情報通信機器におけるZIPCの適用 ～ZIPCリターン～

サクサシステムエンジニアリング株式会社
第一技術部
十日市　勉

●はじめに

当社、サクサシステムエンジニアリング㈱（旧：タイコーシステムエンジニアリング）は電話をはじめとする情報通信機器のソフトウェア開発をしています。どの業種でも同様と思われますが、近年開発する機器は多機種且つIP化を含むサービスの多様化により、そのソフトウエア開発規模は年々大きくなりつつあります。また、短期開発が要求されるなかでその複雑化するソフトウェアの開発効率と品質の確保が重要な課題となっています。

当社では、これらの問題の解決手段の一つとしてCASEツールによる開発効率の向上への取り組みを始めました。このたびZIPC2001を導入した経緯について報告いたします。

実のところ当社は一度ZIPCを導入しておりまして、その時にもこの ZIPC WATCHERSに導入経緯や感想を記載させて頂きました。（ZIPC WATCHERS Vol. 2 「ZIPCによるシステム開発について」をご参照ください）

●再度ZIPC

ZIPCを導入したことがあるので今では実績を上げていると思われそうですが、実際は導入時に適用したプロジェクトが最初で最後になり、現在まで使うことがありませんでした。その原因はZIPCの問題ではなく、開発の状況が適さなかったことが大きく上げられます。開発費が厳しいことで既存ソフトの流用が大半を占めることが多く、一部だけのCASEツール適用ではメリットが大きく出せないことが大きかったと言えます。しかし、サービスの多様化から新規に作成する部分が大きくなりCASEツールを利用するメリットが出てきました。

時代が進んでCASEツールも豊富な世の中になり、UMLを主体としたツールなども多く出ています。しかし、社内ではプログラムの振る舞いを表現するのに状態遷移表が一番多く使用されており、使い慣れたドキュメントが使用できるZIPCが相変わらず導入の筆頭株でした。

●開発上の問題とZIPCの適用

以下の表は状態遷移表を多用するプロジェクトで問題点とZIPC導入により改善される部分をまとめたものです。

作業フェーズ	現在の作業内容	ZIPC導入後
作成時の問題	状態遷移表はエクセルを使用し作成している。	ZIPCで状態遷移表示自体を作成する
	表が大きくなればなるほど、遷移先名などの記載に誤記が多くなる。（レビューの前に記載内容を確認する作業に非常に時間がかかる）	状態遷移表をシミュレートできる為、遷移先が誤記などで存在しないものである場合、ツールから指摘されることにより誤記のレビューへの持込は軽減される。
	状態遷移表中の処理セルの記載の仕方が個人毎にまちまちで統一されていない。また統一しづらい。（処理セルの処理の複雑度に依存して、記載内容がまちまちになる）	状態遷移表作成の専用機能で作成される為記載方はある程度統一される。
	作成途中で遷移の確認をパソコン上で行う場合、制御パスを確認したか否かを残す手段が面倒。（画面に色を塗りながら確認することも可能であるが、後で元に戻す作業が必要になる=>無駄）	上記状態遷移のシミュレートを行うことで、作成者による事前確認（自己DR）が可能。

確認作業時の問題点	状態遷移動作の確認は、紙面上で各遷移トリガを元にシミュレートを行い、紙面にマーキングなどを行い、制御パス確認を行っている。	複数人での制御パス確認は、シミュレート・カバレッジ機能を使用し、パソコン画面上で確認が可能。
	状態遷移表を紙にレビューのために印刷をすると、A3用紙を多数必要とする。レビューの時に紙の取りまわしで無駄な時間が多い。	上記の通り、パソコン画面上で確認が可能であるため紙に印刷する必要性は少ない。 （ただし、同時複数人で使用するだけのライセンスが必要）
	紙面上で確認を行う為、時間がかかる。 （レビューを繰り返せば品質は良くなるだろうが、時間ばかりかかってしまう）	同上
	紙面上でレビューを実施していると、問題指摘→修正→再レビューを行う必要があり、修正点だけではなく、状態遷移表中の影響範囲を再度レビューを行う必要がある。 指摘事項をその場で適用しながら、レビューを進めていきたい（手間を減らしたい）	状態遷移作成、シミュレートを併用することにより可能。指摘事項をその場で修正をし、再度シミュレート作業を行うことで継続的な確認を行うことが可能。
	紙面上でレビューを行うと、ある処理を動作させる遷移についてはよくレビューされるが、状態が空き状態へ復帰する処理などにレビュー漏れがありがち。	同上

●DR時間の節約がポイント

上記の表で示すとおりZIPC導入による改善点で重視されたのは膨大なDR時間と確認、修正にかかる時間の軽減です。設計上で避けては通れないDRですが数人で設計内容を検証するため、時間×人数の膨大な時間を必要とします。ZIPC導入で期待できるのが、ZIPCは状態の遷移ミスなどの簡単な誤りを前もって指摘してくれます（チェッカー機能）。また、実装前でも状態遷移をシミュレーションすることが出来るため、ここでも単純なミスを未然に発見することが出来ます（シミュレーション機能）。

