# ZIPC　Fortune　～ next version 構想 ～

渡辺　政彦

## 1. はじめに

現在、ZIPC Ver.4.0w が市場に出ています。ZIPC ユーザの方はもちろん、これから導入しようかと迷っている方、皆さんこれからの　ZIPC がどのようになっていくのか、興味津々ではないかと思います。

本稿は技術論文スタイルではなくエッセイ風に仕立て、皆さんが気楽に読んでいただけるようにしたいと思います。

さて、次のバージョンの　ZIPC がどうなるかは、実は皆さんのご意見次第ということになります（イオイ、それじゃもうおしまいじゃないか。次の構想まで、それはユーザ責任ですなんてヒドイんじゃないの？）。

皆さんはキャッツの営業から耳にタコができる程「ZIPC は国産です」という台詞をお聞きになっているのではないでしょうか。ソフトウェア後進国、日本をもってして国産とは、一体何がセールストークなんだか。キャッツの営業が言いたいのは「ZIPC は日本で作りました。ですから日本のユーザ様のご意見を輸入モノの CASE とは異なり、聞き入れます」ということです。ですからドシドシご意見を下さい。　ZIPC は本気で皆さんの意見を聞きます。遠い国の xxx xxx ピー）が何か言ってるぞ、ではないのです。

## 2. 手法

次の ZIPC ではツールそのものの改良の他に、CASE ツールとしての屋台骨となる「手法」も改良されます。これは「拡張階層化状態遷移表設計手法－第2版－」として、実に5年ぶりに改訂されます（初版は1992年8月に出版され、キャッツから2万円で販売されています。皆さんからは高すぎるとのご批判をいただき、第2版はグッとお求めやすい価格になる予定です）。「拡張階層化状態遷移表設計手法－第2版－」では舌を噛み切りそうですし、この文字だけでやたら紙面を取ってしまいそうなので、以降は「新手法」と呼びます。新手法は　1997 年 1 月 20 日-21 日の 2日間日本テクノセンター主催の「階層化状態遷移表(ELSTM)設計手法による組み込みソフトウェアの設計・開発と生産性向上策」という長いタイトルで講演した際に、はじめてその概要を公表しました。新手法／旧手法を比較したものが**表1**です。

これをイチイチ説明していくと手法書が出来上がってしまいますので、ここでは代表的なものをサラっと紹介します。

**(1)駆動型**
駆動型をステートドリブンとすることで、プログラムで時間を作り込む、Cyclic Executive がしやすくなります。時間やサイズの問題で　RTOS を搭載できない組み込みソフトウェアでも　STM で設計し、現実的なコードが生成されます。

**(2)状態型**
従来、階層化でしか表現できなかった並列を、状態の型に並列属性を持たせることで、STM をスッキリ簡単にします。この並列状態型で最新ハレル型の STD(State Transition Diagram)で記述できることが STM でも記述できます。

(3)**遷移型**
親の状態に遷移させる際に子供の状態を遷移型で指定できることにより、子供の状態を意識したフラグ的な状態の増加を抑えることができます。

(4)**事象型**
割り込みを直接事象としてとらえることに成功しました。STM で設計中に割り込みの発生を意識して設計することができます。

(5)**アクティビティ**
アクティビティを実装することで、定期的な処理をワザワザ、アクションセルに記述しなくても良くなります。

(6)**STM プレーン**
デバイスドライバのユニット管理や、伝送制御のポート管理のように1つの　STM で複数の対象物を管理できます。

(7)**階層型**
従来の事象階層型はどうしてもプログラム構造をイベントドリブンにすることが望まれていました。状態の階層化により、プログラム構造をステートドリブンにすることができます。

