); NEXTSTATE -;
2936    STATE ~;INPUT   MCrq(pdu); TASK pdu!kind := MCrq; CALL Input_PDU(pdu); NEXTSTATE -;
2937    STATE ~;INPUT   MCcf(pdu); TASK pdu!kind := MCcf; CALL Input_PDU(pdu); NEXTSTATE -;
2938    STATE ~;INPUT   PCin(pdu); TASK pdu!kind := PCin; CALL Input_PDU(pdu); NEXTSTATE -;
2939    STATE ~;INPUT   MTrq(pdu); TASK pdu!kind := MTrq; CALL Input_PDU(pdu); NEXTSTATE -;
2940    STATE ~;INPUT   MTcf(pdu); TASK pdu!kind := MTcf; CALL Input_PDU(pdu); NEXTSTATE -;
2941    STATE ~;INPUT   PTin(pdu); TASK pdu!kind := PTin; CALL Input_PDU(pdu); NEXTSTATE -;
2942    STATE ~;INPUT   DPum(pdu); TASK pdu!kind := DPum; CALL Input_PDU(pdu); NEXTSTATE -;
2943    STATE ~;INPUT   RJum(pdu); TASK pdu!kind := RJum; CALL Input_PDU(pdu); NEXTSTATE -;
2944    STATE ~;INPUT   AUrq(pdu); TASK pdu!kind := AUrq; CALL Input_PDU(pdu); NEXTSTATE -;
2945    STATE ~;INPUT   AUcf(pdu); TASK pdu!kind := AUcf; CALL Input_PDU(pdu); NEXTSTATE -;
2946    STATE ~;INPUT   DUrq(pdu); TASK pdu!kind := DUrq; CALL Input_PDU(pdu); NEXTSTATE -;
2947    STATE ~;INPUT   DUin(pdu); TASK pdu!kind := DUin; CALL Input_PDU(pdu); NEXTSTATE -;
2948    STATE ~;INPUT   CJrq(pdu); TASK pdu!kind := CJrq; CALL Input_PDU(pdu); NEXTSTATE -;
2949    STATE ~;INPUT   CJcf(pdu); TASK pdu!kind := CJcf; CALL Input_PDU(pdu); NEXTSTATE -;
2950    STATE ~;INPUT   CLrq(pdu); TASK pdu!kind := CLrq; CALL Input_PDU(pdu); NEXTSTATE -;
2951    STATE ~;INPUT   CCrq(pdu); TASK pdu!kind := CCrq; CALL Input_PDU(pdu); NEXTSTATE -;
2952    STATE ~;INPUT   CCcf(pdu); TASK pdu!kind := CCcf; CALL Input_PDU(pdu); NEXTSTATE -;
2953    STATE ~;INPUT   CDrq(pdu); TASK pdu!kind := CDrq; CALL Input_PDU(pdu); NEXTSTATE -;
2954    STATE ~;INPUT   CDin(pdu); TASK pdu!kind := CDin; CALL Input_PDU(pdu); NEXTSTATE -;
2955    STATE ~;INPUT   CArq(pdu); TASK pdu!kind := CArq; CALL Input_PDU(pdu); NEXTSTATE -;
2956    STATE ~;INPUT   CAin(pdu); TASK pdu!kind := CAin; CALL Input_PDU(pdu); NEXTSTATE -;
2957    STATE ~;INPUT   CErq(pdu); TASK pdu!kind := CErq; CALL Input_PDU(pdu); NEXTSTATE -;
2958    STATE ~;INPUT   CEin(pdu); TASK pdu!kind := CEin; CALL Input_PDU(pdu); NEXTSTATE -;
2959    STATE ~;INPUT   SDrq(pdu); TASK pdu!kind := SDrq; CALL Input_PDU(pdu); NEXTSTATE -;
2960    STATE ~;INPUT   SDin(pdu); TASK pdu!kind := SDin; CALL Input_PDU(pdu); NEXTSTATE -;
2961    STATE ~;INPUT   USrq(pdu); TASK pdu!kind := USrq; CALL Input_PDU(pdu); NEXTSTATE -;
2962    STATE ~;INPUT   USin(pdu); TASK pdu!kind := USin; CALL Input_PDU(pdu); NEXTSTATE -;
2963    STATE ~;INPUT   TGrq(pdu); TASK pdu!kind := TGrq; CALL Input_PDU(pdu); NEXTSTATE -;
2964    STATE ~;INPUT   TGcf(pdu); TASK pdu!kind := TGcf; CALL Input_PDU(pdu); NEXTSTATE -;
2965    STATE ~;INPUT   TIrq(pdu); TASK pdu!kind := TIrq; CALL Input_PDU(pdu); NEXTSTATE -;
2966    STATE ~;INPUT   TIcf(pdu); TASK pdu!kind := TIcf; CALL Input_PDU(pdu); NEXTSTATE -;
2967    STATE ~;INPUT   TVrq(pdu); TASK pdu!kind := TVrq; CALL Input_PDU(pdu); NEXTSTATE -;
2968    STATE ~;INPUT   TVin(pdu); TASK pdu!kind := TVin; CALL Input_PDU(pdu); NEXTSTATE -;
2969    STATE ~;INPUT   TVrs(pdu); TASK pdu!kind := TVrs; CALL Input_PDU(pdu); NEXTSTATE -;
2970    STATE ~;INPUT   TVcf(pdu); TASK pdu!kind := TVcf; CALL Input_PDU(pdu); NEXTSTATE -;
2971    STATE ~;INPUT   TPrq(pdu); TASK pdu!kind := TPrq; CALL Input_PDU(pdu); NEXTSTATE -;
2972    STATE ~;INPUT   TPin(pdu); TASK pdu!kind := TPin; CALL Input_PDU(pdu); NEXTSTATE -;
2973    STATE ~;INPUT   TRrq(pdu); TASK pdu!kind := TRrq; CALL Input_PDU(pdu); NEXTSTATE -;
2974    STATE ~;INPUT   TRcf(pdu); TASK pdu!kind := TRcf; CALL Input_PDU(pdu); NEXTSTATE -;
2975    STATE ~;INPUT   TTrq(pdu); TASK pdu!kind := TTrq; CALL Input_PDU(pdu); NEXTSTATE -;
2976    STATE ~;INPUT   TTcf(pdu); TASK pdu!kind := TTcf; CALL Input_PDU(pdu); NEXTSTATE -;

2978    ENDPROCESS;
```

付録５　終端プロセスのＳＤＬ記述

（ＪＴ－Ｔ１２５に対する）

```
1   PROCESS Endpoint;
 2    FPAR    tcId           TCEndpointId,          /* incoming TC if not Null */
 3            localTSAP      TSAPAddress,           /* own transport address */
 4            remoteTSAP     TSAPAddress,           /* other transport address */
 5            givenQOS       TransportQOS,          /* target or offered QOS */
 6            minQOS         TransportQOS,          /* minimum acceptable QOS */
 7            parameters     DomainParameters;      /* values established or null */

 9            /* Data declarations */

11    DCL     control        PId,                   /* single Control process */
12            domain         PId;                   /* selected Domain process */

14            /* Input transitions */

16            DCL     localDomain     DomainSelector,
17                    remoteDomain    DomainSelector,
18                    upward          Boolean,
19                    targetParms     DomainParameters,
20                    minParms        DomainParameters,
21                    maxParms        DomainParameters,
22                    userData        UserData,
23                    result          Result,
24                    ccId            Natural,
25                    label           Natural,
26                    tsdu            TSDU,
27                    dp              DataPriority,
28                    pdu             PDUStruct;
29            START
30            COMMENT'State machine:              NEXTSTATE
31                           1  connecting   * . 2 . . . 6
32                           2  connbusy     * . 2 3 . 5 6
33                           3  connready      . 2 3 . 5 6
34                           4  busy           . . . 4 5 6
35                           5  ready          . . . 4 5 6
36                           6  disconnected   . . . . . 6
37
38            TASK    control := PARENT,
39                    label := 0;
40            DECISION tcId = Null;
41            (True): OUTPUT  T.Connect.request(label, localTSAP, remoteTSAP,
42                                                    givenQOS, minQOS);
43                    NEXTSTATE connecting;
44            (False):OUTPUT  T.Connect.response(tcId, givenQOS);
45                    OUTPUT  T.ready(tcId);
46                    NEXTSTATE connbusy;
47            ENDDECISION;

49    STATE *;
50            INPUT   Exit;
51            STOP;

53    STATE *;
54            INPUT   T.Disconnect.indication(label, tcId);
55            OUTPUT  Quit TO control;
56            NEXTSTATE disconnected;

58    STATE connecting;
59            INPUT   T.Connect.confirm(label, tcId, givenQOS);
60            OUTPUT  T.ready(tcId);
61            NEXTSTATE connbusy;

63    STATE connecting;
64            SAVE    *;  /* await outcome of TC */

66    STATE connbusy, connready, busy, ready;
67            INPUT   *;
68            OUTPUT  T.Disconnect.request(tcId);
69            OUTPUT  Quit TO control;
70            NEXTSTATE disconnected;

72    STATE connbusy;
73            INPUT   T.ready(tcId);
```

```
74          NEXTSTATE connready;

76    STATE connbusy;
77          SAVE    Connect.Initial, Connect.Response, Connect.Additional, Connect.Result;

79    STATE connready;
80          INPUT   Connect.Initial(localDomain, remoteDomain, upward,
81                  targetParms, minParms, maxParms, userData);
82          TASK'tsdu := encode Connect-Initial using BER';
83           1b :
84          OUTPUT  T.Data.request(tcId, tsdu);
85          NEXTSTATE connbusy;

87    STATE connready;
88          INPUT   Connect.Response(result, ccId, parameters, userData);
89          TASK'tsdu := encode Connect-Response using BER';
90          JOIN 1b;

92    STATE connready;
93          INPUT   Connect.Additional(ccId, dp);
94          TASK'tsdu := encode Connect-Additional using BER';
95          JOIN 1b;

97    STATE connready;
98          INPUT   Connect.Result(result);
99          TASK'tsdu := encode Connect-Result using BER';
100         JOIN 1b;

102   STATE connbusy, connready;
103         INPUT   T.Data.indication(tcId, tsdu);
104         TASK'connect MCSPDU := decode tsdu using BER';
105         DECISION'connect MCSPDU';
106         ('Connect-Initial'):
107               OUTPUT  Connect.Initial(localDomain, remoteDomain, upward,
108                     targetParms, minParms, maxParms, userData) TO control;
109               NEXTSTATE -;
110         ('Connect-Response'):
111               OUTPUT  Connect.Response(result, ccId, parameters, userData) TO
control;
112               NEXTSTATE -;
113         ('Connect-Additional'):
114               OUTPUT  Connect.Additional(ccId, dp) TO control;
115               NEXTSTATE -;
116         ('Connect-Result'):
117               OUTPUT  Connect.Result(result) TO control;
118               NEXTSTATE -;
119         ELSE:
120               OUTPUT  T.Disconnect.request(tcId);
121               OUTPUT  Quit TO control;
122               NEXTSTATE disconnected;
123         ENDDECISION;

125   STATE connbusy;
126         INPUT   PDU.ready(dp);
127         TASK    domain := SENDER;
128         OUTPUT  T.ready(tcId);
129         NEXTSTATE ready;

