マルチメディア通信と分散処理 63−11
グループウェア 5−11
（1994. 1. 28）

MSC と LOTOS による通信システム設計支援システム

田口 毅　　高橋 薫　　土岐田 義明

(株) 高度通信システム研究所

あらまし　本論文では，ITU-T 勧告 Z.120 に述べられている MSC の定義を損なうことなく仕様検証能力を向上させることと，MSC と LOTOS それぞれで記述された仕様間の橋渡しを実現することを目的として，MSC で記述された仕様の LOTOS 変換の方法を提案する．まず，MSC 仕様の示している対象システムの構造を LOTOS 記述にいかにマッピングするかを述べ，次に MSC 仕様の記述の変換の方法について示す．また，筆者らが開発中の通信システム設計支援システム ITECS の 1 部であり，本方法の実装を進めている ASSISts について紹介する．最後に，交換サービスの MSC 仕様を例に LOTOS 変換を行い，その評価と検証に対する適応性について述べる．

A Specification Support System for Communication Software Design based on MSCs and LOTOS

Takeshi Taguchi　Kaoru Takahashi　　and　Yoshiaki Tokita

AIC System lab.

abstract　This paper describes a translation method of MSCs to LOTOS, which aims to extend the power of verification without losing the semantics of MSC described in ITU-T recommendation Z.120, and aims to bridge MSC and LOTOS. First, we show how the system structure implied by an MSC specification corresponds to LOTOS description, and present a translation method of MSCs to LOTOS. Second, we introduce a specification support system named " ASSISts", which we are now implementing with the proposed method. Finally, we give an application example of the method using an ISDN service, and describe an evaluation of the method and adaptability to the verification.

1 はじめに

近年，ソフトウェアの仕様を形式的仕様記述言語を用いて定義することで自動プログラミングや仕様検証といった機械的支援を期待するとともに，仕様の曖昧性の除去・欠落の防止を望む傾向が強くなっている．これに伴い通信ソフトウェアにおいても ITU-T，ISO で形式的仕様記述言語が勧告されている [CCI 92a], [CCI 92b], [ISO]．これらの仕様記述言語は，ITU-T 勧告の MSC や SDL は交換ソフトウェアに，ISO 勧告の LOTOS はプロトコル設計などに主に用いられている．

MSC は，システム構成要素とそれらの環境の間で交換されるメッセージの系列を記述することで，システムの仕様を記述する仕様記述言語である．MSC の単純な仕様記述例を図 1 に示す．MSC において，縦線はインスタンス

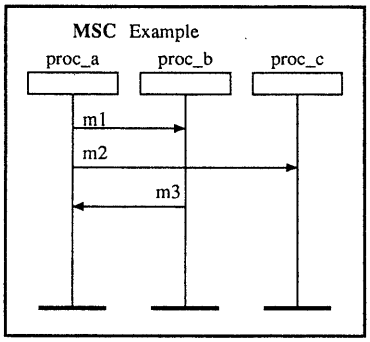

図 1: MSC の記述例

と呼ばれるシステムの構成要素を表し，横向きの矢印は 2 つのインスタンス間，あるいは，1 つのインスタンスと環境との間の情報交換を表す．MSC の長所は，その単純な言語仕様から言語の学習が容易であり，また，直観的な理解性が高い点にある．しかし，MSC 自体の持つ仕様検証能力が乏しいために，仕様検証の立場からは MSC 仕様のシミュレーションによる動作確認や，あるいは，言語仕様を独自に拡張し，MSC にプロセス代数によるフォーマルセマンティクスを与えるといった研究が行われてきた [JOZ 93].

一方 LOTOS は，外部的に観測可能な動作によってイベント間の時間的な順序関係を定義することによりシステムの仕様を記述する形式的仕様記述言語である．その特徴としては，インプリメンテーションの方法と完全に独立であるような高いレベルの抽象化を行える点などが知られているが，仕様検証の立場からは，デッドロックやライブロック・到達可能性を調べるといった Validation 機能に加えて，複数の仕様間の関係 (同値関係，前順序関係など) を調べる Verification 機能を有している点があげられる [ISO], [KAM 89],[KAO 90],[EIJ 89]．例えば，LOTOS で記述したサービス仕様に対して何らかの付加サービスの記述を追加した場合，追加した仕様が元の仕様を満足しているかどうかを調べるといった Verification が可能である [KAO 93].