(8)**STM 所属**
事象欄に関数コールの記述が可能となり、これにより、ミドルウェアのようなライブラリからデバイスドライバまで　STM で記述できます。

| | 新手法 | 旧手法 |
|---|---|---|
| ＳＴＭ駆動型 | イベントドリブン型<br>ステートドリブン型 | イベントドリブン型 |
| 状態方 | 排他型<br>並列型 | 排他型 |
| 遷移型 | デフォルト型<br>履歴型<br>深層履歴型 | 深層履歴型 |
| 事象型 | ポーリング型<br>同期型<br>通信型<br>割り込み型 | ポーリング型<br>同期型<br>通信型 |
| アクティビティ | スタートアクティビティ<br>エンドアクティビティ | なし |
| | モードアクティビティ<br>ディスパッチアクティビティ<br>イベント解析スタートアクティビティ<br>イベント解析エンドアクティビティ<br>イベント検出スタートアクティビティ<br>イベント解析スタートアクティビ | |
| ＳＴＭプレーン | 複数 | 単数 |
| 階層型 | 事象階層<br>状態階層 | 事象階層 |
| ＳＴＭ所属 | タスク部<br>モジュール部<br>ライブラリ部<br>デバイスドライバ部 | タスク部<br>モジュール部 |

**表１新手法と旧手法の違い**

**ここまで読んで、フムフムと理解できる方はいませんよね（ダメリマエだっ）。そこで、新手法の手法書を購入して下さい（イオイ、営業かよ）。次号で具体的な例をあげて説明したいと思います。**

## 3. ツール

**ある**ユーザさんがおっしゃいました。「今は手法を勉強してどうのこうのでなくツールが一体何をもたらしてくれるかが重要なんだよ。手法を勉強して、はじめてCASEツールが使えるなどとふんぞり返る時代ではなくCASEツールを使うとこれだけ簡単・便利気持ちイイんだから手法を勉強するかという時代なんです。手法のためにツールがあるのではなくツールがあってはじめて手法が生きるのです。いいコンパイラデバッグ環境があって、C言語となるようにです。ADAがイイぞ、SMALL TALKがやっぱりオブジェクト指向だよと言われても、学者じゃないんですから購入しやすい便利なツールがなければ、実務では使えません。

個々のツールの改良は、最初に述べましたように、ユーザ様からのご意見が反映されます。例えば、ZIPC SIMULATORの構造体ポインタのサポートiTRON対応、ZIPC EMUALTORの対応デバイスを増やすといったことです。本稿では新しく登場する（させたい）機能についてお話します。

ZIPC next versionの動作環境はwin32となり、ネイティブにWindows95, WindowsNT4.0で動作するアプリケーションとなります。この win32 化で新たに作成されるZIPC next versionの目玉は３つあります。

(1) STM EditorのActiveX対応
(2) VBとリンクしたビジュアルプロトタイピング リアルタイムシミュレーション
(3) ZOW(ZIPC On Windows)

### 3－1 ZIPC STM Editor next version

CASEツールは単なるお絵描きツールじゃない。この考え方をしてCASEツールを作成した結果ジェネレーション、シミュレーションできればドキュメントはOKという考えに支配されてしまいました。「CASEツールのエディタも立派なお絵描きツールじゃなきゃダメだ」というユーザや開発担当者の声に目から鱗でした。自動試験検証や効率的なプログラム生成といったことばかりに目を向けていたのです。しかし、設計書はそれ自体立派なドキュメントであり、それを様々な形態で運用できるようにすべきだったのです。

そして、win32 化するにあたり、目玉の1つとしてZIPC STM Editorの立派なお絵描きツール化が決定したのです。立派なお絵描きツールとするために、ユーザ様からの貴重な意見を取り入れる他 ZIPC STM EditorをActiveX対応とすることが決定しました。

Windows95,Windows NTとインターネットイントラネットの急速な普及を無視することはできないと判断したからです。HTML化についても検討はされましたが、ネイティブフォーマット以外に同じ情報を含む2つのファイルが存在し、それぞれの情報が同期しない可能性の問題を重要視し、かつ、現在のHTMLでは複雑なグラフィック情報の取り扱いが不得手であることから、ActiveX対応に決定しました。STM EditorがActiveX対応になるということは、状態遷移表を4つの運用(実行)環境をユーザは手に入れることになります。

- 単独に実行される ZIPC STM Editor
- OLE2対応アプリケーション(Word等)で実行される ZIPC STM Editor
- Office95バインダーで実行される ZIPC STM Editor
- ActiveX対応インターネットビューア(インターネットエクスプローラ等)で実行される ZIPC STM Editor

