MSCとルールを併用した通信サービスの効率的開発法*

6C－9

寺内 敦, 金井 敦†
NTT ソフトウェア研究所‡
{terauchi,kanai}@slab.ntt.jp

1 はじめに

通信サービスの開発において、近年、メッセージシーケンスチャート(MSC)[1]がその理解性の良さなどから多く使われてきている。従来、MSCはシステムの粗い仕様を記述するのみに用いられることが多かったが、MSCでエラー処理を含むすべての仕様を記述しようとするアプローチも提案されている[2]。

このアプローチは到達可能解析に伴う状態爆発の問題を回避できるという点で有望であるが、記述すべきMSCの枚数が多くなり仕様の抜けなどが起こりやすくなるという問題がある。この問題を克服するために、筆者らはMSCとエラー処理を集約的に記述することのできるルールを併用した仕様記述法を提案し、その有効性を実験的に確認した[3]。

この記述法の実用性をさらに高めるためには、記述された仕様の正しさを検証する手段を与えることが不可欠である。そこで本稿では、まずルールからMSCを生成するアルゴリズムを示す。そして、MSCとルールの仕様から得られるMSC集合が、MSC集合に要求される3つの性質feasibility, consistency, completenessのうちfeasibility, consistencyを満たすための条件を示す。

2 前提とする通信サービス開発法

2.1 概要

筆者らが提案した通信サービス開発法[3]は、MSC生成ツールEHGENと適当なMSC→SDL変換システムを用いて実現される。本稿ではMSC変換システムとしてSDE[2]を用いる。設計者はまず、システムの主要な動作(正常系と呼ぶ)をMSCによって記述する。正常系MSCとルールをMSC生成ツールEHGENに入力すると、EHGENはエラー処理MSCの集合を生成する。正常系MSCとエラー処理MSCの集合をSDEに入力することにより、MSC集合の完全性の検証やプロセス毎の状態遷移図およびCプログラムの自動生成が可能である。

2.2 ルールの記述

まず、本稿では正常系からエラー処理への分岐は信号分岐のみで起こると仮定する。

このとき、エラー処理MSCを作るのに必要な情報は

- 分岐が起こ(り得)る場所(状態)…判断点
- 分岐を起こす信号シーケンス…トリガ
- 分岐後の処理…エラー処理
- 分岐後の処理の目標状態…戻り点

である。これに基き、ルールは以下の形式で与える。

```
If(判断点検出条件)on(トリガ)then(エラー処理,戻り点)
```

ここでルールの要素はすべてMSC形式で与えるものとする。また、本稿で対象とするエラー処理はエラー検出後、サービスの終了状態あるいは初期状態に戻るものとする。

2.3 MSC生成アルゴリズム

定義1 (MSC) *MSC* m を以下のように定義する。

$$\begin{cases} u_p = \{e_{p_1}, \ldots, e_{p_n}\} \\ m = \{u_1, \ldots, u_n, Y\} \end{cases} \cdots(1)$$

e_{p_k} はプロセス p が初期状態から k 番目に実行するイベント、u_p はプロセス p が m 中で実行するイベント列、Y は m におけるプロセス間の送受信イベントの対応を示す2項関係で $Y = \{(e_s, e_r) | e_s \in E_s^p, e_r \in E_r^p\}$ と定義する。ここで、(e_s, e_r) は e_s によって送信されたメッセージは e_r によって受信されることを表す。また、m に含まれるイベントのうち、他のプロセスへメッセージを送信するイベントの集合を E_s^p、他のプロセスからのメッセージを受信するイベントの集合を E_r^p とする。 □

定義2 (MSCのマッチとPreambleシーケンス) 入力となる正常系 *MSC* m が上記 *(1)* のように定義されているとする。

m の**部分シーケンス** m' を以下のように定義する。

$$\begin{cases} u'_p = \{e_{p_i}, \ldots, e_{p_j}\} (1 \le p_i < p_j \le p_n) \\ m' = \{u'_1, \ldots, u'_n, Y'\} : Y' \subseteq Y \end{cases}$$

u'_p は u_p の部分列であり、u_p におけるイベントの出現順序は u'_p においても保存されている。m' が m の部分シーケンスであるとき、m' と m は**マッチする**と言う。また、このとき、

$$\begin{cases} u''_p = \{e_{p_1}, \ldots, e_{p_{i-1}}\} \\ m'' = \{u''_1, \ldots, u''_n, Y''\} : Y'' \subseteq Y \end{cases} \cdots(2)$$

と定義される m'' を m の (m' による)Preamble(シーケンス) という。 □

定義3 (MSCの結合) 2つの *MSC* m, m' を以下のように定義する。

$$\begin{cases} m = \{u_1, \ldots, u_n, Y\} (u_p = \{e_{p_1}, \ldots, e_{p_x}\}) \\ m' = \{u'_1, \ldots, u'_n, Y'\} (u'_p = \{e'_{p_1}, \ldots, e'_{p_y}\}) \end{cases}$$

このとき、m と m' から以下のような *MSC* M を作ることを m と m' を**結合する**という。

---

*Efficient Development for Communications Services by using MSCs and Rules

†Atsushi TERAUCHI, Atsushi KANAI

‡NTT Software Laboratories

$$\begin{cases} U_p = \{u_p, u'_p\} = \{e_{p_1}, \ldots, e_{p_x}, e'_{p_1}, \ldots, e'_{p_y}\} \\ M = \{U_1, \ldots, U_n, Y\} \end{cases} \quad \square$$

