通信制御ソフトウェア開発における状態遷移表の実装効率化への取組み

～ZIPC によるコード生成自動化への取組み～

富士通 株式会社
共通開発本部 第二ソフトウェア開発統括部

三橋 崇

1. はじめに

近年の通信ネットワーク（図１）は、エンドユーザに提供するサービスの多様化、キャリア間の早期サービス提供競争が激化している。

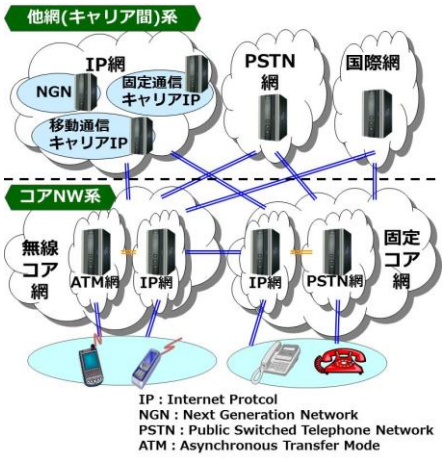


図１．通信ネットワーク イメージ

そのネットワークを構築する通信制御サーバ上のソフトウェア開発には、厳しい条件が要求され続けている（図２）。

図２．近年のキャリアからの要求条件

こういった現状に対し、通信制御ソフトウェア開発の現場では、実績のある既存資産を流用し、それに機能追加・変更を行うことで対応している。（図３）

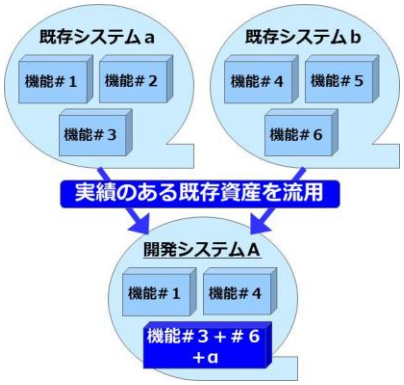


図３．既存資産流用 イメージ

このような流用開発を主体とする中、より短期間で高品質なソフトウェアの提供を可能とする開発プロセス構築を目的として、コード生成自動化に取組んだ。

本稿では、通信制御ソフトウェアの流用開発において ZIPC の導入に至った背景（導入評価結果）と今後の活動内容・課題について紹介する。

2. 通信制御ソフトウェア開発における状態制御の位置付けと開発手法

通信制御サーバ上のソフトウェアでは、複数の通信プロトコルの信号受信を「イベント(契機)」として状態を変化させ、状態毎に処理を決定する「状態制御」が重要な業務である。(図４)

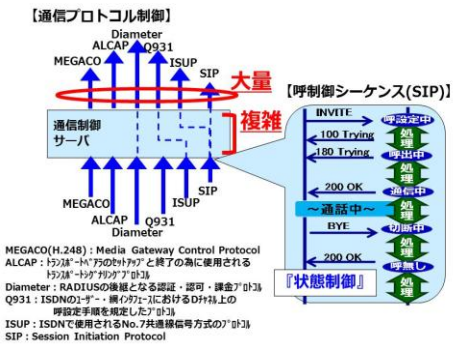


図４．通信制御ソフトウェアにおける状態制御

「状態制御」は、多くの通信プロトコルを扱うために非常に「大量」で、且つ１つの通信プロトコルを他の複数のプロトコルに変換する必要がある等、非常に「複雑」である。

これを特徴とした通信制御ソフトウェアの開発では、状態遷移表を用いて状態制御の網羅性を担保した設計を行い、多くの通信サービスを提供してきた。

3. 通信制御ソフトウェア開発における問題点

前章で述べた通信制御ソフトウェアの状態遷移表は、大きいものになると状態とイベントの組み合わせが数万となる。

この大量な状態遷移表を設計工程で多くの時間をかけて作成しているが、製造工程で「手組み」によりコード化していた為、人為ミスに起因するエディット誤り・漏れにより、問題混入が絶えず、品質安定まで非常に多くの時間を要していた。