このように、多くの人員を拘束するDRに単純なミスを持ち込まないことは、工数の削減効果があるだけでなく、より充実したDRを行うことが可能であり品質の向上も期待できます。

また、シミュレーションにより確認作業にかかるライフサイクルも短くすることが出来ます。

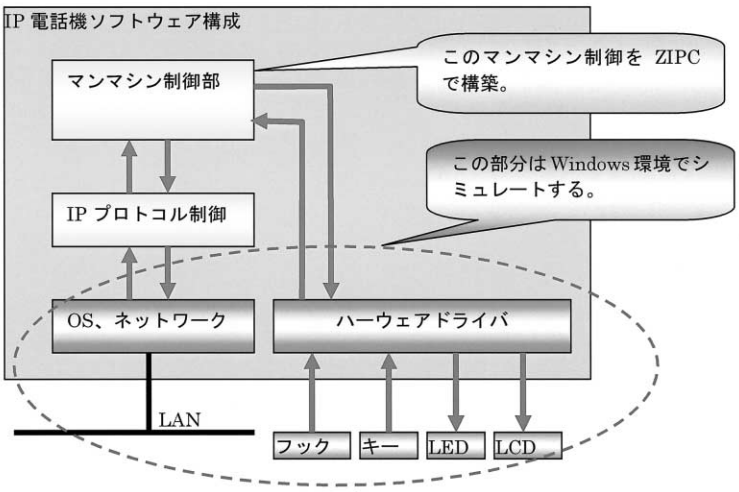
●IP電話機試作のマンマシンへ適用

適用の規模にもよりますが、明らかに導入効果が期待できることからZIPCを購入し、まず試作システムに適用を行いました。この目的はZIPCの機能を十分理解した上で製品には一番良い方法を適用するためです。

ターゲットはIP電話システムの試作IP電話機です。

ハードウエアは実際には作成せずVisual C++ 6.0を使用したPC上のシミュレーションで構築します。

以下に簡単なソフト構成を示しますが、ZIPCはマンマシン部分に適用しました。これは、IPプロトコル部分やドライバは流用が多く、マンマシン部分は製品毎のカスタマイズが多いことから実際の開発でもよくあるケースです。



ZIPCの機能は以下を使いました。
・エディター、チェッカー
・状態遷移シミュレーション
・コードジェネレーション

実際はPC上でバーチャルターゲットを動かすシミュレーションを実施していますが、これにはZIPCが持っているVIPなどの機能は使用していません。当社では既にPC上で組込みシステムをシミュレーションする手法（独自方式）が取り入れられており、使い慣れた環境を選択したためです。このように、自社で使える技術があればそれを合わせて使えばZIPCの機能を全て揃える必要はありません。

●全てが簡単とは言いません

このIP電話機の試作で本気でZIPCと格闘した訳ですが、以下のような状況でした。

①日頃から状態遷移表を使っている担当者であれば、チュートリアルとマニュアルを２日程度読みながら使うことで簡単に操作することが出来る。

②ZIPCの特徴である階層化を使い始めると、イベントの属性とグループ化を気にする事が多くなるため難易度が増してくる。これには２日程度格闘。

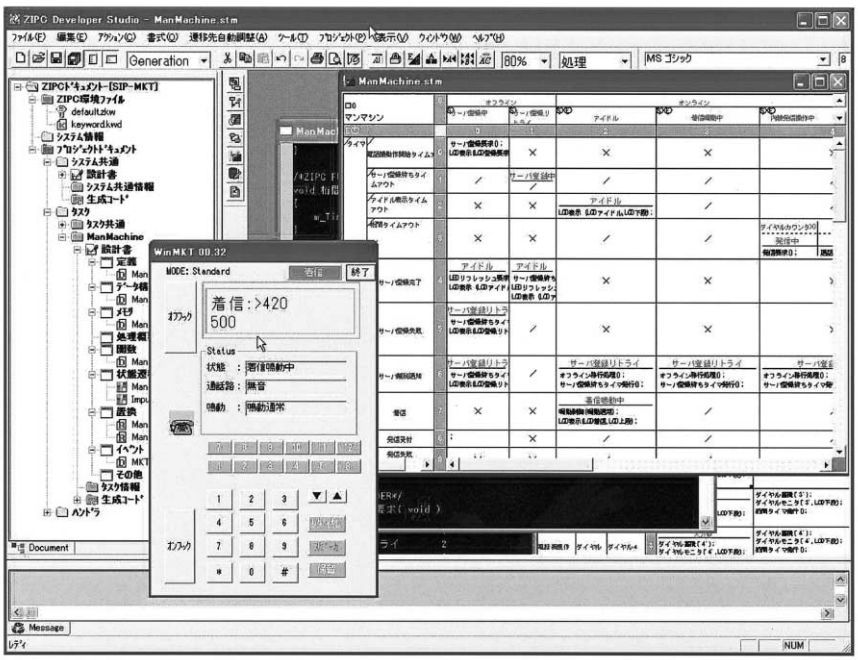
③コードジェネレーションを使うと、変換テーブルなどを作成する必要があるが、これがイベントの属性によって書き方を変える必要があるので最良の方法を導き出しておく必要がある。
これにはイベント属性の見直しとターゲット用の変換テーブルの書き直しで３日ほど試行錯誤。