131   STATE connready;
132         INPUT   PDU.ready(dp);
133         TASK    domain := SENDER;
134         OUTPUT  PDU.ready(dp) TO domain;
135         OUTPUT  T.ready(tcId);
136         NEXTSTATE ready;

138   STATE busy;
139         INPUT   PDU.ready(dp);
140         OUTPUT  T.ready(tcId);
141         NEXTSTATE ready;

143   STATE busy, ready;
144         INPUT   T.ready(tcId);
145         OUTPUT  PDU.ready(dp) TO domain;
146         NEXTSTATE -;

148   STATE busy, ready;
149         INPUT   PDin(pdu); TASK pdu!kind := PDin; JOIN 2f;
150         INPUT   EDrq(pdu); TASK pdu!kind := EDrq; JOIN 2f;
151         INPUT   MCrq(pdu); TASK pdu!kind := MCrq; JOIN 2f;
```

```
152         INPUT   MCcf(pdu); TASK pdu!kind := MCcf; JOIN 2f;
153         INPUT   PCin(pdu); TASK pdu!kind := PCin; JOIN 2f;
154         INPUT   MTrq(pdu); TASK pdu!kind := MTrq; JOIN 2f;
155         INPUT   MTcf(pdu); TASK pdu!kind := MTcf; JOIN 2f;
156         INPUT   PTin(pdu); TASK pdu!kind := PTin; JOIN 2f;
157         INPUT   DPum(pdu); TASK pdu!kind := DPum; JOIN 2f;
158         INPUT   RJum(pdu); TASK pdu!kind := RJum; JOIN 2f;
159         INPUT   AUrq(pdu); TASK pdu!kind := AUrq; JOIN 2f;
160         INPUT   AUcf(pdu); TASK pdu!kind := AUcf; JOIN 2f;
161         INPUT   DUrq(pdu); TASK pdu!kind := DUrq; JOIN 2f;
162         INPUT   DUin(pdu); TASK pdu!kind := DUin; JOIN 2f;
163         INPUT   CJrq(pdu); TASK pdu!kind := CJrq; JOIN 2f;
164         INPUT   CJcf(pdu); TASK pdu!kind := CJcf; JOIN 2f;
165         INPUT   CLrq(pdu); TASK pdu!kind := CLrq; JOIN 2f;
166         INPUT   CCrq(pdu); TASK pdu!kind := CCrq; JOIN 2f;
167         INPUT   CCcf(pdu); TASK pdu!kind := CCcf; JOIN 2f;
168         INPUT   CDrq(pdu); TASK pdu!kind := CDrq; JOIN 2f;
169         INPUT   CDin(pdu); TASK pdu!kind := CDin; JOIN 2f;
170         INPUT   CArq(pdu); TASK pdu!kind := CArq; JOIN 2f;
171         INPUT   CAin(pdu); TASK pdu!kind := CAin; JOIN 2f;
172         INPUT   CErq(pdu); TASK pdu!kind := CErq; JOIN 2f;
173         INPUT   CEin(pdu); TASK pdu!kind := CEin; JOIN 2f;
174         INPUT   SDrq(pdu); TASK pdu!kind := SDrq; JOIN 2f;
175         INPUT   SDin(pdu); TASK pdu!kind := SDin; JOIN 2f;
176         INPUT   USrq(pdu); TASK pdu!kind := USrq; JOIN 2f;
177         INPUT   USin(pdu); TASK pdu!kind := USin; JOIN 2f;
178         INPUT   TGrq(pdu); TASK pdu!kind := TGrq; JOIN 2f;
179         INPUT   TGcf(pdu); TASK pdu!kind := TGcf; JOIN 2f;
180         INPUT   TIrq(pdu); TASK pdu!kind := TIrq; JOIN 2f;
181         INPUT   TIcf(pdu); TASK pdu!kind := TIcf; JOIN 2f;
182         INPUT   TVrq(pdu); TASK pdu!kind := TVrq; JOIN 2f;
183         INPUT   TVin(pdu); TASK pdu!kind := TVin; JOIN 2f;
184         INPUT   TVrs(pdu); TASK pdu!kind := TVrs; JOIN 2f;
185         INPUT   TVcf(pdu); TASK pdu!kind := TVcf; JOIN 2f;
186         INPUT   TPrq(pdu); TASK pdu!kind := TPrq; JOIN 2f;
187         INPUT   TPin(pdu); TASK pdu!kind := TPin; JOIN 2f;
188         INPUT   TRrq(pdu); TASK pdu!kind := TRrq; JOIN 2f;
189         INPUT   TRcf(pdu); TASK pdu!kind := TRcf; JOIN 2f;
190         INPUT   TTrq(pdu); TASK pdu!kind := TTrq; JOIN 2f;
191         INPUT   TTcf(pdu); TASK pdu!kind := TTcf;
192          2f :
193         TASK'tsdu := encode pdu using BER or PER,
194             depending on parameters!protocolVersion';
195         OUTPUT  T.Data.request(tcId, tsdu);
196         NEXTSTATE -;

198   STATE ready;
199         INPUT   T.Data.indication(tcId, tsdu);
200         TASK'pdu := decode tsdu using BER or PER,
201             depending on parameters!protocolVersion';
202         DECISION pdu!kind;
203         (RJum): OUTPUT  RJum(pdu) TO control;
204                 OUTPUT  T.ready(tcId);
205                 NEXTSTATE -;
206         (PDin): OUTPUT  PDin(pdu) TO domain; NEXTSTATE busy;
207         (EDrq): OUTPUT  EDrq(pdu) TO domain; NEXTSTATE busy;
208         (MCrq): OUTPUT  MCrq(pdu) TO domain; NEXTSTATE busy;
209         (MCcf): OUTPUT  MCcf(pdu) TO domain; NEXTSTATE busy;
210         (PCin): OUTPUT  PCin(pdu) TO domain; NEXTSTATE busy;
211         (MTrq): OUTPUT  MTrq(pdu) TO domain; NEXTSTATE busy;
212         (MTcf): OUTPUT  MTcf(pdu) TO domain; NEXTSTATE busy;
213         (PTin): OUTPUT  PTin(pdu) TO domain; NEXTSTATE busy;
214         (DPum): OUTPUT  DPum(pdu) TO domain; NEXTSTATE busy;
215         (AUrq): OUTPUT  AUrq(pdu) TO domain; NEXTSTATE busy;
216         (AUcf): OUTPUT  AUcf(pdu) TO domain; NEXTSTATE busy;
217         (DUrq): OUTPUT  DUrq(pdu) TO domain; NEXTSTATE busy;
218         (DUin): OUTPUT  DUin(pdu) TO domain; NEXTSTATE busy;
219         (CJrq): OUTPUT  CJrq(pdu) TO domain; NEXTSTATE busy;
220         (CJcf): OUTPUT  CJcf(pdu) TO domain; NEXTSTATE busy;
221         (CLrq): OUTPUT  CLrq(pdu) TO domain; NEXTSTATE busy;
222         (CCrq): OUTPUT  CCrq(pdu) TO domain; NEXTSTATE busy;
223         (CCcf): OUTPUT  CCcf(pdu) TO domain; NEXTSTATE busy;
224         (CDrq): OUTPUT  CDrq(pdu) TO domain; NEXTSTATE busy;
225         (CDin): OUTPUT  CDin(pdu) TO domain; NEXTSTATE busy;
226         (CArq): OUTPUT  CArq(pdu) TO domain; NEXTSTATE busy;
227         (CAin): OUTPUT  CAin(pdu) TO domain; NEXTSTATE busy;
228         (CErq): OUTPUT  CErq(pdu) TO domain; NEXTSTATE busy;
229         (CEin): OUTPUT  CEin(pdu) TO domain; NEXTSTATE busy;
230         (SDrq): OUTPUT  SDrq(pdu) TO domain; NEXTSTATE busy;
```

```
231         (SDin): OUTPUT  SDin(pdu) TO domain; NEXTSTATE busy;
232         (USrq): OUTPUT  USrq(pdu) TO domain; NEXTSTATE busy;
233         (USin): OUTPUT  USin(pdu) TO domain; NEXTSTATE busy;
234         (TGrq): OUTPUT  TGrq(pdu) TO domain; NEXTSTATE busy;
235         (TGcf): OUTPUT  TGcf(pdu) TO domain; NEXTSTATE busy;
236         (TIrq): OUTPUT  TIrq(pdu) TO domain; NEXTSTATE busy;
237         (TIcf): OUTPUT  TIcf(pdu) TO domain; NEXTSTATE busy;
238         (TVrq): OUTPUT  TVrq(pdu) TO domain; NEXTSTATE busy;
239         (TVin): OUTPUT  TVin(pdu) TO domain; NEXTSTATE busy;
240         (TVrs): OUTPUT  TVrs(pdu) TO domain; NEXTSTATE busy;
241         (TVcf): OUTPUT  TVcf(pdu) TO domain; NEXTSTATE busy;
242         (TPrq): OUTPUT  TPrq(pdu) TO domain; NEXTSTATE busy;
243         (TPin): OUTPUT  TPin(pdu) TO domain; NEXTSTATE busy;
244         (TRrq): OUTPUT  TRrq(pdu) TO domain; NEXTSTATE busy;
245         (TRcf): OUTPUT  TRcf(pdu) TO domain; NEXTSTATE busy;
246         (TTrq): OUTPUT  TTrq(pdu) TO domain; NEXTSTATE busy;
247         (TTcf): OUTPUT  TTcf(pdu) TO domain; NEXTSTATE busy;
248         ELSE:
249               TASK'pdu!initialOctets := truncate tsdu';
250               TASK'pdu!diagnostic := DC_invalid_?ER_encoding';
251               OUTPUT  RJum(pdu) TO domain;
252               NEXTSTATE busy;
253         ENDDECISION;

255   STATE disconnected;
256         INPUT   Quit;
257         OUTPUT  Quit TO control;
258         NEXTSTATE disconnected;

260   ENDPROCESS;
```

付録６　アタッチメント プロセスのＳＤＬ記述

（ＪＴ－Ｔ１２５に対する）

```
1   PROCESS Attachment;
  2   FPAR   label          Natural,              /* for MCS.Attach.User.confirm */
  3          parameters     DomainParameters;      /* values established in domain */

  5          /* Type definitions */

  7   NEWTYPE        MCSRequest
  8   STRUCT
  9          kind           PDUKind;           /* SDrq, USrq, CArq, CErq */
 10          channelId      ChannelId;          /* parameter of request */
 11          segmentation   Segmentation;       /* parameter of request */
 12          userData       UserData;          /* parameter of request */
 13          offset         Integer;           /* octets sent so far */
 14          userIds        UserIdSet;         /* remaining to affect */
 15   ENDNEWTYPE;

 17   NEWTYPE        PrioritySet        SetOf(DataPriority);
 18   ENDNEWTYPE;

 20   NEWTYPE        MCSRequestByPri    Array(DataPriority, MCSRequest);
 21   ENDNEWTYPE;
 22   NEWTYPE        PDUStructByPri     Array(DataPriority, PDUStruct);
 23   ENDNEWTYPE;

 25          /* Data declarations */

 27   DCL    maId           MCSAttachmentId,    /* this Attachment process */
 28          control        PId,               /* single Control process */
 29          domain         PId;               /* selected Domain process */