そこで，MSC の持っている理解性を損なうことなく，かつ，勧告 Z.120 で述べられている言語仕様の範囲を越えることなく，仕様検証能力を向上させることと，MSC と LOTOS それぞれで記述された仕様間の橋渡しを実現することを目的として，MSC で記述された仕様の LOTOS 変換の方法を提案する．具体的には，まず，MSC で記述された仕様を LOTOS 変換するための変換モデルを示す．次に，この変換モデルに基づいて MSC で記述された仕様の動作定義を LOTOS 記述に対応させ，変換を可能にする方法の提案を行う．筆者らは現在，要求仕様記述に MSC を，それ以降の仕様の詳細化に LOTOS を用いた通信システム設計支援システム ITECS の開発を行っている [OHT 93]. 本論文で提案する変換方式は，その ITECS のアプリケーションの 1 つである，LOTOS への変換機能を有する MSC エディタ "ASSISts" の要素技術として開発を進めている．図 2 に，**ASSISts** の実行画面例を示す．

以下，本論文では，2 章で MSC と LOTOS の最大の相違点である通信方式の相違を吸収する方法を述べると共に，LOTOS 解釈のための変換モデルを示す．3 章では MSC の持つ様々な機能の LOTOS への変換法を示すと共に，複数の MSCs からなる MSC document を合成し，LOTOS へ変換する方法を提案する．そして 4 章で，ISDN の基本呼サービスの MSC による概要仕様を LOTOS に変換した例を示し，その考察を行う．5 章は，まとめである．

2 MSC-LOTOS 変換モデル

2.1 非同期通信と同期通信

ここではまず，MSC と LOTOS がそれぞれ基本としているモデルについて比較を行う．MSC 仕様に記述されるインスタンス間の通信では，メッセージの送信と受信は異なる事象であり，それぞれが非同期に送受信を行う [CCI 92b]. このための機構は特に MSC の言語仕様では明確に規定はされていないが，SDL との関係から，各インスタンスには容量が十分大きな FIFO (First In First Out) 型のメッセージ受信バッファを 1 つ持ち，他のインスタンスから受信する信号は全てこのバッファを経由する ([WAK 90]) と考えるのが妥当である．

一方 LOTOS は，システムの記述をプロセスの観点から行い，外部から観測される振舞としての通信能力を記述することでシステムを記述する．プロセスとは，その環境における他のプロセスと通信を行う抽象的な実体を示す．プロセス間の通信は，ゲートと呼ばれる作用点で起こり，その基本単位はアクションと呼ばれ，ゲートと送受される値の組からなる．このアクションは，「アトミック」で「同期的」な相互作用である．アクションがアトミックであるということは，

1. 瞬間的である
2. 細分化できない
3. 他のアクションと時間的な重なりがない

ということを意味している．また，同期的であるというのは，環境も含めて通信する 2 つ以上任意個のプロセスが，同時にその発生に関与するということを表す [KAO 90].

以上の様に，MSC と LOTOS をモデルレベルで比較すると，非同期通信と同期通信というシステム構成要素間の通信方式の違いが最大の相違点であることが分かる．一般に，同期通信モデルの仕様記述言語で非同期通信をエミュレートするには，非同期通信モデルの仕様記述言語内で想定されている FIFO バッファを何らかの形で明示的に記述することが考えられる．LOTOS では，FIFO バッファ処

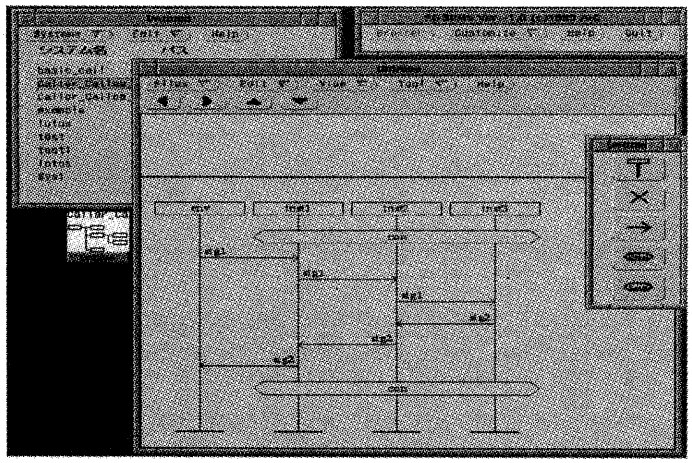

図 2: LOTOS への変換機能を有する MSC エディタ ASSISts

理は抽象データ型を用いたプロセスとして容易に記述できるので，MSC におけるメッセージ通信を，このバッファプロセスを介して行うように LOTOS 上に変換することにする．

2.2 MSC-LOTOS 変換モデル

本論文で提案する MSC - LOTOS 変換では，単に変換を行うだけではなく，変換結果をさらに詳細化し，LOTOS 上での設計作業に利用させることも目的としている [SAR 93]. このためには，変換結果の LOTOS 仕様が高い可読性を持ち，仕様の追加・変更が容易な構造を有していることが望まれる．