注)ここらへんの話は、プレンティホール出版カウレンテーンスト:「ActiveXを知る」に詳しく載っています。

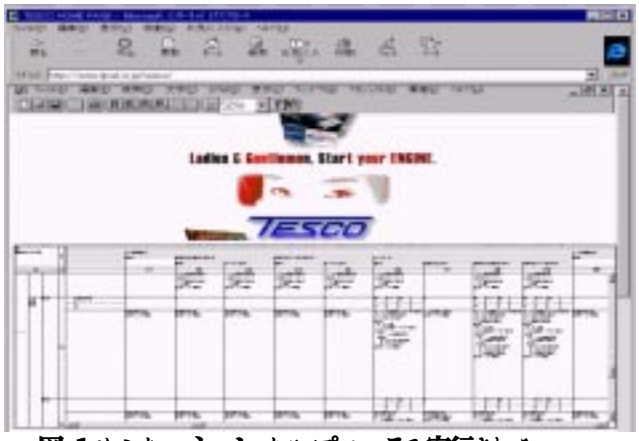

図1インターネットエクスプローラで実行される ZIPC STM Editor

### 3−2 ZIPC Front Panel Emulator new version

ZIPC Front Panel Emulator(以降ZFPEと呼ぶ)は、Visual Basicにより機器やシステムの外観図を作成し、その動作をSTMで記述します。記述されたSTMはZIPC Simulatorによりシミュレーションされることで、ビジュアルプロトタイピングが行なえます。しかも、このSTMは設計書であり、ここからCのソースコードが生成されます。しかし単にビジュアルプロトタイピングをしても、既にこのような製品は世の中には存在します。ZIPCならではを考えたところ、リアルタイム集時間をビジュアルプロトタイピングに取り込むことが検討されました。あるユーザ様にこの考えを相談すると、奇しくも同じような考えを持っておられ、よっしゃ、行こうと勇気づけられました。図2のようなプリペイドカード発行機があったとします。

このような外観をVBで描いて、シミュレーションすることは訳もないことです。しかしこの内部は図3のようになります。各モータの動作時間の経緯によりセンサーが反応し、この[illegible]ます。


図2 プリペイドカード発行機の外観図

この時間タイミングを記述したものが図4のタイミングチャート(以降TCと呼ぶ)です。このタイミングを与えて図5のようなSTMが外観図と一緒に動作することで、ビジュアルプロトタイプリアルタイムシミュレーションが実現できます。時間の生成はZIPC Simulatorで行われます。このため、ZIPC Simulatorは従来のインストラクション時間単位の他にユーザが設定できる実時間情報を処理するようになります。

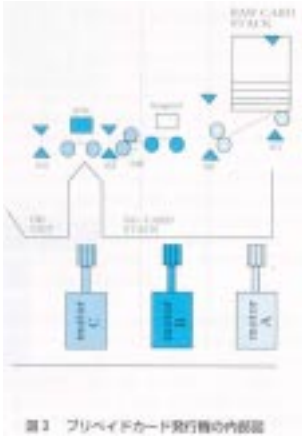

図3 プリペイドカード発行機の内部図

### 3−3 ZIPC On Windows new version

ZIPC On Windows(以降ZOWと呼ぶ)は、ZIPC Generatorが自動生成したCのソースコードをVC++を使ってユーザが用意します)Windows上でネイティブに動作させてしまおうというものです。この際先ほどのZFPEと連動するようにします。これで、チンタラ遅いシミュレーションを何千、何万倍も速いWindowsネイティブなスピードで実行してしまおうというものです。例えば、先ほどのプリペイドカード発行機のタッチパネルの画面制御処理などはリアルタイムなデバッグは必要としないのです。画面のモードが上手くいっているか、そして内部メカ制御と連動できればいいのです。画面制御にもSTMは非常に有効です。小さな画面に必要な情報を出すためにはたくさんのモード(状態)を管理する必要があり、さらにユーザフレンドリなインターフェースのために前画面に戻ったりすること(状態遷移)が必要になります。図6は、プリペイドカード発行機の画面