 31   DCL    user           UserId;            /* unique id of attached user */

 33   DCL    cJoined        ChannelIdSet,       /* channels the user has joined */
 34          cConvened      ChannelIdSet,       /* channels the user has convened */
 35          cAdmitted      ChannelIdSet;       /* channels user was admitted to */

 37   DCL    tPossessed     TokenIdSet,         /* tokens grabbed or inhibited */
 38          tRecipient     TokenIdSet;         /* tokens given waiting response */

 40   DCL    uPending       PrioritySet,        /* request is pending at 0..3 */
 41          mcsreq         MCSRequestByPri,    /* content of segmented request */
 42          dReady         PrioritySet;        /* domain MCSPDU allowed 0..? */

 44   DCL    dPending       PrioritySet,        /* SDin or USin pending at 0..3 */
 45          mcspdu         PDUStructByPri,     /* content of the domain MCSPDU */
 46          uReady         PrioritySet;        /* data indication allowed 0..3 */

 48          /* Procedure decomposition */

 50          /*     Segment_request        (dp)
 51                 Indicate_data
 52                 Track_token            (t, status)     */

 54                                                        /*-------------------*/
 55   PROCEDURE      Segment_request;                        /*  Segment_request  */
 56   FPAR           dp      DataPriority;                  /*-------------------*/

 58          DCL    udp     DataPriority,
 59                 req     MCSRequest,
 60                 pdu     PDUStruct,
 61                 b       Boolean,
 62                 m       Integer,
 63                 n       Integer,
 64                 u       UserId;
 65          START
 66          COMMENT'Feed domain process the next segment of user data
 67                 or the next subset of private channel user ids,
 68                 if ready, for the specified transport priority.
 69                 ';
 70          DECISION dp in dReady;
 71          (False):RETURN;
 72          ELSE:ENDDECISION;
 73          TASK   udp := dp;
```

```
74          1b :   /* for udp = dp..? */
75          DECISION udp = dp or (udp >= parameters!numPriorities and udp < 4);
76          (False):RETURN;
77          (True): DECISION udp in uPending;
78                 (False):TASK    udp := udp + 1;
79                         JOIN 1b;
80                 ELSE:ENDDECISION;
81          ENDDECISION;
82          TASK    dReady := Del(dp, dReady),
83                 dp := udp,
84                 req := mcsreq(dp),
85                 pdu!initiator := user,
86                 pdu!channelId := req!channelId;
87          DECISION req!kind;
88          (SDrq, USrq):
89                 TASK    pdu!dataPriority := dp,
90                        b := req!segmentation!begin,
91                        pdu!segmentation!begin := IF req!offset = 0 THEN b ELSE False FI,
92                        m := parameters!maxMCSPDUsize,
93                        m := IF parameters!protocolVersion = 1 THEN m - 24 ELSE m - 8 FI,
94                        n := Length(req!userData) - req!offset,
95                        n := IF n > m THEN m ELSE n FI,
96                        pdu!userData := Substring(req!userData, 1 + req!offset, n),
97                        req!offset := req!offset + n,
98                        n := Length(req!userData) - req!offset,
99                        b := req!segmentation!end,
100                       pdu!segmentation!end := IF n = 0 THEN b ELSE False FI,
101                       mcsreq(dp)!offset := req!offset;
102         (CArq, CErq):
103                TASK    pdu!userIds := Empty,
104                       n := 0,
105                       m := parameters!maxMCSPDUsize,
106                       m := IF parameters!protocolVersion = 1 THEN (m - 16) / 4
107                                    ELSE (m - 7) / 2 FI;
108                 2b :   /* while n < m */
109                DECISION req!userIds = Empty;
110                (True): TASK    n := 0;
111                (False):TASK    u := Pick(req!userIds),
112                              req!userIds := Del(u, req!userIds);
113                       DECISION u >= 1001;
114                       (True): TASK    pdu!userIds := Incl(u, pdu!userIds);
115                       ELSE:ENDDECISION;
116                       TASK    n := n + 1;
117                       DECISION n < m;
118                       (True): JOIN 2b;
119                       ELSE:ENDDECISION;
120                ENDDECISION;
121         ENDDECISION;
122         DECISION req!kind;
123         (SDrq): OUTPUT  SDrq(pdu) TO domain;
124         (USrq): OUTPUT  USrq(pdu) TO domain;
125         (CArq): OUTPUT  CArq(pdu) TO domain;
126         (CErq): OUTPUT  CErq(pdu) TO domain;
127         ENDDECISION;
128         DECISION n = 0;
129         (True): TASK    uPending := Del(dp, uPending);
130                DECISION req!kind;
131                (SDrq, USrq):
132                       OUTPUT  MCS.ready(maId, dp);
133                ELSE:ENDDECISION;
134         ELSE:ENDDECISION;
135         RETURN;
136         ENDPROCEDURE;

138                                                          /*-----------------*/
139   PROCEDURE      Indicate_data;                          /*  Indicate_data  */
140                                                          /*-----------------*/
141         DCL     dp      DataPriority,
142                pdu     PDUStruct;
143         START
144         COMMENT'Indicate the receipt of user data, if ready
145                and none pending at higher priorities.
146                ';
147         TASK    dp := 0;
148          1b :   /* for dp = 0..3 */
149         DECISION dp < 4 and (dp in uReady or not dp in dPending);
150         (False):RETURN;
151         (True): DECISION dp in dPending;
152                (False):TASK    dp := dp + 1;
```

```
153                     JOIN 1b;
154                ELSE:ENDDECISION;
155          ENDDECISION;
156          TASK    dPending := Del(dp, dPending),
157                 pdu := mcspdu(dp);
158          DECISION pdu!kind;
159          (SDin): OUTPUT  MCS.Send.Data.indication(maId, pdu!channelId,
160                     pdu!dataPriority, pdu!initiator, pdu!segmentation,
pdu!userData);
161          (USin): OUTPUT  MCS.Uniform.Send.Data.indication(maId, pdu!channelId,
162                     pdu!dataPriority, pdu!initiator, pdu!segmentation,
pdu!userData);
163          ENDDECISION;
164          TASK    uReady := Del(dp, uReady),
165                 dp := IF dp < parameters!numPriorities THEN dp
166                           ELSE parameters!numPriorities - 1 FI;
167          OUTPUT  PDU.ready(dp) TO domain;
168          JOIN 1b;
169          ENDPROCEDURE;

171                                                          /*---------------*/
172    PROCEDURE      Track_token;                             /*  Track_token  */
173    FPAR           t      TokenId,                         /*---------------*/
174                   status  TokenStatus;

176          START
177          COMMENT'Condense status into possessed or recipient.
178                  ';
179          TASK    tPossessed := Del(t, tPossessed),
180                 tRecipient := Del(t, tRecipient);
181          DECISION status;
182          (SelfGrabbed, SelfInhibited, SelfGiving):
183                 TASK    tPossessed := Incl(t, tPossessed);
184          (SelfRecipient):
185                 TASK    tRecipient := Incl(t, tRecipient);
186          ELSE:ENDDECISION;
187          RETURN;
188          ENDPROCEDURE;

190          /* Input transitions */

192          DCL     dp              DataPriority,
193                 pdu             PDUStruct,
194                 c               ChannelId,
195                 cSet            ChannelIdSet,
196                 t               TokenId,
197                 u               UserId,
198                 uSet            UserIdSet,
199                 segmentation    Segmentation,
200                 userData        UserData,
201                 result          Result,
202                 kind            PDUKind,
203                 reason          Reason;
204          START
205          COMMENT'State machine:          NEXTSTATE
206                        1  initial    * . 2 . . . 6
207                        2  attaching    . 2 3 4 . 6
208                        3  busy         . . 3 4 5 6
209                        4  ready        . . 3 4 . 6
210                        5  detaching     . . . . 5 6
211                        6  detached      . . . . . 6
212                  ';
213          TASK    maId := SELF,
214                 control := PARENT,
215                 user := 0;
216          NEXTSTATE initial;

218    STATE *;
219           INPUT   Exit;
220           STOP;

222    STATE initial, attaching;
223           INPUT   *;
224           OUTPUT  MCS.Attach.User.confirm(label, RT_unspecified_failure, maId, user);
225           OUTPUT  Quit TO control;
226           NEXTSTATE detached;

228    STATE initial, attaching;
229           INPUT   Quit;
```

```
230         OUTPUT  MCS.Attach.User.confirm(label, RT_domain_disconnected, maId, user);
231         OUTPUT  Quit TO control;
232         NEXTSTATE detached;

234   STATE initial;
235         INPUT   PDU.ready(dp);
236         TASK    domain := SENDER;
237         DECISION dp = 0;
238         (False):TASK    dReady := Incl(dp, dReady);
239               NEXTSTATE -;
240         ELSE:ENDDECISION;
241         OUTPUT  PDU.ready(0) TO domain;
242         OUTPUT  AUrq(pdu) TO domain;
243         NEXTSTATE attaching;

245   STATE attaching;
246         INPUT   PDU.ready(dp);
247         TASK    dReady := Incl(dp, dReady);
248         NEXTSTATE -;

250   STATE attaching;
251         INPUT   PCin(pdu), PTin(pdu), DUin(pdu);
252         /* no action */
253         NEXTSTATE -;

255   STATE attaching;
256         INPUT   AUcf(pdu);
257         TASK    user := pdu!initiator;
258         OUTPUT  MCS.Attach.User.confirm(label, pdu!result, maId, user);
259         DECISION pdu!result = RT_successful;
260         (False):OUTPUT  Quit TO control;
261               NEXTSTATE detached;
262         (True): TASK    dp := 0;
263                 1b :   /* for dp = 0..3 */
264               DECISION dp < 4;
265               (True): OUTPUT  MCS.ready(maId, dp);
266                     DECISION dp < parameters!numPriorities;
267                     (True): OUTPUT  PDU.ready(dp) TO domain;
268                     ELSE:ENDDECISION;
269                     TASK    dp := dp + 1;
270                     JOIN 1b;
271               ELSE:ENDDECISION;
272               DECISION 0 in dReady;
273               (False):NEXTSTATE busy;
274               (True): NEXTSTATE ready;
275               ENDDECISION;
276         ENDDECISION;

278   STATE busy, ready;
279         INPUT   Quit;
280         OUTPUT  MCS.Detach.User.indication(maId, user, RN_provider_initiated);
281         OUTPUT  Quit TO control;
282         NEXTSTATE detached;

284   STATE busy, ready;
285         INPUT   MCS.ready(maId, dp);
286         TASK    uReady := Incl(dp, uReady);
287         CALL    Indicate_data;
288         NEXTSTATE -;

290   STATE busy;
291         INPUT   MCS.Detach.User.request(maId);
292         TASK    reason := RN_user_requested;
293         NEXTSTATE detaching;

295   STATE busy;
296         SAVE    *;  /* defer MCS other */

298   STATE ready;
299         INPUT   MCS.Detach.User.request(maId);
300         TASK    pdu!reason := RN_user_requested,
301               pdu!userIds := Incl(user, Empty);
302         OUTPUT  DUrq(pdu) TO domain;
303         NEXTSTATE detached;

305   STATE ready;
306         INPUT   MCS.Channel.Join.request(maId, c);
307         TASK    pdu!initiator := user,
308               pdu!channelId := c;
```