LOTOS では仕様記述のスタイルとして表 1 に示すものが提案されている [VIS 88]. モノリシックスタイルは，MSC - LOTOS 変換の出力スタイルとして考えた場合，可読性や，仕様の追加・変更の容易性を考えると好ましくない．状態指向スタイルは，SDL の様な状態を陽に記述した拡張有限状態機械に基づいた仕様記述言語からの変換の場合は有力な候補と考えられるが，MSC では全ての状態を陽に記述するとは限らないために好ましくないと考える．リソース指向スタイルは，MSC 仕様のインスタンスをそのままリソースと考えることができ，変換の候補としては有力である．しかし，一般にリソース指向の記述はよりインプリメントに近い詳細設計段階で用いられる記述法であり，我々の考える仕様設計工程モデルからは望ましくない．

そこで，変換出力として制約指向スタイルを考える．制約指向スタイルの仕様は，一般に，システムを構成する各プロセス各々の通信に関する制約条件であるローカル制約と，通信を行うプロセス間の制約である end-to-end 制約の論理積で記述される．MSC からの変換を考えると，MSC の各インスタンス毎の通信シーケンスは，インスタンス毎のローカル制約として記述できる．また，インスタンスに受信されるメッセージが FIFO バッファ処理されることも，ローカル制約の一部と考えることが出来る．従って，ローカル制約は通信シーケンスに関する制約条件と，通信がバッファ処理されなければならないというバッファ制約の論理積として表される．一方，end-to-end 制約は，インスタンス間の通信路の振舞に関する制約条件として記述される．図 1 の MSC を例とした，制約条件全体の構造を図 3 に示す．メッセージによる通信は，MSC インスタンス名と同名のゲートで観測される，3.1 節で述べる様な値の入出力で表現され，ローカル制約と end-to-end 制約は，それらのゲートを共有し，同期並列オペレータ || で結合され，論理積の関係にある．ローカル制約は，それぞれの MSC インスタンス毎のローカル制約の独立並列関係で記述される．インスタンス毎のローカル制約は，前述の通り，通信シーケンスに関する制約と，受信バッファに関する制約の，内部ゲートを介した一般並列関係で記述される．この内部ゲートでは，受信したメッセージの消費が観測される．

制約指向スタイルでは，制約条件を付加・変更することにより，設計者は容易に仕様の付加・変更を行うことができる．また，制約指向で変換出力した場合は，インスタンス間の通信がどの様に行われるかを制約条件として陽に記述するが，これらの制約条件は 3 章で示すように，ほぼ定型であるので，LOTOS での設計環境でライブラリ化して設計者から隠蔽することも可能であり，さらに，設計者が陽に記述を加えることで，4 章で示すように，通信路の品質が悪く，遅延やメッセージの欠落が発生し得るといった，様々な通信路環境をエミュレートすることも可能となる．

3 MSC の LOTOS 変換

本章では，MSC のメッセージの LOTOS 上での表現方法を述べ，制約指向の LOTOS 仕様に変換した場合の ローカル制約と end-to-end 制約各々の構造について述べる．

3.1 通信の表現

本変換方法では，他の MSC ツールとのデータ互換性を考え，入力は テキスト表現の MSC とした．テキスト MSC では，メッセージの送受信は 5 項組 $\langle Mo, Name, Inst, Parm, Adr \rangle$ で表される．ただし，Mo は，送信・受信の区別，$Name$ はメッセージ名，$Inst$ はメッセージインスタンス名，$Parm$ はパラメタリスト，Adr は相手先のアドレスである．本変換方法では，これらをすべて LOTOS の

表 1: LOTOS の記述スタイル一覧

モノリシックスタイル	システムの外部的な挙動を，構造を持たせずに平坦に記述
制約指向スタイル	システムの外部的な挙動を，種々の独立した制約条件の集まりで記述
状態指向スタイル	システムの状態を記述上に陽に反映させて記述
リソース指向スタイル	システムの構成要素を記述上に反映させて記述