そこで、手組みによるコード化率を削減することにより、問題混入を防止し、早期品質安定を目的とした、状態遷移表からのコード生成自動化に取組むこととした。

4. コード生成自動化ツールの選定条件

世の中には、様々なコード生成自動化ツールが存在するが、多くのツールがコード生成の自動化をターゲットにし過ぎており、自動生成するためのインプットが、設計工程で生産する設計書ではなく、コード生成自動化に向けた中間生産物的なものとなっている。

開発現場では、短期間のソフトウェア開発が要求され続けていることもあり、

コードとそれを生成する為だけの中間生産物の２つ資産のメンテナンスを行う手間を避け、結局、中間生産物を捨て、コードを直接編集するに至っていることが多い。

以上のことより、コード生成を自動化する為だけに必要な中間生産物は、作業の無駄を発生させる為、設計工程として必要な生産物からコード生成の自動化が出来なければ開発プロセスへの組み込みは難しい。

よって、今回の取組みで導入するツールは、設計工程生産物として可読性の高い状態遷移表からコード生成自動化が可能であることを最大の評価ポイントとして選定を行った。

5. ZIPC 導入評価結果

評価では、ZIPC Ver.10.0 SP3 により、通信制御ソフトウェア内の階層構成（図５）上のサービス制御階層の１機能ブロック(Function Block、以下、FB と表現)をトライアルターゲットとして、既存の状態遷移表を ZIPC に移植し、自動生成したコードと既存コードとの比較を行った。

図５. 通信制御ソフトウェア階層構成

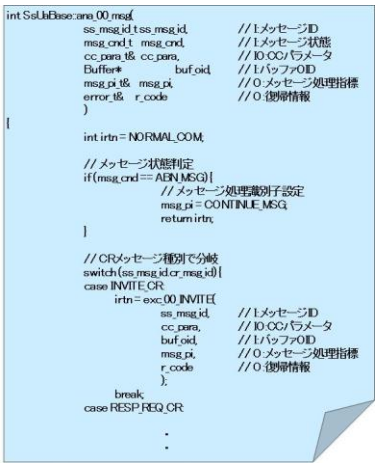
既存の状態遷移表とコードの自動生成の為に移植した ZIPC の状態遷移表を図６、７に示す。

	#define	SS_IDLE	SS_CALLPROC
	状態	空き	呼設定中
	method	ana_00_msg	ana_10_msg
define値	イベント	0x0	0x10
1	INVITE_CR	【動作関数】 exc_00_INVITE	【動作関数】 exc_INVITE_RETRY
		【動作内容】 ・入力情報チェック ・INVITEメッセージをコールデータに保持する ・隣接サーバ制御から送信先アドレスを取得する ・送信メッセージの生成 ・セッションタイマ制御 ・出接続側プロトコル制御の生成 ・バッファを載せ替えてイベント再分配実施	【動作内容】 ・出接続側プロトコル制御の再生成 ・送信メッセージの生成 ・分配継続
		【次状態】 →0x10(呼設定中)	【次状態】 状態遷移なし
8	BYE_CR	【動作関数】 ana_00_msg	【動作関数】 exc_10_BYE
		【動作内容】 ・ログ(CP-FAULT)を出力して透過	【動作内容】 ・入力情報チェック ・CANCELバッファ生成 ・バッファを載せ替えてイベント再分配実施 ・BYEバッファ解放
		【次状態】 状態遷移なし	【次状態】 状態遷移なし