```
309          OUTPUT  CJrq(pdu) TO domain;
310           2b :
311          TASK    dReady := Del(0, dReady);
312          NEXTSTATE busy;

314    STATE ready;
315          INPUT   MCS.Channel.Leave.request(maId, c);
316          TASK    cJoined := Del(c, cJoined),
317                 pdu!channelIds := Incl(c, Empty);
318          OUTPUT  CLrq(pdu) TO domain;
319          JOIN 2b;

321    STATE ready;
322          INPUT   MCS.Channel.Convene.request(maId);
323          TASK    pdu!initiator := user;
324          OUTPUT  CCrq(pdu) TO domain;
325          JOIN 2b;

327    STATE ready;
328          INPUT   MCS.Channel.Disband.request(maId, c);
329          DECISION c in cConvened;
330          (True): TASK    cAdmitted := Del(c, cAdmitted),
331                         cJoined := Del(c, cJoined);
332          ELSE:ENDDECISION;
333          TASK    cConvened := Del(c, cConvened),
334                 pdu!initiator := user,
335                 pdu!channelId := c;
336          OUTPUT  CDrq(pdu) TO domain;
337          JOIN 2b;

339    STATE ready;
340          INPUT   MCS.Channel.Admit.request(maId, c, uSet);
341          TASK    kind := CArq;
342          JOIN 3f;

344    STATE ready;
345          INPUT   MCS.Channel.Expel.request(maId, c, uSet);
346          TASK    kind := CErq;
347           3f :
348          TASK    mcsreq(0)!kind := kind,
349                 mcsreq(0)!channelId := c,
350                 mcsreq(0)!userIds := uSet,
351                 uPending := Incl(0, uPending);
352          CALL    Segment_request(0);
353          DECISION 0 in dReady;
354          (False):NEXTSTATE busy;
355          (True): NEXTSTATE ready;
356          ENDDECISION;

358    STATE ready;
359          INPUT   MCS.Send.Data.request(maId, c, dp, segmentation, userData);
360          TASK    kind := SDrq;
361          JOIN 4f;

363    STATE ready;
364          INPUT   MCS.Uniform.Send.Data.request(maId, c, dp, segmentation, userData);
365          TASK    kind := USrq;
366           4f :
367          DECISION dp >= 4 or dp in uPending;
368          (True): OUTPUT  MCS.Detach.User.indication(maId, user,
RN_provider_initiated);
369                 TASK    pdu!reason := RN_provider_initiated,
370                        pdu!userIds := Incl(user, Empty);
371                 OUTPUT  DUrq(pdu) TO domain;
372                 NEXTSTATE detached;
373          (False):TASK    mcsreq(dp)!kind := kind,
374                         mcsreq(dp)!channelId := c,
375                         mcsreq(dp)!segmentation := segmentation,
376                         mcsreq(dp)!userData := userData,
377                         mcsreq(dp)!offset := 0,
378                         uPending := Incl(dp, uPending),
379                         dp := IF dp < parameters!numPriorities THEN dp
380                                        ELSE parameters!numPriorities - 1 FI;
381                 CALL    Segment_request(dp);
382                 DECISION 0 in dReady;
383                 (False):NEXTSTATE busy;
384                 (True): NEXTSTATE ready;
385                 ENDDECISION;
386          ENDDECISION;
```

```
388   STATE ready;
389         INPUT   MCS.Token.Grab.request(maId, t);
390         TASK    pdu!initiator := user,
391                 pdu!tokenId := t;
392         OUTPUT  TGrq(pdu) TO domain;
393         JOIN 2b;

395   STATE ready;
396         INPUT   MCS.Token.Inhibit.request(maId, t);
397         TASK    pdu!initiator := user,
398                 pdu!tokenId := t;
399         OUTPUT  TIrq(pdu) TO domain;
400         JOIN 2b;

402   STATE ready;
403         INPUT   MCS.Token.Give.request(maId, t, u);
404         DECISION u >= 1001;
405         (False):OUTPUT  MCS.Token.Give.confirm(maId, t, RT_no_such_user);
406                 NEXTSTATE -;
407         (True): TASK    pdu!initiator := user,
408                         pdu!tokenId := t,
409                         pdu!recipient := u;
410                 OUTPUT  TVrq(pdu) TO domain;
411                 JOIN 2b;
412         ENDDECISION;

414   STATE ready;
415         INPUT   MCS.Token.Give.response(maId, t, result);
416         DECISION result = RT_successful;
417         (False):TASK    result := RT_user_rejected;
418         ELSE:ENDDECISION;
419         DECISION t in tRecipient and result = RT_successful;
420         (True): TASK    tPossessed := Incl(t, tPossessed);
421         ELSE:ENDDECISION;
422         TASK    tRecipient := Del(t, tRecipient),
423                 pdu!result := result,
424                 pdu!recipient := user,
425                 pdu!tokenId := t;
426         OUTPUT  TVrs(pdu) TO domain;
427         JOIN 2b;

429   STATE ready;
430         INPUT   MCS.Token.Please.request(maId, t);
431         TASK    pdu!initiator := user,
432                 pdu!tokenId := t;
433         OUTPUT  TPrq(pdu) TO domain;
434         JOIN 2b;

436   STATE ready;
437         INPUT   MCS.Token.Release.request(maId, t);
438         TASK    tPossessed := Del(t, tPossessed),
439                 pdu!initiator := user,
440                 pdu!tokenId := t;
441         OUTPUT  TRrq(pdu) TO domain;
442         JOIN 2b;

444   STATE ready;
445         INPUT   MCS.Token.Test.request(maId, t);
446         TASK    pdu!initiator := user,
447                 pdu!tokenId := t;
448         OUTPUT  TTrq(pdu) TO domain;
449         JOIN 2b;

451   STATE busy, ready;
452         INPUT   PDU.ready(dp);
453         TASK    dReady := Incl(dp, dReady);
454         CALL    Segment_request(dp);
455         DECISION 0 in dReady;
456         (False):NEXTSTATE busy;
457         (True): NEXTSTATE ready;
458         ENDDECISION;

460   STATE busy, ready;
461         INPUT   PCin(pdu);
462         TASK    reason := RN_channel_purged;
463         DECISION user in pdu!detachUserIds;
464         (True): OUTPUT  MCS.Detach.User.indication(maId, user, reason);
465                 OUTPUT  Quit TO control;
```

```
466                 NEXTSTATE detached;
467           (False):TASK    uSet := pdu!detachUserIds;
468                  5b :   /* for u in uSet */
469                 DECISION uSet = Empty;
470                 (False):TASK    u := Pick(uSet),
471                                 uSet := Del(u, uSet);
472                       OUTPUT  MCS.Detach.User.indication(maId, u, reason);
473                       JOIN 5b;
474                 ELSE:ENDDECISION;
475                 TASK    cSet := pdu!purgeChannelIds;
476                  6b :   /* for c in cSet */
477                 DECISION cSet = Empty;
478                 (False):TASK    c := Pick(cSet),
479                                 cSet := Del(c, cSet);
480                       DECISION c in cConvened;
481                       (True): TASK    cConvened := Del(c, cConvened),
482                                       cAdmitted := Del(c, cAdmitted),
483                                       cJoined := Del(c, cJoined);
484                             OUTPUT  MCS.Channel.Disband.indication(maId, c, reason);
485                       ELSE:ENDDECISION;
486                       DECISION c in cAdmitted;
487                       (True): TASK    cAdmitted := Del(c, cAdmitted),
488                                       cJoined := Del(c, cJoined);
489                             OUTPUT  MCS.Channel.Expel.indication(maId, c, reason);
490                       ELSE:ENDDECISION;
491                       DECISION c in cJoined;
492                       (True): TASK    cJoined := Del(c, cJoined);
493                             OUTPUT  MCS.Channel.Leave.indication(maId, c, reason);
494                       ELSE:ENDDECISION;
495                       JOIN 6b;
496                 ELSE:ENDDECISION;
497                 OUTPUT  PDU.ready(0) TO domain;
498                 NEXTSTATE -;
499           ENDDECISION;

501     STATE busy;
502           INPUT   PTin(pdu);
503           DECISION (pdu!purgeTokenIds and (tPossessed or tRecipient)) = Empty;
504           (False):OUTPUT  MCS.Detach.User.indication(maId, user, RN_token_purged);
505                 TASK    reason := RN_token_purged;
506                 NEXTSTATE detaching;
507           (True): OUTPUT  PDU.ready(0) TO domain;
508                 NEXTSTATE -;
509           ENDDECISION;

511     STATE ready;
512           INPUT   PTin(pdu);
513           DECISION (pdu!purgeTokenIds and (tPossessed or tRecipient)) = Empty;
514           (False):OUTPUT  MCS.Detach.User.indication(maId, user, RN_token_purged);
515                 TASK    pdu!reason := RN_token_purged,
516                         pdu!userIds := Incl(user, Empty);
517                 OUTPUT  DUrq(pdu) TO domain;
518                 NEXTSTATE detached;
519           (True): OUTPUT  PDU.ready(0) TO domain;
520                 NEXTSTATE -;
521           ENDDECISION;

523     STATE busy, ready;
524           INPUT   AUcf(pdu);
525           OUTPUT  MCS.Detach.User.indication(maId, user, RN_unspecified);
526           OUTPUT  Quit TO control;
527           NEXTSTATE detached;

529     STATE busy, ready;
530           INPUT   DUin(pdu);
531           DECISION user in pdu!userIds;
532           (True): OUTPUT  MCS.Detach.User.indication(maId, user, pdu!reason);
533                 OUTPUT  Quit TO control;
534                 NEXTSTATE detached;
535           (False):TASK    uSet := pdu!userIds;
536                  7b :   /* for u in uSet */
537                 DECISION uSet = Empty;
538                 (False):TASK    u := Pick(uSet),
539                                 uSet := Del(u, uSet);
540                       OUTPUT  MCS.Detach.User.indication(maId, u, pdu!reason);
541                       JOIN 7b;
542                 ELSE:ENDDECISION;
543                 OUTPUT  PDU.ready(0) TO domain;
544                 NEXTSTATE -;
```

```
545         ENDDECISION;

547   STATE busy, ready;
548         INPUT   CJcf(pdu);
549         TASK    c := pdu!channelId;
550         DECISION pdu!result = RT_successful;
551         (True): TASK    cJoined := Incl(c, cJoined);
552         ELSE:ENDDECISION;
553         OUTPUT  MCS.Channel.Join.confirm(maId, pdu!requested, pdu!result, c);
554          8b :
555         OUTPUT  PDU.ready(0) TO domain;
556         NEXTSTATE -;

558   STATE busy, ready;
559         INPUT   CCcf(pdu);
560         TASK    c := pdu!channelId;
561         DECISION pdu!result = RT_successful;
562         (True): TASK    cConvened := Incl(c, cConvened),
563                         cAdmitted := Incl(c, cAdmitted);
564         ELSE:ENDDECISION;
565         OUTPUT  MCS.Channel.Convene.confirm(maId, pdu!result, c);
566         JOIN 8b;