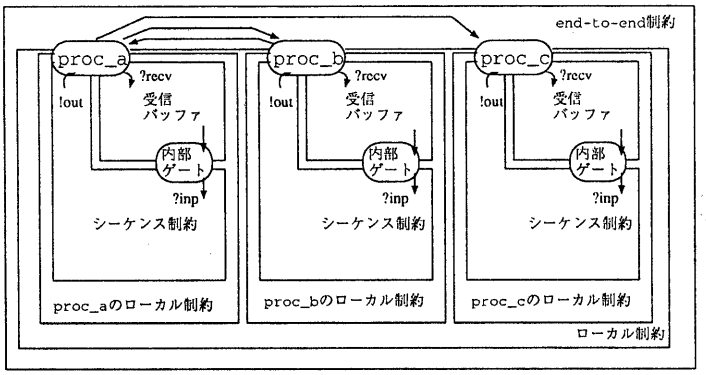

図 3: 制約条件の構造

抽象データ型として定義し，その値のゲートへの入出力でメッセージの送受信を表現する．以下に，各々の抽象データ型定義を述べ，送受信の例を示す．

送受の区別 *Mo* は，図 4 に示す抽象データ型 CommunicationModeType で表現できる．

```
type CommunicationModeType is Boolean
  sorts Mode
  opns inp,out,recv : → Mode
       _eq_ : Mode,Mode → Bool
  eqns ofsort Bool
    (* 演算子 _eq_ の定義: 省略 *)
endtype (* CommunicationModeType *)
```

図 4: 送受信の事象の区別を表す抽象データ型

ここで，inp，out は各々メッセージの送受信を表し，recv はメッセージが受信先インスタンスの受信バッファに格納されることを表す．

メッセージ名 *Name*，メッセージインスタンス名 *Inst*，パラメタリスト *Param* は，定数演算として MSC 上での識別子名を持つ抽象データ型 MessageType，SignalInstanceType，ParamType としてそれぞれ定義される．ただし，MessageType でのメッセージ名の定義では，パラメタを持つ場合はその ParamType のソートを定数演算の引数リストとして与えるよう定義しなければならない．例えば，パラメタ p1,p2 を持つメッセージ m2 の定義を記述した MessageType は，図 5 に示すように定義できる．

```
type MessageType is Boolean
  sorts Message
  opns m2 : Parm,Parm → Mode
       _eq_ : Message,Message → Bool
  eqns ofsort Bool
    (* 演算子 _eq_ の定義: 省略 *)
endtype (* MessageType *)
```

図 5: メッセージ名を表す抽象データ型

Adr は，MSC では通信する相手先のインスタンス名であるが，本変換方法では出力される LOTOS の読みやすさを考え，メッセージの送信元のインスタンス名との組で表現することとした．この抽象データ型 AddressType を図 6 に示す．

```
type AddressType is Boolean, InstanceType
  sorts Address
  opns from_to : Instance,Instance → Address
       _eq_ : Address,Address → Bool
  eqns ofsort Bool
    (* 演算子 _eq_ の定義: 省略 *)
endtype (* AddressType *)
```

図 6: メッセージのアドレッシングを行うための抽象データ型

ここでソート Instance を持つ抽象データ型 InstanceType は，変換する MSC 仕様に含まれる全てのインスタンス名を定数演算子として定義したデータ型である．

以上の様に抽象データ型を定義することにより，メッセージの送受信は定義した抽象データ型のゲートでの入出力で表現される．メッセージの送信の形式を図 7 に示す．

OutGate!out!*Message*!*SignalInstance*!*Address*

図 7: メッセージ送信の表現形式

ここで *OutGate* は送信を行うインスタンス名を持つゲート，*Message*は抽象データ型 MessageType で定義したメッセージ名，*SignalInstance* はシグナルインスタンス名，*Address* は抽象データ型 AddressType で定義したアドレッシング情報である．

例えば，図 1 に示した MSC のインスタンス proc_b の場合，メッセージ m3 の出力は以下のようになる．

```
proc_b!out!m3!0!from_to(proc_b,proc_a);
```

図 8: メッセージ送信の記述例

一方，メッセージの受信の表現は，図 9 に示すようになる．

```
InpGate!Mode?Message?SigInst?Address
  [(Message eq MesName
   and (SigInst eq InstName)
   and (Address eq AdrInfo)]
```

図 9: メッセージの受信の表現

ここで，*InpGate* は，メッセージの受信が観測されるゲート名，*Mode* は，CommunicationModeType 型の定数演算子 recv か inp，*SigInst*，*Address* は，それぞれ SignalInstanceType，AddressType 型の変数であり，定数 *MesName*，*InstName*，はそれぞれ，受信されるメッセージ名，シグナルインスタンス名，*AdrInfo* は受信されるメッセージのアドレッシング情報である．

3.2 ローカル制約の構造

ローカル制約 Local_Constraints は，2.2 節で述べたように，変換する MSC 仕様に含まれる環境を含めた全てのインスタンス毎のローカル制約の論理和で表される．これは例えば，図 10 に示すようになる．

```
process Local_Constraints[Inst_1,...,Inst_m
                          ]: noexit:=
  Inst_1_Local[Inst_1]
|||
  ...
|||
  Inst_m_Local[Inst_m]
endproc (* Local_Constraints *)
```

図 10: ローカル制約のトップレベルの構造

ここで $Inst_1,...,Inst_m$ は，MSC 仕様に含まれるインスタンス名であり，インスタンス間の通信を行うためのゲート名，および，そのインスタンス毎のローカル制約を表すプロセス名の識別子に用いている．