図６. サービス制御 FB の既存状態遷移表

図７. サービス制御 FB の ZIPC 状態遷移表

既存コードと ZIPC の状態遷移表から自動生成したコードを図８，９に示す。

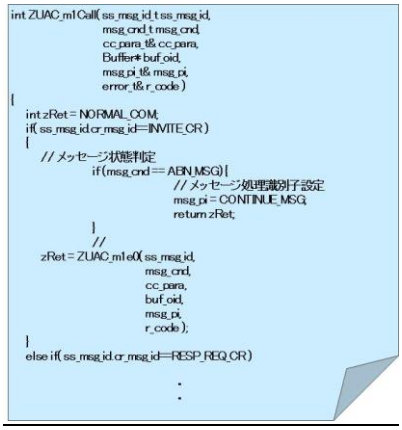


```
int SsUaBase::ana_00_msg(
        ss_msg_id_t ss_msg_id,          //1:メッセージID
        msg_cnd_t  msg_cnd,             //1:メッセージ状態
        cc_para_t& cc_para,             //10:CCパラメータ
        Buffer*    buf_oid,             //1:バッファOID
        msg_pi_t&  msg_pi,              //0:メッセージ処理指標
        error_t&   r_code               //0:復帰情報
        )
{
        int irtn = NORMAL_COM;

        //メッセージ状態判定
        if(msg_cnd == ABN_MSG){
                //メッセージ処理識別子設定
                msg_pi = CONTINUE_MSG;
                return irtn;
        }

        //CRメッセージ種別で分岐
        switch(ss_msg_id.cr_msg_id){
        case INVITE_CR:
                irtn = exc_00_INVITE(
                        ss_msg_id,      //1:メッセージID
                        cc_para,        //10:CCパラメータ
                        buf_oid,        //1:バッファOID
                        msg_pi,         //0:メッセージ処理指標
                        r_code          //0:復帰情報
                        );
                break;
        case RESP_RESQ_CR:
                .
                .
```

図８. サービス制御 FB の既存コード



```
int ZUAC_m1Call( ss_msg_id_t ss_msg_id,
                 msg_cnd_t msg_cnd,
                 cc_para_t& cc_para,
                 Buffer* buf_oid,
                 msg_pi_t& msg_pi,
                 error_t& r_code )
{
   int zRet = NORMAL_COM;
   if( ss_msg_id.cr_msg_id==INVITE_CR )
   {
      //メッセージ状態判定
            if(msg_cnd == ABN_MSG){
                        //メッセージ処理識別子設定
                        msg_pi = CONTINUE_MSG;
                        return zRet;
            }
            //
      zRet = ZUAC_m1e0( ss_msg_id,
                        msg_cnd,
                        cc_para,
                        buf_oid,
                        msg_pi,
                        r_code );
   }
   else if( ss_msg_id.cr_msg_id==RESP_REQ_CR )
                        :
                        :
```

図９．サービス制御 FB の ZIPC 自動生成コード

自動生成されたコードは、コード上で状態遷移する際の判断順序が「イベント」→「状態」と、既存コードとは逆という差分はあるが、状態遷移タイミングを指定することが可能であり、図９の自動生成コード、図８の既存コードを比較すると分かる通り、ほぼ同レベルのコードが生成でき、それを生成する元となった状態遷移表が設計工程生産物として作成した既存の状態遷移表と同等の可読性を維持出来てきていた。

トライアルターゲットとしたサービス制御 FB の中で、状態制御を行うほぼ全ての関数の自動生成において同様の結果が得られ、その割合はサービス制御 FB の約 49%(80/162 関数)を占めていた。

以上の結果より、可読性の高い状態遷移表の作成と既存と同等のコード生成の観点で、状態遷移表からのコード生成が一番優れていると判断したキャッツ社の ZIPC を通信制御ソフトウェアの流用開発プロセスに組込むこととした。

６．今後に向けて

今回の導入評価にて、設計工程生産物として作成した既存の状態遷移表と同等の状態遷移表が作成でき、更にそれを使って既存コードと同等のコードが自動生成できたことで、問題となっていた製造工程での問題混入を回避することが可能となる。更なる品質向上に向け、設計工程の充実として、より可読性の高い状態遷移表の設計(記述)を行うことによる高品質な設計作業を追求し、継続して ZIPC 開発元のキャッツ社と連携した活動を進める。

なお、今後の実プロジェクトへの展開に向けては、既存資産の状態制御を行う関数に対して、ZIPC の状態遷移表への移植を進め、後にその資産を流用した次期開発への着手を目指す。

また、今回のコード自動生成化の実現だけに留まらず、ZIPC の状態遷移表を用いた、自動試験やモデル検査への展開を模索し、更なるソフトウェア開発の効率化を実現していく。