568   STATE busy, ready;
569         INPUT   CDin(pdu);
570         TASK    c := pdu!channelId,
571                 reason := RN_channel_disbanded;
572         DECISION c in cConvened;
573         (True): TASK    cConvened := Del(c, cConvened),
574                         cAdmitted := Del(c, cAdmitted),
575                         cJoined := Del(c, cJoined);
576                 OUTPUT  MCS.Channel.Disband.indication(maId, c, reason);
577         ELSE:ENDDECISION;
578         DECISION c in cAdmitted;
579         (True): TASK    cAdmitted := Del(c, cAdmitted),
580                         cJoined := Del(c, cJoined);
581                 OUTPUT  MCS.Channel.Expel.indication(maId, c, reason);
582         ELSE:ENDDECISION;
583         JOIN 8b;

585   STATE busy, ready;
586         INPUT   CAin(pdu);
587         TASK    c := pdu!channelId,
588                 cAdmitted := Incl(c, cAdmitted);
589         OUTPUT  MCS.Channel.Admit.indication(maId, c, pdu!initiator);
590         JOIN 8b;

592   STATE busy, ready;
593         INPUT   CEin(pdu);
594         TASK    c := pdu!channelId,
595                 reason := RN_user_requested;
596         DECISION c in cAdmitted;
597         (True): TASK    cAdmitted := Del(c, cAdmitted),
598                         cJoined := Del(c, cJoined);
599                 OUTPUT  MCS.Channel.Expel.indication(maId, c, reason);
600         ELSE:ENDDECISION;
601         JOIN 8b;

603   STATE busy, ready;
604         INPUT   SDin(pdu);
605         TASK    pdu!kind := SDin;
606         JOIN 9f;

608   STATE busy, ready;
609         INPUT   USin(pdu);
610         TASK    pdu!kind := USin;
611          9f :
612         TASK    c := pdu!channelId,
613                 dp := pdu!dataPriority;
614         DECISION c in cJoined;
615         (True): TASK    mcspdu(dp) := pdu,
616                         dPending := Incl(dp, dPending);
617                 CALL    Indicate_data;
618         (False):TASK    dp := IF dp < parameters!numPriorities THEN dp
619                                 ELSE parameters!numPriorities - 1 FI;
620                 OUTPUT  PDU.ready(dp) TO domain;
621         ENDDECISION;
622         NEXTSTATE -;
```

```
624   STATE busy, ready;
625         INPUT   TGcf(pdu);
626         CALL    Track_token(pdu!tokenId, pdu!tokenStatus);
627         OUTPUT  MCS.Token.Grab.confirm(maId, pdu!tokenId, pdu!result);
628         JOIN 8b;

630   STATE busy, ready;
631         INPUT   TIcf(pdu);
632         CALL    Track_token(pdu!tokenId, pdu!tokenStatus);
633         OUTPUT  MCS.Token.Inhibit.confirm(maId, pdu!tokenId, pdu!result);
634         JOIN 8b;

636   STATE busy, ready;
637         INPUT   TVin(pdu);
638         TASK    tRecipient := Incl(pdu!tokenId, tRecipient);
639         OUTPUT  MCS.Token.Give.indication(maId, pdu!tokenId, pdu!initiator);
640         JOIN 8b;

642   STATE busy, ready;
643         INPUT   TVcf(pdu);
644         CALL    Track_token(pdu!tokenId, pdu!tokenStatus);
645         OUTPUT  MCS.Token.Give.confirm(maId, pdu!tokenId, pdu!result);
646         JOIN 8b;

648   STATE busy, ready;
649         INPUT   TPin(pdu);
650         TASK    t := pdu!tokenId;
651         DECISION t in tPossessed or t in tRecipient;
652         (True): OUTPUT  MCS.Token.Please.indication(maId, t, pdu!initiator);
653         ELSE:ENDDECISION;
654         JOIN 8b;

656   STATE busy, ready;
657         INPUT   TRcf(pdu);
658         CALL    Track_token(pdu!tokenId, pdu!tokenStatus);
659         OUTPUT  MCS.Token.Release.confirm(maId, pdu!tokenId, pdu!result);
660         JOIN 8b;

662   STATE busy, ready;
663         INPUT   TTcf(pdu);
664         OUTPUT  MCS.Token.Test.confirm(maId, pdu!tokenId, pdu!tokenStatus);
665         JOIN 8b;

667   STATE detaching, detached;
668         INPUT   *;
669         NEXTSTATE -;

671   STATE detaching, detached;
672         INPUT   Quit;
673         OUTPUT  Quit TO control;
674         NEXTSTATE detached;

676   STATE detaching;
677         INPUT   PDU.ready(dp);
678         DECISION dp = 0;
679         (False):NEXTSTATE -;
680         (True): TASK    pdu!reason := reason,
681                         pdu!userIds := Incl(user, Empty);
682                 OUTPUT  DUrq(pdu) TO domain;
683                 NEXTSTATE detached;
684         ENDDECISION;

686   STATE detaching, detached;
687         INPUT   PCin(pdu);
688         DECISION user in pdu!detachUserIds;
689         (False):NEXTSTATE -;
690         (True): OUTPUT  Quit TO control;
691                 NEXTSTATE detached;
692         ENDDECISION;

694   STATE detaching, detached;
695         INPUT   DUin(pdu);
696         DECISION user in pdu!userIds;
697         (False):NEXTSTATE -;
698         (True): OUTPUT  Quit TO control;
699                 NEXTSTATE detached;
700         ENDDECISION;