このインスタンス毎のローカル制約は，メッセージのシーケンスに関する制約と，受信メッセージは FIFO バッファリングされなければならないという制約条件から構成される．これは，図 11 に示すように記述される．

```
process Inst_i_Local[Inst_i]: noexit:=
  hide BUF in
    Inst_i_Sequence[Inst_i,BUF]
|[BUF]|
    buffer[Inst_i,BUF](create)
endproc (* Inst_i_Local *)
```

図 11: インスタンス毎のローカル制約

ここでプロセス $Inst_i$_Sequence $(1 \leq i \leq m)$ は，インスタンス $Inst_i$ の通信シーケンスに関する制約であり，プロセス buffer はバッファ制約である．ゲート BUF が，図 3 で示したローカル制約内の内部ゲートに当たる．

シーケンス制約 $Inst_i$_Sequence は，基本的には以下のように変換される．

```
process Inst_i_Sequence[Inst_i,BUF]: noexit:=
 MessageInOut_1;
 ...
 MessageInOut_n;
  stop
endproc (* Inst_i_Sequence *)
```

図 12: シーケンス制約の例

ここで $MessageInOut_i (1 \leq i \leq n)$ は，3.1 節で述べた通信の表現である．ただし，メッセージの送信ではゲート *OutGate* として $Inst_i$ を，受信ではゲート *InpGate* に BUF を，定数 *Mode* には inp を用いる．

バッファ制約の記述は，一般的な FIFO バッファの記述であり，ここでは省略するが，メッセージの Overtaking を含むような仕様でない限り，バッファ制約はすべてのローカル制約に共通に記述できることに注意されたい．

3.3 end-to-end 制約の構造

end-to-end 制約は，インスタンス名ゲートに出力されるメッセージは相手先のインスタンス名ゲートに送られなければならないという制約であり，以下のような簡潔な制約条件，

```
process InstanceToInstanceTransfer[Src,Dest]
          (Src,Dest:Instance): noexit:=
 Src!out?Name:Message
   ?Inst:SignalInstance!from_to(Src,Dest);
 Dest!recv!Name!Inst!from_to(Src,Dest);
  InstanceToInstanceTransfer[Src,Dest](Src,Dest)
endproc (* InstanceToInstanceTransfer *)
```

図 13: end-to-end 制約

を，通信を行うインスタンス間に掛けるように記述する．

4 適用例と考察

本章では，ISDN の基本呼サービスを例にし，その LOTOS 変換の出力を示すと共に，その例を用いて，MSC の言語仕様に含まれる幾つかの機能の変換法を提案する．また，複数ページから構成される MSC document の形式の MSC 仕様の変換法を示すため，前述の例題の MSC にページを付加し，その場合の変換法を示す．最後に，LOTOS 変換を用いた MSC のシミュレーション，および検証について考察する．

4.1 ISDN 基本呼サービス

例題として用いた ISDN 基本呼サービスの MSC 仕様例を図 14 に示す．この LOTOS 変換の出力結果を以下に示す．

```
specification StandardSequence[UserA,TS,UserB]:
                                          noexit
(* 抽象データ型の定義: 省略 *)
 behavior
  StandardSequence[UserA,TS,UserB]
   where
    process StandardSequence[UserA,TS,UserB]:
                                          noexit:=
     Local_Constraints[UserA,TS, UserB]
   ||
     Global_Constraints[UserA,TS,UserB]
      where
```