④その後は、純粋にZIPCで各ドキュメントを拡張することでマンマシンを完成。

特に③が難しいところでした。グループ化＋階層化＋コード生成を行う際は、早い段階で特徴的なルートを作成し、イベントの属性などを変えながらコード生成を行うことで、最適な状態遷移表の構成を決めることが重要です。いくらツールと言えども、後から全体をやり直すのは辛いものがあります。

このシステムでは、最終的にイベントの属性は「フラグ・条件型」で統一しました。これが階層化したときに統一性を取ることが出来て、シミュレーション環境とも相性が良かったためです。

完成したIP電話シミュレーション



画面中央がVCで作成した独自バーチャルターゲット（バーチャルですが実際にソフトフォンとして使用可能）。

これからZIPCを導入し、コードジェネレーションを含む利用をするのであれば、中堅のエンジニアが２週間程度じっくり研究を行ってから本格的に使い始めるのが良いと思います。

●これは１つの方法論であった

実際に私自身がZIPCを使用してみましたが、ZIPCを使用する前の理解は、
「状態遷移表を専用ツールで入力出来て、便利に扱えるもの」
しかし、最新のZIPCを実際に使ってその機能を知った結果は
「これは固有の方法論である」
ということです。
実際に触れてみると状態遷移表がここまで賢く使えることには驚きを感じました。これは単なる状態遷移表ではなく新しい方法論であるかのような感触を受けました。
以下にその一部を挙げます。これらはZIPCのカタログなどに書かれていることですが、実際に使ってみるとその有効性の高さを実感できます。

①状態遷移表の階層化

状態遷移表は状態とイベントが多くなるほど難しくなりますが、状態やイベントをグループ化することができます。更に階層化することで１つの表は小さくなり、意味のないセルを少なくすることが出来ます。
この階層化は単純に状態図を分割するだけではなく機能単位のモジュールと考えることが出来ます。これは、オブジェクト指向とは言わないまでも、責務を持ったクラスを階層的に実装するのに似ており、大きな状態遷移を効率良く整理することが可能であり、更に大きなシステムの理解度も高まります。
但し、階層化するとコードジェネレーション時にイベントの取り扱いが若干難しくなるので注意が必要です。
この機能はZIPCサンプルの中の「携帯電話」サンプルで理解できます。

②開始アクティビティー、終了アクティビティー

状態であればその状態に入る時やその状態から抜ける時、イベントであればやイベントに対して必ず実行する処理を定義することができます。
コーティングレベルではよくやることですが、それを設計書で定義出来るのは便利です。
同じような記述を何カ所にも入れる手間が省けるほか、状態やイベントを増やしたときに共通の処理を入れ忘れることを防止できます。例としては、ある状態から抜けるときは必ずタイマーを停止しなければならないのに、後から追加した状態に追加を忘れるケースなどが自然に防止できます。
意外に便利だったのが、この部分にデバック用の表示機能を埋め込むことで、状態遷移を外部で観察できるように出来たことです。
但し、あくまでもエディターの中ではオプション的な表示なので、この部分に処理を集中すると見落とす可能性も否定できません。使う場合は使い方をルール化することが重要だと思われます。

●重要なファクターは楽しいこと

その他にZIPCを使ってみて非常に重要なことに気が付きました。それは使う楽しさです（健康器具のTVショッピングのようで恐縮ですが…）。
最初は試行錯誤する部分もありますが、ZIPCプロジェクトの基礎ができあがり、手順が明確になると、その後は純粋にZIPCの情報だけに向き合うことが出来ます。
今までは設計書とプログラムの間を行ったり来たりが当然でしたが、ZIPCでは状態遷移表を修正して動作確認までがとても簡単に出来ます。２つのことが１回で済むのがなんとも言えず快感です。
CASEツールはエンジニアに対する負担が大きいので、現場から敬遠されがちなところがありますが、この楽しさがあれば多く受け入れられるのではないでしょうか。

