計測器の組込みソフトウェア開発への ZIPC 適用事例

コニカミノルタセンシング株式会社　開発部開発 3 課

久保 明

1. 導入の背景

導入の理由としては、以下の 3 点である。

- ソフトウェアの大規模化、短納期化への対応
- ソースコードとドキュメントを一致させる
- 仕様変更への短期間での対応

上記の解決手段を模索していたところ、マネージャから ZIPC の紹介があったため、導入するか評価を行った。

2. ZIPC 選定理由

ZIPC 選定理由は、以下の 3 点である。

2.1　ZIPC 選定理由

- ツールの特徴
  多彩なオプションがあり、ソースコードを思い通りに出力できる。
- プロジェクトメンバーの適性
  プロジェクトメンバー全員が STM（State Transition Matrix：状態遷移表）設計の経験者であった。
- グループ内での導入実績
  グループ内で、導入実績があり、マネージャが導入の価値ありと判断。

2.2　選定にあたっての不安点と対策

選定にあたり、以下の不安点もあったが、対策も同時に考えることで導入を進めることにした。

- ツールの使い方の習得に時間がかかる
  サポートの有効活用。
- 効果が出ない
  小さな範囲で導入し、効果を早く確認する。
- ツール自体に不具合がある
  ツールの標準的使い方をする。

3. 実施内容

3.1　導入計画

まずは、影響範囲が小さく、ソースコード規模も小さいキー認識部分で適用し、導入効果を確認する。その結果をふまえ、導入可能と判断したら、影響範囲が大きく、ソースコード規模も大きい画面遷移部分に適用する。工夫点としては、変更時にソースコードが大きく変わってしま

わないために、状態とイベントにはそれぞれ、名前をつけるようにし、状態、イベントが追加、削除された場合にもソースコードが大きく変わらないようにした（図 1参照）。あまり多くの機能は使わないようにし、メンテナンスがしやすいようにした。

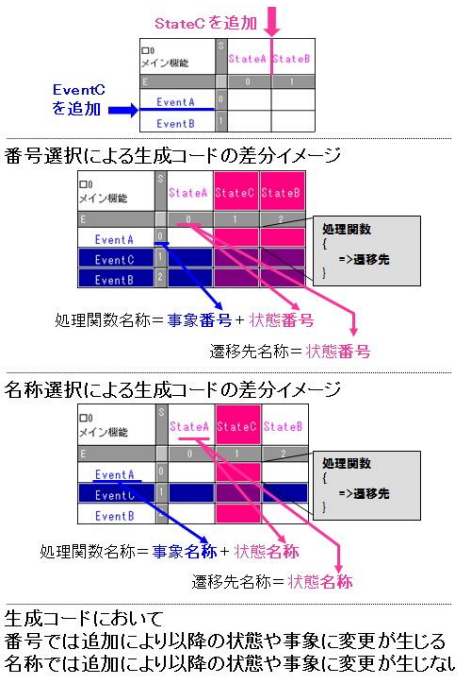

図1　状態・事象の追加によるコード変更

3.2　キー認識

図 2に示すように、STM の作成からシミュレーションでのテストまでを ZIPC にて行った。シミュレーションで確認を行うため、できる限り実機に近い状態で行うことにした。STM の起動条件を、ポートの割り込みで、その後の動作を周期ハンドラで行うことにし、OS 動作も含めてシミュレーションした。

シミュレーションでのテストに当初 ATV（Auto Test and Verification）を使用する計画であったが、今回の条件では回帰テストが行えないことがわかり、サポートに相談の上、バッチファイルを作成して対応した。

実機での確認では、所望の信号を作成することができず、ソースコードのカバレッジをとることができていなかったが、シミュレーション上でテストを行うことにより、カバレッジをとることができ、設計通りの動作をしていることが確認できた。

ただし、キー認識などのように、人間の感覚によるものについては、シミュレーションだけでは、完結せず実機を用いて人の確認を行い、微調整を行った。

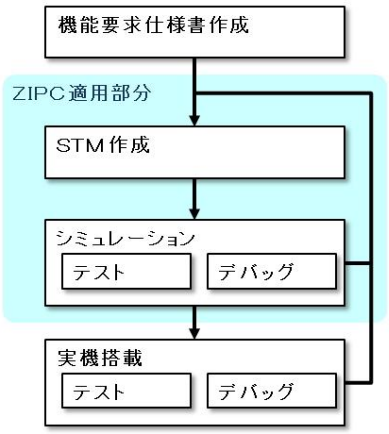

図2　キー認識部分でのプロセス

3.3　画面遷移

画面遷移については、サポートと相談し、S 型を選択し、テストは実機で行うこととした（図 3参照）※（詳細は後述）。画面遷移図が存在し、画面数は 100 以上になることがわかっていたため、階層化を行うこととした。階層化は、画面遷移図レベルで表記できるものを、第 1 階層に、