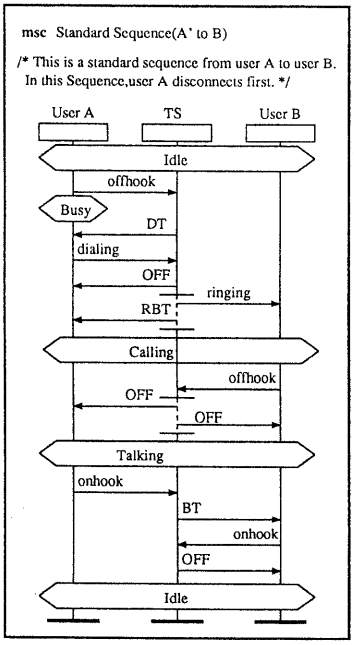

図 14: ISDN 基本呼サービス

```
process Local_Constraints[UserA,TS,UserB]:
                                        noexit:=
  hide GC0,GC1  in
  UserA_Local[UserA,GC0,GC1]
|[GC0,GC1]|
  UserB_Local[UserB,GC0,GC1]
|[GC0]|
  TS_Local[TS,GC0]
   where
    process UserA_Local[UserA,GC0,GC1]:
                                        noexit:=
     hide BUF  in
     UserA_Sequences[UserA,BUF,GC0,GC1]
    |[BUF]|
     buffer[UserA,BUF](create)
      where
      process UserA_Sequences[UserA,BUF,GC0,
                                  GC1]: noexit:=
        UserA_Idle[UserA,BUF,GC0,GC1]
         where
       process UserA_Idle[UserA,BUF,GC0,GC1]:
                                        noexit:=
          GC0!Idle;
          (* offhook 送信: 略 *)
          UserA_Busy[UserA,BUF,GC0,GC1]
           where
        process UserA_Busy[UserA,BUF,GC0,GC1]:
                                        noexit:=
            (* DT 受信: 略 *)
            (* dialing 送信: 略 *)
            (* OFF 受信: 略 *)
            (* RBT 受信: 略 *)
           UserA_Calling[UserA,BUF,GC0,GC1]
             where
            process UserA_Calling[UserA,BUF,
                                  GC0,GC1]: noexit:=
              GC1!Calling;
              (* メッセージ OFF の受信: 省略 *)
             UserA_Talking[UserA,BUF,GC0,GC1]
              where
              process UserA_Talking[UserA,
                    BUF,GC0,GC1]: noexit:=
                GC0!Talking;
                (* onhook 送信: 略 *)
               UserA_Idle[UserA,BUF,GC0,GC1]
                 endproc (* UserA_Talking *)
             endproc (* UserA_Calling *)
          endproc (* UserA_Busy *)
       endproc (*UserA_Idle *)
    endproc (* UserA_Sequences *)
endproc (* UserA_Local *)
process UserB_Local[UserB,GC0,GC1]:
                                        noexit:=
(* インスタンス UserB のローカル制約: 省略 *)
endproc (* UserB_Local *)
process TS_Local[TS,GC0]:
                                        noexit:=
  hide BUF  in
  TS_Sequences[TS,BUF,GC0]
|[BUF]|
  buffer[TS,BUF](create)
   where
    process TS_Sequences[TS,BUF,GC0]:
                                        noexit:=
     TS_Idle[TS,BUF,GC0]
      where
       process TS_Idle[TS,BUF,GC0]:
                                        noexit:=
        GC0!Idle;
        (* offhook 受信: 略 *)
        TS!out!DT!0!from_to(TS,UserA);
       (* メッセージ dialing の受信など: 省略 *)
        BUF!inp
        ?Mes2:Message
        ?SiI2:SignalInstance
        ?Adr2:Address[(Mes2 eq offhook) and
                  (SiI2 eq 1) and
              (Adr2 eq from_to(UserB,TS))];
        (
        TS!out!OFF!1!from_to(TS,UserA); exit
        |||
        TS!out!OFF!2!from_to(TS,UserB); exit
        )
        > >
        TS_Talking[TS,BUF,GC0]
         where
          process TS_Talking[TS,BUF,GC0]:
                                        noexit:=
           GC0!Talking;
           BUF!inp
           ?Mes0:Message
           ?SiI0:SignalInstance
         ?Adr0:Address[(Mes0 eq onhook) and
                   (SiI0 eq 0) and
              (Adr0 eq from_to(UserA,TS))];
           TS!out!BT!0!from_to(TS,UserB);
           BUF!inp
           ?Mes1:Message
           ?SiI1:SignalInstance
         ?Adr1:Address[(Mes1 eq onhook) and
                   (SiI1 eq 1) and
              (Adr1 eq from_to(UserB,TS))];
           TS!out!OFF!3!from_to(TS,UserB);
           TS_Idle[TS,BUF,GC0]
          endproc (* TS_Talking *)
       endproc (* TS_Idle *)
    endproc (* TS_Sequences *)
endproc (* TS_Local *)
process buffer[INP,OUT](Q:Queue):
                                        noexit:=
```

```
        (* バッファ制約: 省略 *)
      endproc (* buffer *)
    endproc (* Local_Constraints *)
    process Global_Constraints[UserA,TS,UserB]:
                                          noexit :=
     (* グローバル制約: 省略 *)
    endproc (* Global_Constraints *)
  endproc (* StandardSequence *)
endspec (* StandardSequence *)
```