上の定義を用いてルールからエラー処理 MSC を生成する手続きを以下に示す。

1. 正常系 MSC m とルールの判断点検出条件 MSC m' を比較して m と m' がマッチするかどうか調べる
2. マッチしたら m の (m' による)Preamble を取り出す
3. Preamble と、ルール中のトリガ MSC、エラー処理 MSC を結合する

エラー処理 MSC 生成の概要を図 1に示す。

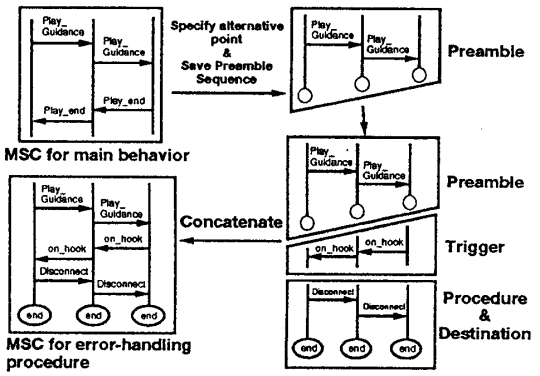

図 1: エラー処理 MSC の生成

3 仕様の完全性

一般に、MSC 集合には feasibility, consistency, completeness という 3 つの性質 [2] を満たすことが要求され、3 つの性質を満たすときに MSC 集合は「完全である」という。本稿では、MSC とルールの仕様に要求される性質「完全性」を、MSC とルールの仕様から提案アルゴリズムによって生成される MSC 集合が「完全である」ことと定義する。本章では、MSC とルールの仕様が「完全性」の 3 つの条件のうち、feasibility と consistency を満たすための条件について検討する。

3.1 feasibility

MSC 集合が feasiblity を満たすとは、集合に含まれる個々の MSC についてすべてのイベントの送受信の対応が取れていることである。

まず、以下の補題が成り立つ。

補題 1 *MSC* $m_1, \ldots, m_k$ *がすべて feasibility を満たすならば、それらを結合して得られる MSC* m *も feasibility を満たす。*

この補題が成り立つことは、m が feasibility を満たさない、すなわち送受信の対応の取れないイベントが少なくとも存在すると仮定すれば矛盾が起こることから容易に証明可能である。

次に、正常系 MSC m とルール $\mathrm{If}(C_m)\ \mathrm{on}(T_m)\ \mathrm{then}(P_m)$ からエラー処理 MSC m' が生成されたとする。ここで、m は feasibility を満たしているものと仮定する。このとき、補題 1より以下の定理が導ける。

定理 1 C_m, T_m, P_m *が共に feasibility を満たし、かつ、* C_m *による Preamble が feasiblity を満たすならば、* m' *は feasibility を満たす。*

RPC 型の各プロセスが同期して動作するサービスでは C_m が feasibility を満たせば Preamble が feasibility を満たすことが保証されるため、定理 1より、C_m,T_m,P_m の feasibility を検証すれば生成される MSC の feasibility を知ることができる。

3.2 consistency

MSC 集合が consistency を満たすとは、任意の MSC 間で非決定的動作が生じないことである。

ここでは、正常系 MSC m と、m とルール $\mathrm{If}(C_m)\ \mathrm{on}(T_m)\ \mathrm{then}(P_m)$ から生成された MSC m' とが consistency を保つための条件を考える。このとき、m と m' はそれぞれ以下のように表せる。ここで、m,m' および Preamble はすべて feasibility を満たすと仮定する。

$$\begin{cases} m = \{Preamble, m_{post}\} \\ m' = \{Preamble, m'_{post}\} \end{cases}$$

ここで、m と $Preamble$ がそれぞれ定義 1の (1) と定義 2の (2) のように定義されているとすると m_{post} は以下のように定義される MSC である。

$$\begin{cases} u'''_p = \{e_{p_i}, \ldots, e_{p_n}\} \\ m_{post} = \{u'''_1, \ldots, u'''_n, Y'''\} : Y''' \subseteq Y \end{cases}$$

このとき、m と m' が consistency を満たす条件は明らかに m_{post} と m'_{post} が干渉[1]しないことである。アルゴリズムより m'_{post} の前半は T_m であり、さらに m_{post} の前半は C_m である。これにより以下の定理が成り立つ。

定理 2 *ルールに記述されたトリガ* T_m *と、判断点検出条件* C_m *とが干渉しなければ、ルールと正常系 MSC から生成されるエラー処理 MSC と元の正常系 MSC とは干渉しない。*

この定理は、ある正常系 MSC m とルールから提案アルゴリズムによりエラー処理 MSC m' が生成されたとき、m と m' が干渉しないことはルールの検証だけで分かることを示している。しかし、一般に正常系 MSC は複数あるから m とは別の正常系 m'' と m' とが干渉する可能性があるが、そのことはルールのみからは検証できない。

4 まとめと今後の課題

本稿では、MSC とルールを併用した仕様記述法における MSC 生成アルゴリズムを提案し、提案アルゴリズムにより MSC とルールから得られた MSC 集合が feasibility, consistency を満たすための条件を示した。

今後は、正常系 MSC とルールから生成される MSC 集合が completeness を満たすための条件および生成された MSC 集合に誤りが発見されたときに、元の正常系 MSC とルールの修正を支援するような機構の検討が必要である。

参考文献

[1] CCITT: "Recommendation Z.120", 1992.

[2] H. Ichikawa et al.: "SDE: Incremental Specification and Development of Communications Software", *IEEE Trans. Computers*, Vol. 40, No. 4, pp. 553–561, 1991.

[3] A. Terauchi, K. Yamanaka and J. Kato: "Efficient Specification Description of Communications Software via Message Sequence Chart and Rule", *Proc. of GLOBECOM'94*, 1994, To be published.

[1] 2 つの MSC 間に非決定的動作が含まれること