●最新状況

試作への適用が終わり、現在では製品プロジェクトへの適用が始まっています。これも、全体から見ると一部のサブシステムへの適用ですが、コンパイラとの相性などを確認できました。
GNUコンパイラでZIPCの生成コメントを /* */ で生成するとコンパイルエラーになりますが、これは生成オプションで // コメントを選

択できるので難なく回避できています。
以下は、製品プロジェクトへ適用したエンジニアのコメントです。

良い点

①シミュレーションデバック（単体デバック）の効率が向上できた。
どの状態でも、自在にイベント発行ができるメリットが大きい。
②状態の増減やイベントの増減が容易に対応できた。
状態を追加した場合でも、遷移先は自動的に変わる。
さらに、ソースコードをわざわざ変更する必要なし。
③自動生成されたＣソースコードも比較的理解しやすい。
④状態やイベント毎に共通処理を指定できる。
（前述の、開始アクティビティー、終了アクティビティーです）
⑤日本語で状態遷移内の処理を記述できる
見た目で理解しやすい。

要望事項

①メイン関数をアプリから操作できるようにしてほしい。
既存システムに組み込み場合、隠れているメイン関数を若干修正する必要があったが、これをZIPCの管理下で自由に改造できると良い。
②マニュアルをもう少しわかりやすく且つ具体的にして欲しい。
マニュアルは日本語で詳しく書かれており、リファレンスとして全ての内容が書いてあるのは良いが、目的別に具体的な記述が多いものが用意されると更に良いと思う。

●操作性が格段に良くなりました

以前ZIPCを導入したときは「ZIPC ver.4.0w」でした。UNIX版から始まったZIPCがWindows版を提供して間もない頃です。その当時は、状態遷移表やシーケンスを専用ツールで作成、自動チェックできることに感動していましたが、現在のZIPCと比べると以下のような問題がありました。

①コードジェネレートで１つの状態遷移表を複数リソースで動かす生成が出来なくてコード生成を断念。
②STMエディターでアンドゥが１回も出来なくて辛かった。
③UNIXとWin3.1を継承しているので全体的にマンマシンI/Fが古いかった。

①は現在では「クローン機能」として実装されています。
②③に対しては現在のZIPCの操作性は飛躍的に向上しました。VCなどと同じような操作性になっているので最近の開発ツールに慣れたエンジニアであれば比較的楽に使いこなせると思います。

今も昔も変わらないのは、高解像度のモニターが欲しくなることです。図になったドキュメントはできるだけ広範囲を一度に見渡したいものですが、状態遷移表がちょっと大きくなると頻繁にスクロールが必要になります。フォントの変更や縮小は出来ても限度があります。しかし、これはZIPCに限ったことではなく、CASEツール全てが抱える問題です。だれか、画期的な発明をして欲しいところです。

●今後の課題と展望

当社はZIPCの導入フェーズから製品へ適用するフェーズに入りました。これからは、更に拡張した使い方を目指したいと思っています。ZIPCは開発工程の中流から下流をサポートしてくれますが、現在のソフトウェア開発では上流工程の問題も多くなっており、今後は要求の管理や分析の部分にも取り組んで行きたいと思います。これらについてもキャッツさんが取り組みを始めているようなので期待しています。

製品へ適用するにあたり一番の課題がZIPC以外のところにあります。それは製品への適用を開発依頼元に相談してもなかなか了承を得られないことです。その理由としては、

・納入品を改造する場合に必ずZIPCが必要になるが、どこでも持っているものではない。
・成果がまだ不明確。
・既に導入済みのツールがあるからそちらを使いなさい。

依頼元の立場から考えると仕方がないとも思えます。これらはZIPCの成果が少しずつ実証さ

れるなかで、理解が得られるものと考えています。

今後は少しでも適用範囲を広げ、実績を上げていきたいと思います。

●まとめ

今回、ZIPCの導入から試作システムへの適用、製品への一部適用を行いましたが、製品適用は開発中であるため、効果を数値的に表すことは出来ていません。しかし、コードジェネレーションまで行うことで、明らかにエンジニアの負担は軽減できることを確認しています。これは、工数の削減と品質の向上に繋がると言えます。

但し、全てが簡単ではなくZIPCで効果が大きい機能を有効に使いたい場合（階層化やコードジェネレーション）は、十分に研究する時間を見込む必要があります。最初は時間がかかるかも知れませんが、自分たちのセオリーを見つければその後は楽に開発を進めることが出来るようになります。

少し苦労はあっても、うまく導入ができれば厳しい開発のなかで楽しさを手に入れることが出来るかもしれません。

●最後に

私はキャッツさんのZIPCに対するアグレッシブな取り組みが好きです。また、世界に誇れる純国産のZIPCは私自身が多くの人に使って欲しいと思っています。

キャッツさんから「ZIPC WATCHERS に2回 原稿を出してくれる方はなかなかいません」と言われましたが、このつたない記事でも興味をもってくれる人がいれば幸いです。