一覧です。

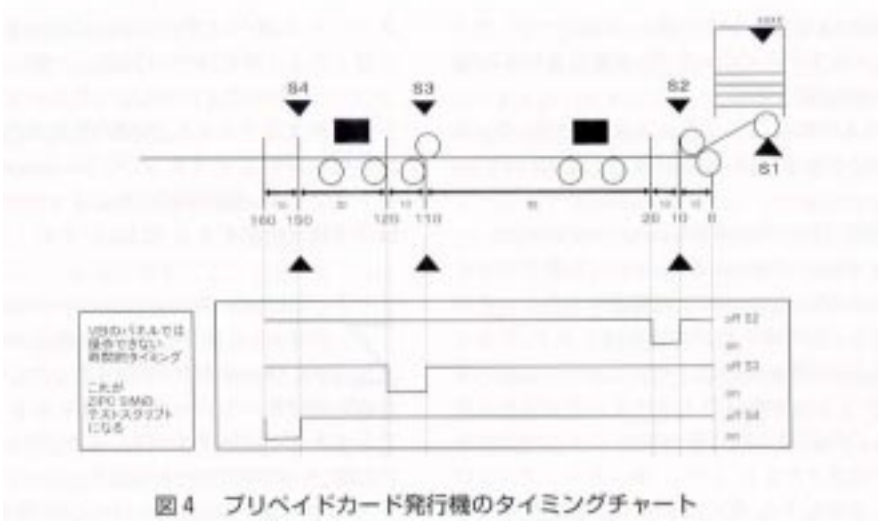


図4　プリペイドカード発行機のタイミングチャート

図5　プリペイドカード発行機のＳＴＭ

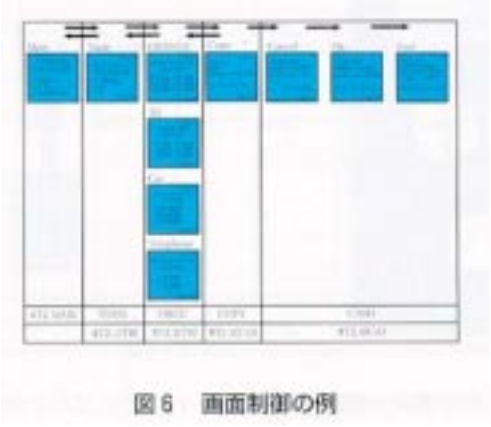

図6　画面制御の例

現在考えられているZOWには弱点があります。例えばI/O部分に関してはZFPEとの連携により動作はできますが、RTOS機構はWindows上ネイティブに動作できないため、マルチタスク機構システムコール等は全滅です。割り込みの機構も対応はとれません。Windowsでネイティブに動作するRTOSの供給がないと技術的なブレイクスルーはできないでしょう。RTOSのシステムコールは使えませんが、ZOWからZIPC Simulatorに対してイベントは発行できます)。

## 4. さらなる未来

3章でお話したnext versionは1年後に実現を目指しているものです。これからお話するのは3年後に実現を目指すものです。1つはペリフェラルデバイスのシミュレーション、もう1つはオブジェクト指向との融合です。

### 4−1 ペリフェラルデバイスシミュレーション

ASICやLSI等をレジスタレベルでSTMを記述します。ここで言うレジスタとはコマンドレジスタやステータスレジスタのことです。これにより、実機を待たずにシミュレーションが行なえるようになります。EDAなどのVHDLやVerilogといったシミュレーションよりも現実的な速度でソフトウェア設計シミュレーションが行なえるようにしたいのです。プログラムとペリフェラルデバイスの接点はペリフェラルデバイスのレジスタであり、RAMがACKを返すタイミングの検証はプログラムの検証レベルではありません。このようなレベルはハードウェアがEDAを使って検証すればイイのです。ソフトウェアはペリフェラルデバイスが反応する時間とCPUの処理時間から、システムとしての実時間を満足できるものかを検証し、論理的にペリフェラルデバイスを使えればイイのですから。

MCU,ASIC,LSIのメーカの皆さん、分厚いマニュアルを提供するのではなく設計書シミュレーションができるSTMをユーザに供給することを検討してみてください。ユーザの皆さんサンプルデバイスができるのを待っている間に、デバイスSTMでシミュレーションすることについてどう思われますか。