702   ENDPROCESS;
```

付録７　付録の特徴

（ＪＴ－Ｔ１２５に対する）

１．SDL図

付図2-1/JT-T125 は、システム内のMCSプロバイダを図示している。１つの制御アプリケーションへの Control MCSAP、０または１以上のアタッチしているユーザへのMCSAP、０または１以上のトランスポート サービス プロバイダへのTSAPの３つのチャネルがMCSプロバイダとインタフェースをとっている。これらの外部チャネルは、MCSプロバイダの各プロセスと信号経路を形成している。

各々の８角形の要素は、プロセスのタイプを示しており、括弧内の数字は（最小値，最大値）を示している。１つのコントロール プロセスは、永久に存在する。破線は、それが３つのプロセスを生成していることを示している。アタッチメント、ドメイン、終端は、初期状態では存在していないが、それらの生成数は無制限である。

各プロセス間の信号経路を用いてSDL信号が運ばれる。与えられた方向へ転送された信号のセットは、矢印の先頭付近に示されている。括弧内には、関連した信号のリストが示されている。これらのリストの拡張は図中に入れるには長すぎる。詳細は、付録２のテキスト部に記述されている。

２．サービス定義

外部チャネル（Control MCSAP、MCSAP、TSAP）の信号は、TTC標準ＪＴ－Ｔ１２２とCCITT勧告Ｘ.２１４に抽象的に定義されたMCSサービスとトランスポートサービスを使用している。SDLでのこれらのモデル化は、さらに、相互作用の詳細についての仮定を必要とする。これに関し、２つの特記事項がある。すなわち、サービスプリミティブを特定するための終端識別子の使用と、データの転送を要求するフロー制御レディ信号の使用である。

MCSコネクションID、MCSアタッチメントID、TC終端IDの３つの識別子があり、それらはサービス定義に明示されていないが暗示的に記述されている。このモデルでは、それらの識別子は責任のあるプロバイダにより割り当てられることを仮定している。例えば、TC 終端 ID は T-CONNECT 指示と T-CONNECT 確認に含まれている。サービスを利用するユーザが要求プリミティブに対する確認プリミティブを確実に識別できるように、ユーザはエコーバックされる任意のラベルを指定することができる。

フロー制御は、１つのウィンドウを用いた方法でモデル化されている。レディ信号は、相手側がデータを送信する前に相手側へ送信されなければならない。次のデータは、もう１つのレディ信号を受信した後に送信されなければならない。実装上、レディ信号を受信すると、ポインタを移動したりカウンタに１を加算したりするかもしれない。レディ信号を送信することは、データ転送ほど重要ではない。レディ信号には、T.ready、PDU.ready、MCS.readyの３つがある。これらは、それぞれ１つのTSDU、１つのDomain MCSPDU、そして１つのMCSサービス データ ユニットの転送を許可する。フロー制御は、信号経路の２つの方向で、独立に行われる。

TC上のトランスポートサービス品質は、スループット、伝送遅延、データ プライオリティのパラメータでモデル化されている。これは、相互に一貫性のない標準の混合物である。実際には、恐らく詳細は異なるであろう。QOSの詳述は、制御アプリケーションにとって避けることのできない義務である。

3．ドメインへの入口

ドメイン プロセスはMCSプロバイダの分解図の中央部である。他のものは同じプロバイダ内の別のドメインを表す。ドメイン プロセスは固定値のパラメータセットを伴って生成される。これらのドメインパラメータはコントロール プロセスによって提供される。コントロール プロセスはドメイン セレクタとドメイン プロセスの構成を決定する。MCS-CONNECT-PROVIDERの間にパラメータ交渉の余地があるなら、コントロール プロセスは相互のやりとりを管理する。

ドメイン プロセスを可能な限り単純にするために、インタフェースで連結されているMCSアタッチメントとMCS コネクションは幾つかの共通の特徴を伴った入口として表れるように抽象化される。入口は1つのアタッチメント プロセスか、MCS コネクションの各TCに1つずつある終端プロセスのセットのいずれかである。コントロール プロセスは入口に ID を割り当てて、ドメイン プロセスに対して、それらの結合の開閉を制御する。それは構築と分解のプロセスの詳細と Connect MCSPDU の交換を隠す。

ドメイン プロセスにとっては、1つの入口は実装されている各データ プライオリティの信号経路のセットの様に見える。おのおのは MCSPDU の実体を PDU.ready フロー制御部に伝達する。そこでは必ず処理の区別が必要であり、入口を3種類（アタッチ、上位へのリンク、下位へのリンク）の内の1つとして分類することによって呼び出される。

付図7－1／JT－T125から付図7－8／JT－T125までは、典型的なシーケンスフローの例である。それらは1つの MCS コネクションを持つドメインにローカルにアタッチする1ユーザを仮定している。終端1と終端2は2つのデータ プライオリティがドメインに実装されているという仮定から生じている。

入口を閉じるための手順はやや長い。これは、いずれかのプロセスが他の停止したプロセスに信号を送り、SDL 図でランタイムエラーが発生するという可能性を防止する。アタッチメントと終端プロセスは Quit を最後の送信信号にして、Exit の受信で停止する。ドメインはコントロールにShut.portalを送った後は入口へ信号を送らず、最後の入口を閉じた後に停止する。

他の信号では、Drop.portalは両方向あり、DPumに対応する。Report.portalは診断が別の MCS プロバイダに向かって出されたことをコントロールに警告する。終端で生成される RJum はドメインを経由して初期 TC に送られる。受け取られる RJum は終端からコントロールまで直接急送される。