それ以外を階層化することとした。工夫点は、STM に画面描画処理を記載されて煩雑になるのを防ぐために、開始アクティビティに画面描画処理を記載したことである。

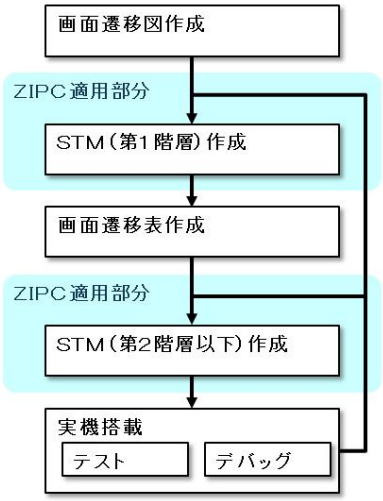

図３　画面遷移のプロセス

※S 型の選択理由

- 画面＝状態としたため、状態の階層が深くなる。
- S 型は階層間の状態に依存があり、表現しやすい。
- E 型は階層間の状態に依存がなく、グローバル遷移を多用することになる。

S 型選択時の注意点：

- 階層の規模が大きくなりすぎると、1 ファイルの生成コード規模がターゲットデバッガによっては許容量を超える可能性がある。

実施時に感じたことは、以下の通り。

- 担当者が STM 設計経験者であり、STM 自体を理解していると導入はスムーズ
- ZIPC の機能として、STM→コード生成は、理解しやすい
- シミュレーションをしようとすると、ツールの習熟が必要

4. 結果

4.1　導入効果

以下の効果があった。

1）工数削減

2）コードとドキュメントの一致

3）シミュレーション環境での検証

各効果についての考察は以下の通り。

1）については、ZIPC を導入しないことを前提に作成した計画（設計・コーディング期間）に対して、導入初期のキー認識は約 75%の増（初期導入も含む）、画面遷移部分は、約 40%削減した。

キー認識で計画より時間がかかったことに関しては、後述する（反省点参照）

画面遷移部分での削減については、ツールに習熟してきていたこと、使う機能を限定したこと。（シミュレーションでの検証は行わなかった）によるものと考えている。

トータルでは、12%程度の削減となった。

これは画面遷移部分のコードの規模が大きいことにより、全体としては、削減できたことを表している。

2）については、ZIPC 導入への期待する効果が得られたと想定する。仕様変更時のドキュメントのメンテナンスが不要になった。※ZIPC の設計書を GUI 仕様書として利用す

ることまではできず、解決しなかったこと参照

メンバー同士の確認も、コードではなく、STM ベースで行うことができ、効果的であった。

3）については、今までは、実機での動作とコードレビューにより検証を行っていたが、動作が可視化され、カバレッジもとることができ、検証を効率的に行え、不具合が起きなかった。何より設計者が設計に自信をもてることも大きいと思われる。

4.2 解決しなかったこと

1）STM 設計レベル向上への寄与
2）GUI 仕様書を ZIPC の設計書にて記載

各項目に関する考察は、以下の通り。

1）に関しては、ZIPC を導入し効率化した時間の一部を用いて、スキルアップをすることが大切であると考える。STM 設計に限った話ではないが、ソフトウェアの世界は変化が激しく、スキルアップの時間は必要であると考える。STM の複雑度やモレ、抜けが判定できる機能があれば、設計能力向上に寄与できると思われる。

2）に関しては、ZIPC の状態遷移表をうまくエクセルなどにエクスポートする機能があるか、もしくは各画面とリンクできるような機能が良いように思う。今後の機能向上に期待する。

4.3 反省点

導入初期のキー認識において、導入計画に比べ、多くの機能を使おうとしすぎ、作業に時間がかかった点が反省点としてあげられる。具体的には、キー認識の次に行う計画だった画面遷移も含めてひとつのプロジェクトにできるように、以下を行った。

1）割り込みのシミュレート
2）周期ハンドラのシミュレート

1）、2）により、ATV でのテストができなくなった。

割り込みや周期ハンドラは、自前でコーディングして ZIPC のイベントを周期ハンドラが起こすようにしておけば、ZIPC としての、基本的機能だけで済んだため、最初の検証が、より早く行うことができたと思われる。

5. 今後の計画

1）他のプロジェクトへの展開

現在取り組み中ではあるが、社内のプロジェクトに水平展開していきたい。

2）Garakabu2 を使った事前検証

現在、試験適用中である。画面仕様決定時に、ツールで検証ができれば、仕様作成時間が短縮され、よりよいものができると想定している。

6. 導入のポイント

以下が導入のポイントであると想定する。

1）導入を目的としない
2）Small Start
3）サポートの活用

特に、3）は、ツールに不具合がある場合もあり、早めに相談することが大切であると感じる。

今後、導入される方の参考になれば幸いである。