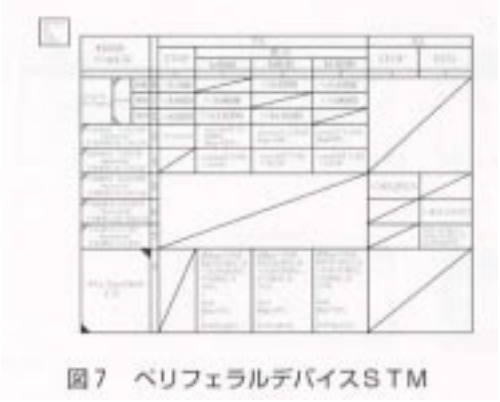

図7 ペリフェラルデバイスSTM

### 4−2 オブジェクト指向との融合

ZIPCはシステム制御に焦点を絞っています。1990年初期には「状態遷移」から「実時間+オブジェクト指向」へと移行するようになっています。ZIPCも1990年終わりごろに移行した製品を供給できると考えています。ZIPC next-versionではZIPC Ver.4.0wと比べると、TCの支援、ZIPC Simulatorの実時間管理により、実時間への対応が強化されます。あとはオブジェクト指向へのアプローチです。オブジェクト指向へ移行するには、キチっとしたオブジェクト指向言語コンパイラが必須です。設計分析方法論だけではダメです。C言語の普及と同じ努力が必要です。

制御系プログラマがC言語に飛びついた最大の理由は移植性にあると私は考えます。そしてオブジェクト指向言語(例えばC++)に制御系エンジニアが飛びつく最大の理由は再利用性にあるのではないでしょうか。この再利用はクラス継承による差分プログラミングにより実現されます。分析設計レベルでなく、言語レベルでソフトウェアを部品化なくてはなりません。それでは設計では何をするかといえば、設計で実時間をこしらえていくのです。言語に実時間をこしらえる能力はありません(ADAならできるかもしれません)。アセンブラ言語、C言語、C++言語を使おうとも、それは処理スピードの問題であって、時限内に反応を保証する実時間とは別の次元です。

ZIPC next versionでは、事象欄に関数を記述できます。さらに割り込みを事象欄に記述することもできます。これらの機能により、STMでクラス設計が行なえます。図10がSTMでクラス設計をした場合の例になります。後は、VBで外観図を作成し、デバイスをSTM化し、ZIPC Simulatorにより実時間シミュレーションを行います。実時間の保証がとれれば、後はZIPC GeneratorでC++を自動生成するだけです。オブジェクト指向言語コンパイラやRTOSのオブジェクト指向化といった問題があります。この点については今後、各分野のスペシャリストの方々と楽しい議論をしたいと思っています。

## 5. ロードマップ

図9にZIPC製品のロードマップを示します。 ZIPC Jupiterとは ZIPC Ver.5.0wの開発コードネイムです。ZIPC Jupiterは win32で動作します。 ZIPC零は ZIPC Jupiter, ZIPC Ver.5.0wの STM Editorと同一のものです。早めにSTM Editorのモニタ版を提供させていただき、製品版のリリースまでにいろいろと皆さんからの意見を取り込もうと考えています。 ZIPC Ver.5.0wは英語バージョンも作成し、海外への販売も開始する考えです（まごッチに先をこされてしまった）。世紀末には レジスタシミュレーション、オブジェクト指向へのブレークスルーを目指す巨人の神"Saturn"が現れるでしょう（これがノストラダムスの予言の1つの解釈です）。

98年以降は、現在のネットスケープナビゲータ等のように、インターネットによるソフトウェアの供給となり、毎日がアップデートといった形態になるでしょう。また、皆さんが作成した VB部品やSTM,デバイス STMのマーケットも考えています。 作ったらまずは会社の資産として社外秘にして、新しいものができたら古いものを市場に供給し、開発費の補充にする、こんなことができたらイイなと思いませんか。

## 6. おしまいに

携帯電話やPHSにタマゴっちのような電子ペット機能を付けたら結構売れると思います。メーカの方どうでしょう？そのときは、開発環境にZIPCを使って下さい。

渡辺 政彦（わたなべ まさひこ）

図8 クラス設計ＳＴＭ

図9 ロードマップ