図 15: MSC による ISDN 基本呼サービスの LOTOS 変換例

4.2 MSC の各機能の LOTOS への変換

本節では，MSC の言語仕様に含まれる機能の内，例題で用いたコンディション，コリージョンについて，その変換法を述べる．

4.2.1 コンディション

コンディションは，シーケンス制約をそのコンディションで定義されるシーケンス毎に分割し，それらの末尾でそれぞれの次のコンディションのシーケンス制約をインスタンシエートするように記述することで，表現できる．ただし，グローバルコンディションに対しては，共有するインスタンス毎に同期ゲートを設ける必要がある．これは，LOTOS 上のインスタンス毎のローカル制約は完全に非同期に動作するため，グローバルコンディションを共有するインスタンスが同時に，その状態に遷移するよう明示しなければならないからである．例題では，例えばインスタンス UserA は，コンディション Idle，Busy，Calling，Talking に属するシーケンスは各々，UserA_Idle，UserA_Busy，UserA_Calling，UserA_Talking で制約され，各々が次の制約条件を呼び出すように変換されている．

4.2.2 コリージョン

コリージョンは，その関係にある通信のシーケンス制約での記述を $B_1, B_2, ...B_n$ とし，その前後のシーケンスの記述を各々，B_0，B_{rest} とした時，$B_0; (B_1; \mathbf{exit} \;|||\; B_2; \mathbf{exit} \;|||\; ...B_n; \mathbf{exit}) \gg B_{rest}$ とすることで，変換できる．例題では，インスタンス TS のコンディション Talking の直前の，2 つのメッセージ OFF の送信のコリージョンの変換を示した．

4.3 MSCs の合成

MSC の仕様記述は複数ページの MSC から構成される MSC document の形式を取る場合が一般的である．本節では，その様な MSCs の合成法について述べる．

MSCs の合成法としては，[MAU 93] の様な，複数の MSCs を 1 つの構造を持たない MSC に合成する研究が知られている．この様なアプローチは非常に興味深いが，これを LOTOS 変換に応用する場合に，その変換結果を LOTOS 上でさらに詳細化することを考えると，入力 MSCs が大規模となる場合には望ましくない．そこで，本変換法では LOTOS のチョイス演算子を用いて，MSCs の構造を保存する様な変換を考える．この変換は非常に単純であり，MSCs の合成の始点をコンディションと考え [CCI 92b]，インスタンス $I_1, ..., I_m (m \geq 1)$ に共有されるコンディション C に対する，$MSC_1, ..., MSC_n (n \geq 2)$ それぞれのシーケンス制約の記述を LOTOS プロセス $I_1\_C_1, ..., I_m\_C_n$ とした時，その合成を $I_1\_C_1 [] I_2\_C_2 [] ... [] I_m\_C_n$ で与える．この様に変換することにより，MSCs のそれぞれのページ単位に LOTOS プロセスが生成されるので，設計者は比較的容易に MSCs と 変換された LOTOS 仕様との対応を取ることができる．

以下に，図 14 に示した ISDN 基本呼サービスに対する，図 16 に示した MSC 仕様の合成を示す．この例では，

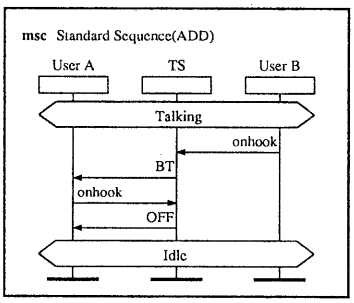

図 16: 追加 MSC 仕様 (披呼者先掛け)

発呼者先掛けであった図 14 の仕様に，披呼者先掛けの仕様を追加合成する．インスタンス TS を例に取る．追加仕様のインスタンス TS のコンディション Talking におけるシーケンス制約プロセス TS_Talking_ADD は以下のようになる．

```
process TS_Talking_ADD[TS,BUF,GC0]: noexit:=
 GC0!Talking;
 BUF!inp
 ?Mes0:Message
 ?SiI0:SignalInstance
 ?Adr0:Address[(Mes0 eq onhook) and
              (SiI0 eq 0) and
              (Adr0 eq from_to(UserB,TS))];
 (* 一部略 *)
 TS!out!OFF!4!from_to(TS,UserA);
 TS'Idle[TS,BUF,GC0]
endproc (* TS_Talking_ADD *)
```

図 17: インスタンス TS に対する追加制約

このプロセスを合成するには，プロセス TS_Idle を以下のように変更して出力する．

```
process TS_Idle[TS,BUF,GC0]: noexit:=
 (
    TS!out!OFF!1!from_to(TS,UserA); exit
 |||
    TS!out!OFF!2!from_to(TS,UserB); exit
 )
 ≫
 (
    TS_Talking[TS,BUF,GC0]
 []
    TS_Talking_ADD[TS,BUF,GC0]
  where
    (* 以下略 *)
endproc (* TS_Idle *)
```

図 18: インスタンス TS に対する追加制約の合成

同様の処理を，他のインスタンス UserA,UserB に対しても行う．

4.4 仕様検証への適用性

本節では，本論文における提案の仕様検証への適用性について述べる．

LOTOS では，記述した仕様間の数学的関係を調べることにより，仕様の詳細化の正当性や，規約や勧告に沿った正しい仕様であるかを調べることができる．従って，本論文の提案を用いることにより，MSC 仕様同士や MSC 仕様と LOTOS 仕様間の関係を調べることで，仕様検証の可能性をもたせることができるようになる．