MCS-ATTATCH-USER-要求と T-CONNECT-指示は、対応する入口を生成する刺激であるためにコントロールに向けられることに注意すること。コントロールは失敗のMCS-ATTATCH-USER-確認、またはT-DISCONNECT-要求で弱まる。

現在のモデルは、プロバイダ内のすべてのドメインを通してのローカルな入口の数をのぞいては、ドメインによるMCS アタッチメントやMCSコネクションの数の制限が無い。ドメインの形態の多くの面は、さらなる標準化が着手されるまで、ローカルに管理されなければならない。

4．MCSPDUの配列

第7章で定義されたMCSPDUは個々に SDL によって表されることで明瞭になる。しかし、ドメインプロセスの簡素性はさらに統一された処理に依存する。ここでの解決策は、どの Domain MCSPDU にも適合する様に選択できる構成要素を持つ PDUStruct データ型を定義することである。ドメイン プロセ

スの外で処理される Connect MCSPDUは、独自の構造を維持する。

PDUStruct の重要な構成要素は、目的の Domain MCSPDU を識別するkind である。kindに基づいてその時々の他のフィールドの構成は変化する。(それらのフィールドの構成は MCSPDU の構成要素としてASN.1記法で述べられている。) データ型を定義したSDLは、第7章のデータ型を定義したASN.1での記述にほぼ一致する。この意味は明らかであり、PDUStructはDomain MCSPDUの解読された表現である内部構造である。

AUcf、CJcf、CCcfの3つの MCSPDU はオプションの構成要素を持つ。SDLのなかで、それらのオプションの未使用はフィールドに0を設定することで示す。これは静的あるいは動的チャネル ID では使わない値である。

アタッチメントとドメインと終端のプロセスの間を送られる PDU.ready 以外の信号は、パラメータとして PDUStruct を持つ。実際の実装においては、この種の通信はバッファポインタを動かす程度かもしれない。

5. SDLの使用法

参照実装には、抽象データ型を定義する能力によってパワーが強化されるプログラミング言語として、SDLのテキスト表現を使う。コントロールとドメインでは、管理するオブジェクトが複雑で多すぎるために、SDLがサポートするある程度の有限のステートマシンから利益を得ることができない。これらのプロセスは入力信号を処理する1つの状態にとどまる。より狭い範囲であるアタッチメントと終端は、さらに有利にするために、少しの状態を使う。

付録2は全プロバイダで使われる SetOf ジェネレータの定義を含んでいる。これは、チャネル ID のような他の定義された型をとってその型の値のサブセットの概念を整える。SetOfは、空でないサブセットから随意の要素を選択する新しいオペレータを組み込むことによって、組み込み式のジェネレータ Powerset を拡張する。この Pick オペレータの意味は簡単な原理によって定義される。

セットは実装の至る所に使われる。例えば、情報ベースの一部はチャネル ID のどのサブセットが使われるいるかや、入口のIDのどのサブセットが特定のチャネルに加入しているかを記録する必要がある。

SDL では、配列は検索データ型の全範囲で使用される。チャネル構造は、たとえば、0から65535までの各チャネル ID に対して存在する。実際の実装ではもっとまばらな配列を扱わなければならない。この最後に、個別の ID セットはチャネルと他の資源がある時間に使用中であるかを記録するために保持されている。設計の原理は、明白に使用中と示されていない ID の情報ベースに配列値を保持する必要がないことである。

多くの反復はよく知られたパターンを示す。対象の候補のセットを作り、そのあとなくなるまで、1つずつその中のメンバを選択し削除する。そのような反復は決定と結合から成る。コメントは意図されている更に高水準の制御構造を示す。ラベルはプロシージャで順次番号付けされる。ラベルには、分岐が前方へか後方へかを示す文字がついている。

呼び出し手順を示す内容の表は各プロセスにおける最初の手続き定義の前に現れる。SDL の手続きは IN/OUT に当たるように宣言された正式のパラメータによって結果を戻すかもしれない。

データ プライオリティの最大数(4)や静的チャネルID と動的チャネル ID の間の分割点(1001)のように、2、3の定数はコードを通して至る所に現れる。これらは値を再定義できないパラメータであ

る。

ユーザIDがチャネルIDのサブセットであることは MCS の中では重要な概念である。しかし、おのおのが使う内容は異なり、２つの個別の型があてられる。付録２ではそれらの間を埋めるために１組のオペレータを設定している。

付録３から付録６までの手順は付録２によって明確に述べられた構成の中で協調する。信号は明確に述べられた経路を強制され、内部を横断する不作法は許されない。守勢のコーディングはプロバイダの外の環境にあるコントローラ及びユーザアプリケーションと、同位のプロバイダから受信する MCSPDU への注意に焦点を絞る。

参照実装は、１つ以上が未決定であるなら入力信号が随意の順序になることを許す。それは、プロセスの相対的なプライオリティの仮定を行わない。しかし、先制権無しの達成を実行する個々の入力移行に頼る。

６．ドメイン プロセスの注意

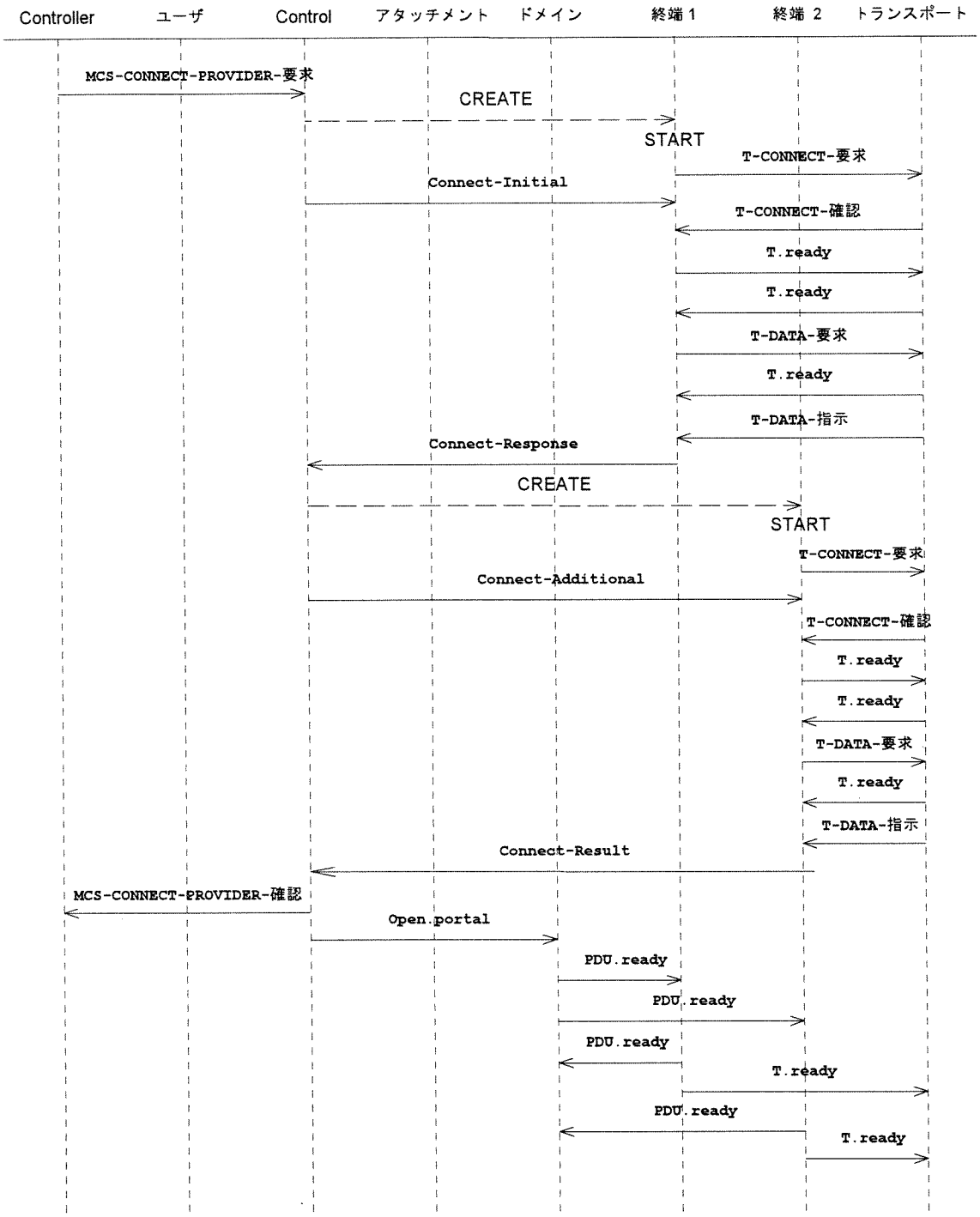
SetOfの他、Queueジェネレータも重要な役割を果たす。バッファは出力の入口にユーザの伝達する順序でキューイングされ、入口のIDは未応答のMCrq、MTrq、AUrq の MCSPDU の順序でキューイングされる。

ユーザデータをコピーすることを回避するために MCSPDU はバッファにおいて処理され、複数のポートに出力されるかもしれない。ドメイン プロセスに利用可能なバッファの数は確定した外部パラメータとしてモデル化される。これは、ドメインごとのコンフィグレーションによって簡単に変更できる。提供されたバッファの数が少なければ、全体的なフロー制御の実装の質を落とす。バッファが解放されると、そのバッファは最も高いプライオリティーのデータフローに適用される。

MCSPDU が入力として届くとき、ドメイン プロセスは起点とデータ プライオリティの入口を知る必要がある。Identify_sender プロシージャはSDLによって知らされたプロセスIDに基づいて開いている入口を徹底的に調査する。必要とされる情報は、直感的な意味が不明確なものをのぞいて、信号のパラメータセットの一部として定義される。実際の実装では、このモデル化は重要な問題ではない。

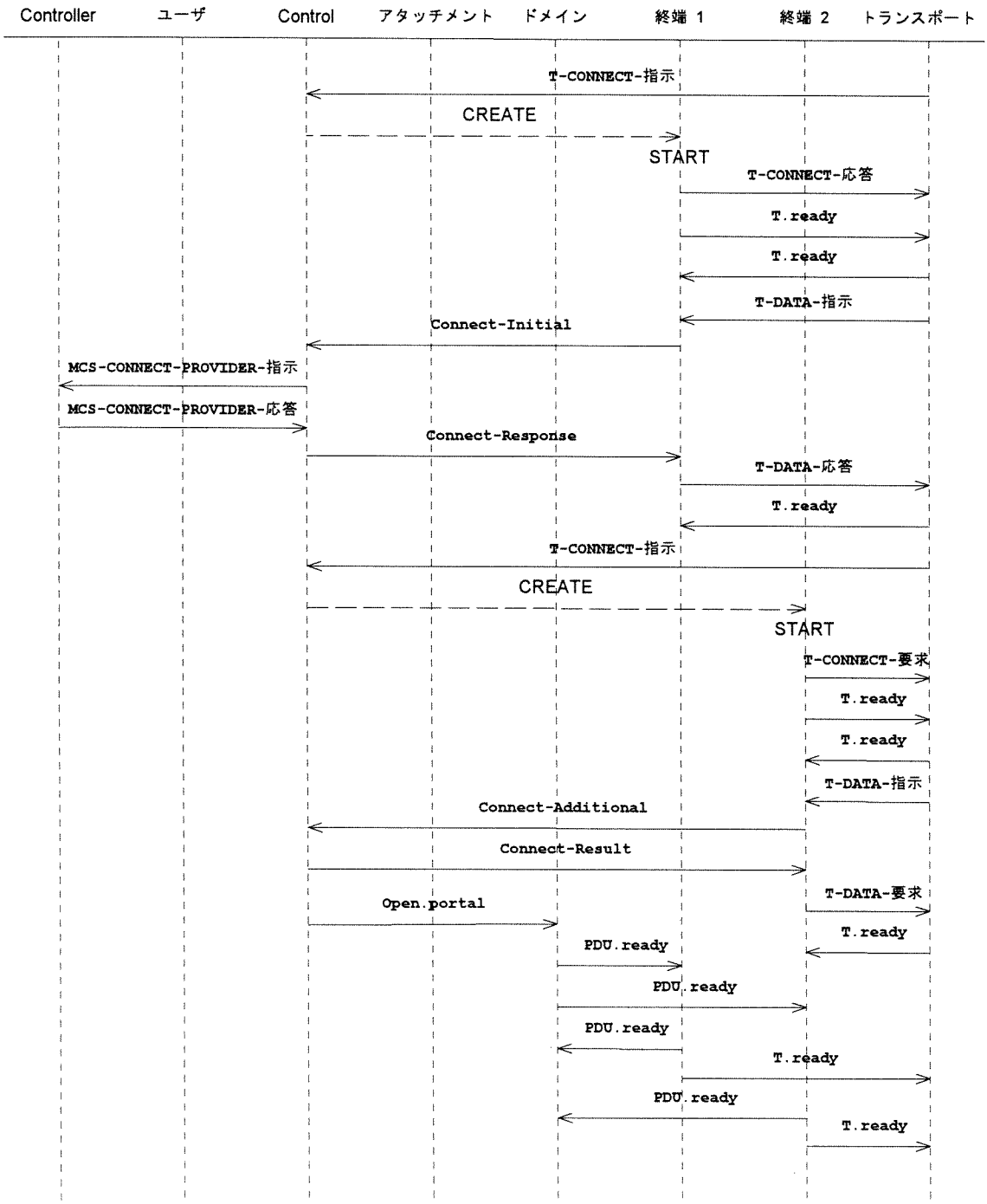
ドメインプロセスの中心は、Process_PDU プロシージャであり、これはValidate_input、Top_provider、Apply_PDUを順に呼び出している。これらは、それぞれ、PDUの種類に基づいた大きな場合分けである。どんな種類のものの詳細もたいてい簡単である。これらのプロシージャにアプローチする良い方法は、SDrq のような重要な MCSPDU のコースに従って、１つずつ水平に切ることかもしれない。

MCSPDU の符号化（プロトコルバージョンに応じた BERかPER を使う）が終端プロセスに残された詳細である。ドメインが知っていなければならないことは、複数のユーザ ID を持つDUrq の様な可変サイズのMCSPDU を、どれくらいの大きさまで作れるかだけである。この問題は、BER符号化方式のための無効定数を使う、略式文書での決定一覧表を通して表わされる。実際問題として、幾つかの可変の容量の限界はIDの最大数の様にドメインが理解する方式で計算されるかもしれない。その他は、符号化に関する直接的な知識を必要とすることがある。



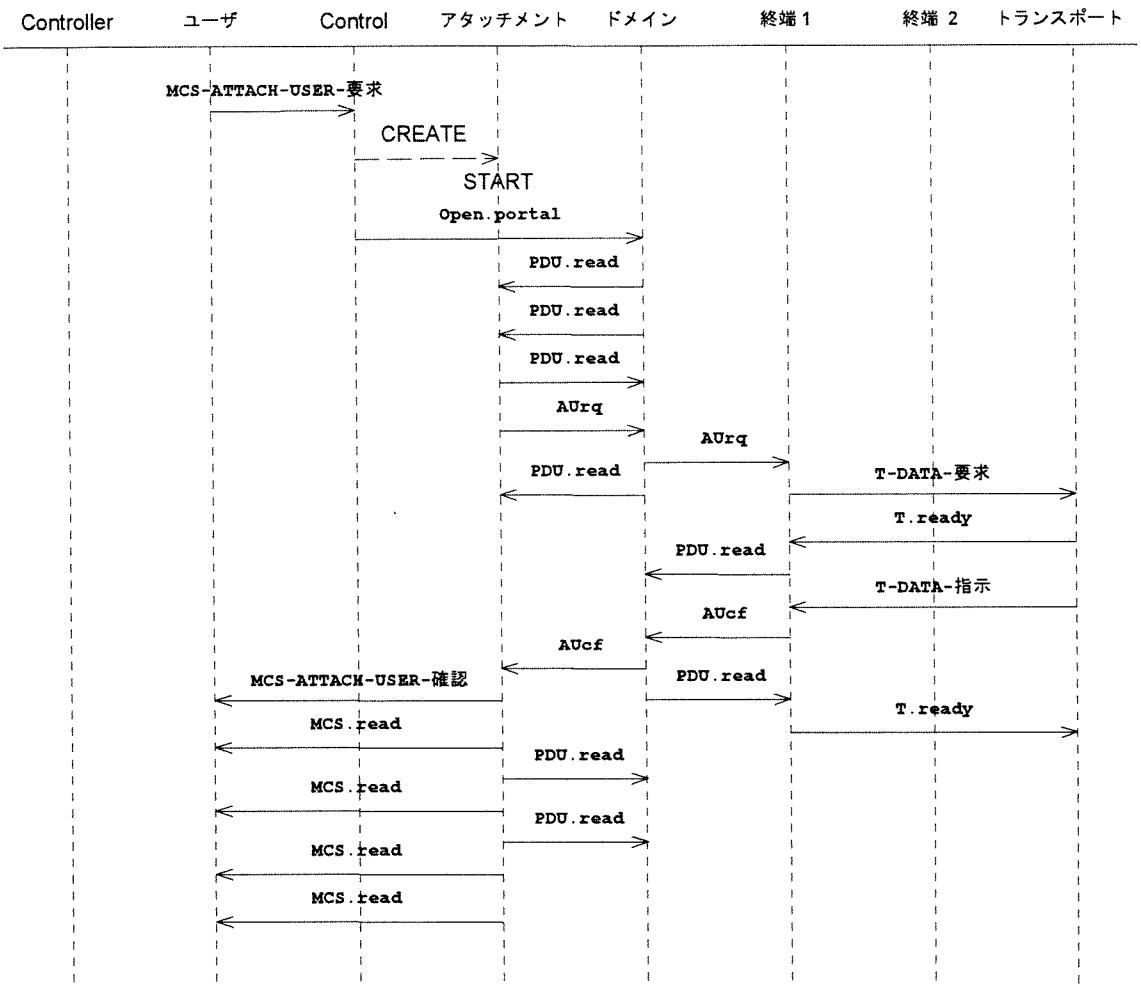
付図７－１／ＪＴ－Ｔ１２５　発側MCSプロバイダ

(ITU-T T.125)



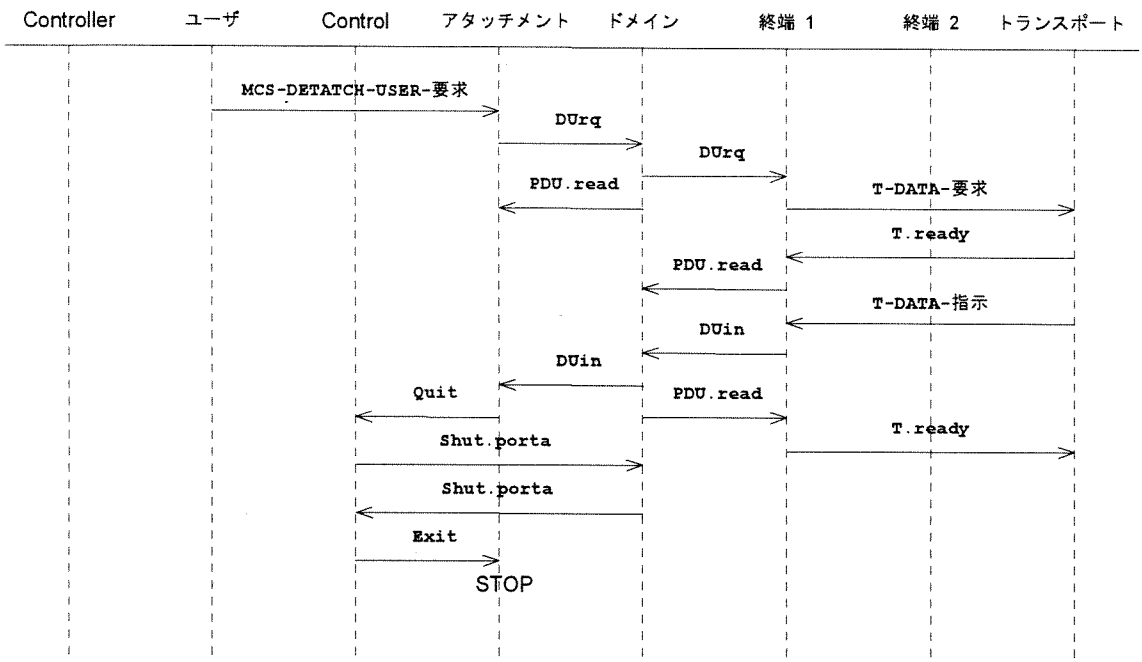
付図７－２／ＪＴ－Ｔ１２５　着側MCSプロバイダ

(ITU-T T.125)



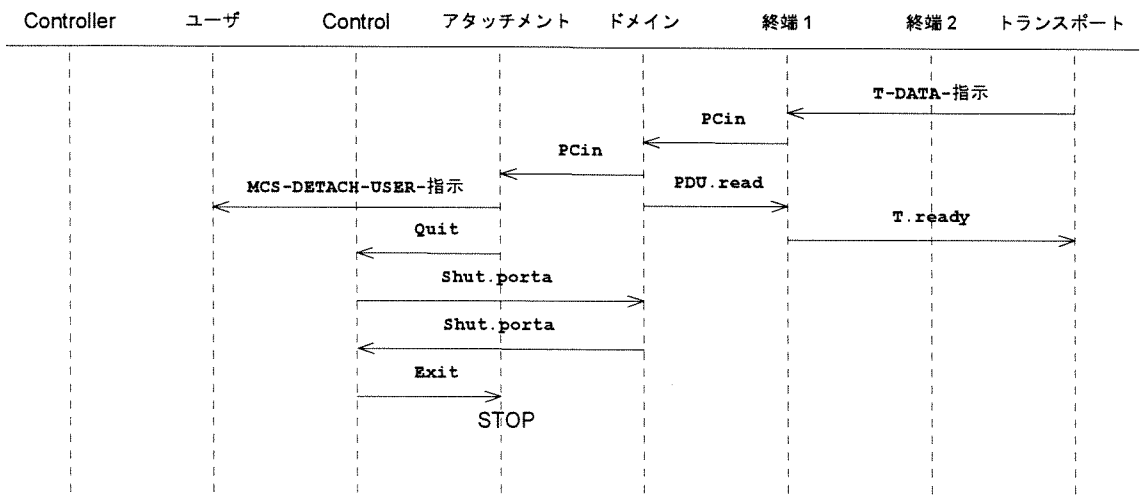
付図７－３／ＪＴ－Ｔ１２５　ユーザアタッチ

(ITU-T T.125)



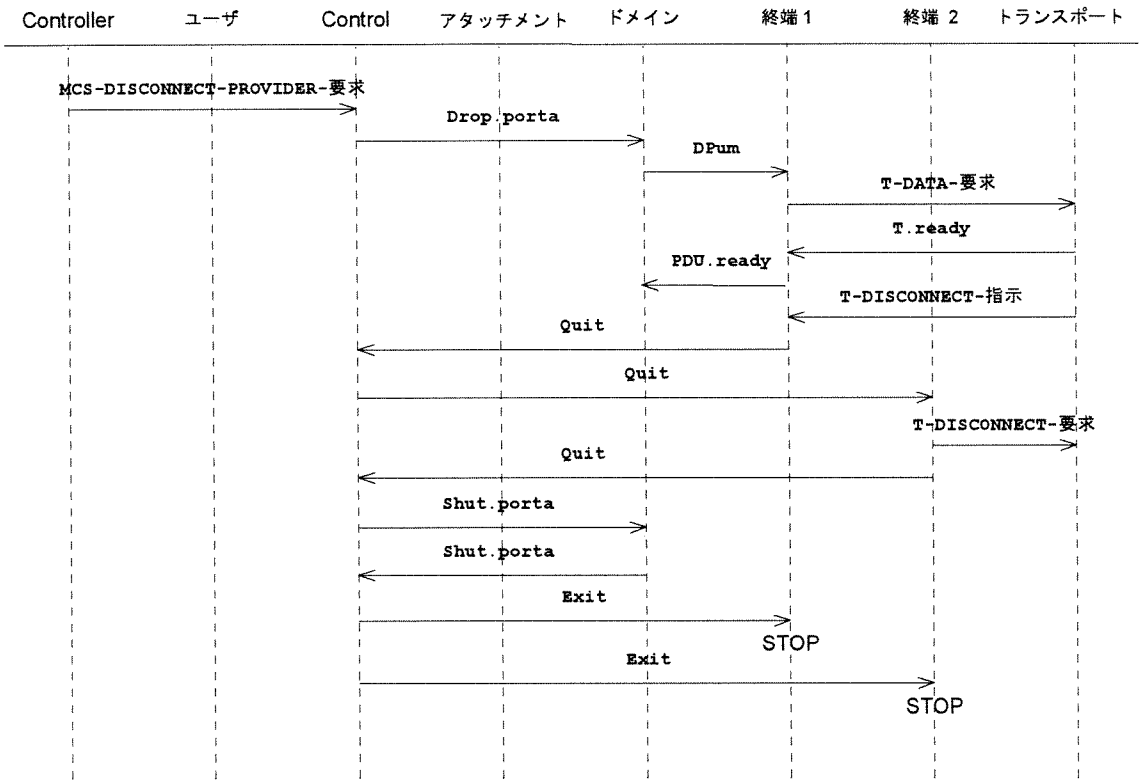
付図７－４／ＪＴ－Ｔ１２５　ユーザからのデタッチ

(ITU-T T.125)



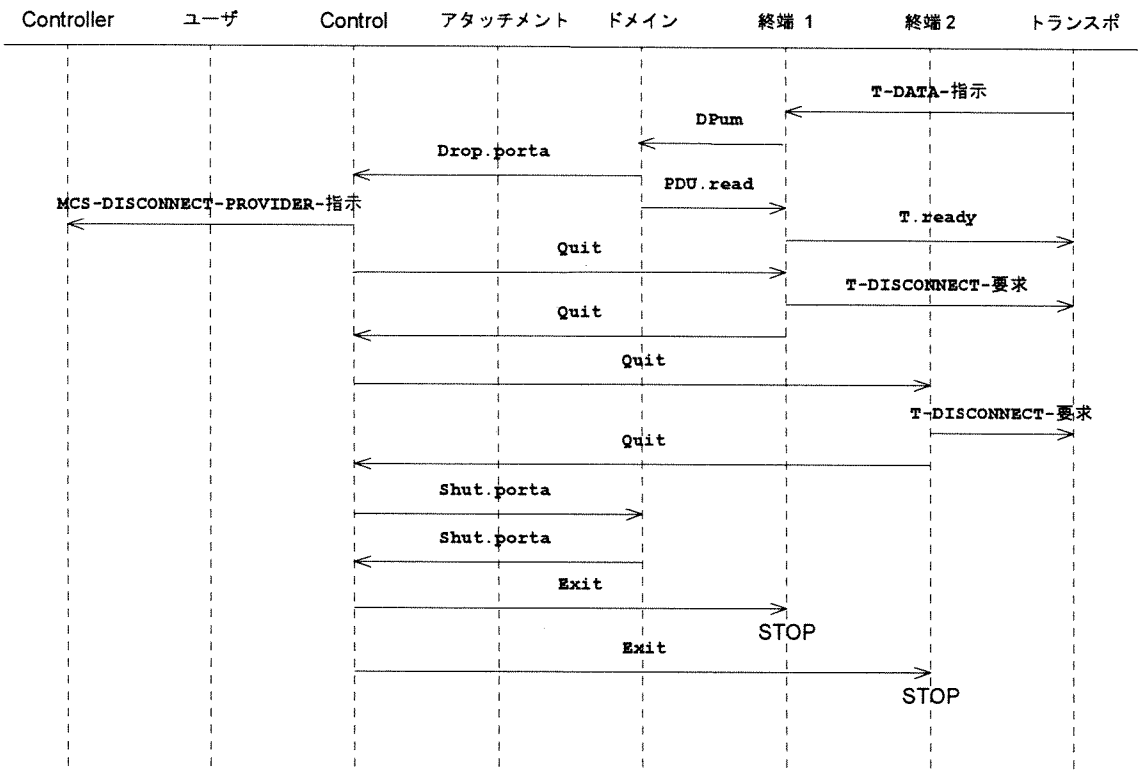
付図７－５／ＪＴ－Ｔ１２５　プロバイダからのデタッチ

(ITU-T T.125)



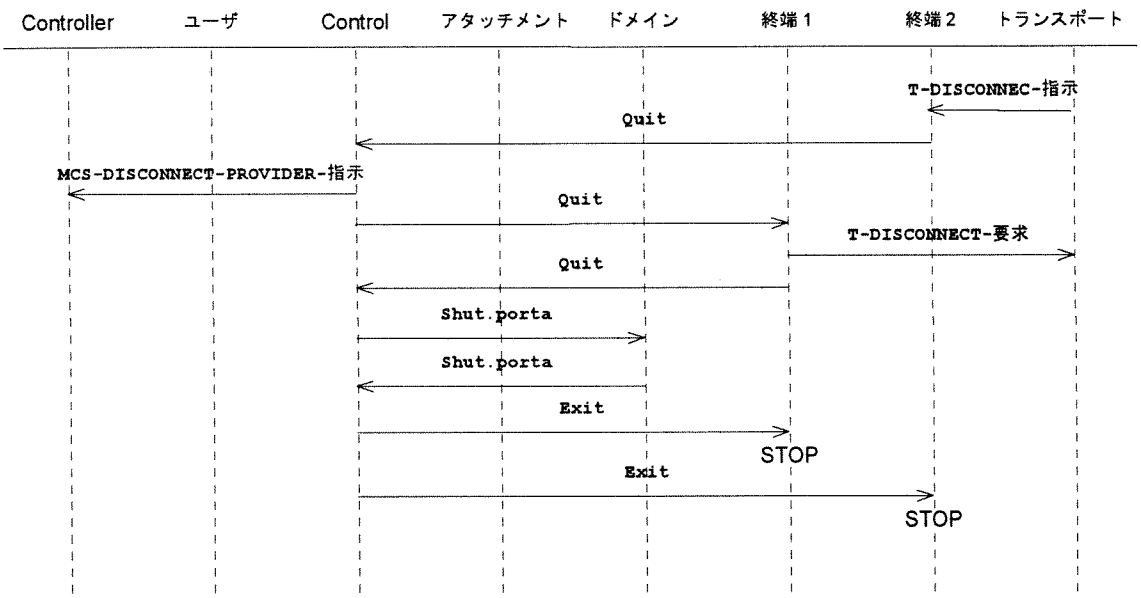
付図７－６／ＪＴ－Ｔ１２５　ローカルコントローラからの切断

((ITU-T T.125)



付図７－７／ＪＴ－Ｔ１２５　リモートコントローラからの切断

(ITU-T T.125)



付図７－８／ＪＴ－Ｔ１２５　プロバイダからの切断

(ITU-T T.125)

執筆作成協力者（１９９５年９月１日時点）

第四部門委員会

部門長	斎藤　幸男	日本電信電話（株）
副部門委員長	長尾　朋明	国際電電信電話（株）
副部門委員長	渡辺　芳明	日本アイ・ビー・エム（株）
委員	須川　毅	住友電気工業（株）
〃	堀　潔洋	（株）東芝
〃	山口　晋五	（株）リコー
〃	菅野　昌志	松下電器産業（株）
〃	羽柴　善安	東京電力（株）
〃	鷹司　尚武	（第１専門委員会　専門委員長）
〃	伊藤　均	（第１２専門委員会　副専門委員長）
〃	山田　満	（第２専門委員会　専門委員長）
〃	松本　充司	（第２２専門委員会　副専門委員長）
〃	藤野　博文	（第２２３専門委員会　副専門委員長）
〃	橋山　康敬	（第３専門委員会　副専門委員長）
〃	青千　昇一	（ｵﾌﾞｼﾞｪｸﾄｺｰﾄﾞ特別専門委員会　専門委員長）
〃	岩本　裕司	（ｵﾌﾞｼﾞｪｸﾄｺｰﾄﾞ特別専門委員会　副専門委員長）

第四部門委員会第２専門委員会

専門委員長	山田　満	国際電信電話（株）
副専門委員長	松本　充司	日本電信電話（株）
副専門委員	藤野　博文	沖電気工業（株）
委員	松本　浩知	国際電信電話（株）
〃	亀井　喜宏	日本電信電話（株）
〃	松原　幸俊	シャープ（株）
〃	礒山　芳一	住友電気工業（株）
〃	池田　強	（株）田村電機製作所
〃	井澤　庄次	（株）テック
〃	野笹　潤	（株）東芝
〃	佐藤　貴瑛	日本電気（株）
〃	鳥田　吉隆	日本無線（株）
〃	岩前　佳博	（株）日立製作所
〃	井木　卓薫	富士ゼロックス（株）
〃	下田　実	松下電送（株）
〃	柴田　博	三菱電機（株）
〃	沢下　眞廣	（株）明電舎
〃	吉　健一	（株）リコー
〃		村田機械（株）
〃		三菱電機（株）

第四部門委員会第２専門委員会ＳＷＧ６

リーダ	沢　健一	三菱電機（株）
サブリーダ	吉前　和俊	日本電気（株）
委員	野口　博幸	日本電信電話（株）
〃	本下　充司	日本電信電話（株）
〃	大上　俊裕	沖電気工業（株）
〃	野松　治仁男	シャープ（株）
〃	若松　久智	（株）東芝
〃	八田　明子	日本アイ・ビー・エム（株）
〃	清水　雅一	日本電気（株）
〃	右馬　伸幸	（株）日立製作所
〃	田　成雅	三菱電機（株）
〃	月岡　薫進	（株）明電舎
〃	藤　健二	（株）リコー
ｵﾌﾞｻﾞｰﾊﾞ	君山　政信	（株）東芝
〃	中村	日本電気（株）

事務局　近藤　康生（第４技術部）