ただし，本論文で提案した方法は非同期通信を無限長の FIFO キューを用いて実現しているため，遷移の数が無限となり，遷移システムベースの検証ツールでは機械的な検証ができない可能性がある．この場合は，FIFO キューに長さの制限 (例えば 1) を持たせることは容易であり，変換ツールでバッファ制約の出力を切替えられるようにすればよい．

さらに，出力された LOTOS 仕様を LOTOS のシミュレータで実行することにより，MSC 仕様のシミュレーションを行うことも可能である．この際に，例えばインスタンス間の通信に関する end-to-end 制約に僅かな変更を加えることにより，インスタンス間の通信路の品質が悪く，メッセージの欠落を生じ得るような場合をエミュレートすることも容易である．

4.5 変換方法の評価

まず，入力となる MSC 仕様の規模と，出力される LOTOS 仕様の規模を比較する．図 14 の例では，テキスト MSC ベースで約 55 行の記述であるが，この LOTOS 出力は約 350 行となる．この差は，様々な抽象データ型の宣言部 (約 180 行，約 50 %) と，バッファ制約及び end-to-end 制約の記述 (約 40 行，約 14%) が半分以上を占め，設計者にとって重要なシーケンス制約の部分の記述は，本変換方式での通信の記述は冗長ではあるが，行数的には差は少ない．

5 まとめ

本論文では，MSC と LOTOS を用いた通信システム設計支援の一環として，MSC 仕様の LOTOS 仕様への変換法を提案した．本変換法では，MSC 仕様を，LOTOS で概要仕様を作成する際に一般的に用いられることの多い制約指向スタイルで LOTOS に変換できることを，例を用いて示した．また，本変換法を用いることにより，MSC 仕様に対する verification への適用可能性を示すことができた．

今後の課題としては，未検討である MSC の機能 (サブ MSC，プロセス生成等) に対しての検討や，変換の正当性の証明，システムとしての実装が挙げられる．

参考文献

[CCI 92a] CCITT, "Functional Specification and Description Language (SDL)", Recommendation Z.100, 1992.

[CCI 92b] CCITT, "Message Sequence Charts (MSC)", Recommendation Z.120, 1992.

[ISO] ISO, "Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", ISO 8807, 1989.

[JOZ 93] J. D. Man, "Towards a Formal Semantics of Message Sequence Charts", In SDL'93: Using Objects, 1993.

[KAM 89] 神長裕明, 高橋薫, 白鳥則郎, 野口正一, "LOTOS 仕様の等価性とその判定法", 信学論 (D-I), **J72-D-I**, 5, pp.367-376, 1989.

[EIJ 89] P.H.J.V Eijk, C.A.V. Vissers and M. Diaz, "The Formal Description Technique LOTOS", North-Holland, 1989.

[WAK 90] 若原 恭, "SDL 言語の特質と処理系の現状と動向", 情報処理, **31**, 1, pp.23-34, 1990.

[KAO 90] 高橋薫, 神長裕明, 白鳥則郎, "LOTOS 言語の特質と処理系の現状と動向", 情報処理, **31**, 1, pp.35-46, 1990.

[KAO 93] 高橋薫, 山野敬一郎, 太田正孝, "プロセス仕様の検証のための模倣性判定法", 信学論 (D-I), **J76-D-I**, 1, pp.27-30, 1993.

[AND 93] 安藤津芳, 太田正孝, 高橋薫, "SDL 仕様の LOTOS による解釈", 信学論 (D-I) **J76-D-I**,6, pp.300-314, 1993.

[SAR 93] Katsuyuki Sarashina, Tsuyoshi Ando, Masataka Ohta, Yoshiaki Tokita, and Kaoru Takahashi, "An Integrated Specification Support System for Communication Software Design Based on Stepwise Refinement and Graphical Representation", In FORTE'93 Participants' Proceedings, 1993.

[VIS 88] C.A. Vissers, G. Scollo, M. van Sinderen, "Architecture and Specification Style in Formal Descriptions of Distributed Systems", Protocol Specification, Testing, and Verification VIII, pp.189-204, Elsevier Science Publishers B.V., 1988.

[MAU 93] S. Mauw, M. van Wijk, and T. Winter "A Formal Semantics of Synchronous Interworkings", In SDL'93: Using Objects, 1993.

[OHT 93] 太田他, "通信ソフトウェア設計支援環境:ITECS(1) — 全体構成 —", 情報処理学会第 46 回全国大会, 6ZJ-2, 1993.