カラーレーザープリンタへの ZIPC の適用

富士ゼロックスプリンティングシステムズ株式会社
商品開発統括部 エンジンコントローラー開発部

小野 明則

1．導入の背景

カラーレーザープリンタの制御システムは、PC から受けたデータを展開するコントローラと用紙を搬送し画像形成を行うエンジン制御部分に分けられます。

私が担当していて、ZIPC を適用したのは、画像形成を行うエンジン制御部分です。より具体的には用紙を搬送してジャム（紙詰まり）の検知、トナーを用紙に転写する電圧制御、トナーを用紙に定着する定着器の温度制御、用紙有無等のセンサ入力の処理を行っています。現在の組み込みソフトウェアの開発は低コスト、短納期、高品質が求められています。弊社で開発しているレーザープリンタは高電圧でトナーを用紙に転写したり、高温でトナーを溶かして用紙に定着させるなど「やってみなければわからない」ことが多く仕様変更への対応スピードや二次不具合を出さないことも要求されます。近年はカラーレーザープリンタへの要求が増大していて、プリンタ制御プログラムにおいては、各色ごとの色ずれ防止や微妙な色合いを再現させるための処理などが必要となりシステムが更に複雑なものになっています。

従来の開発現場では処理にヌケがあると用紙搬送モータが回りっぱなしになったり、修正の二次不具合でジャム（紙詰まり）を誤検知したりする問題が発生していました。この原因はシステムが複雑になるにつれて、処理にヌケ、モレが増えるためと考えました。今まではフラグによる条件分岐を多用していたため、処理の流れが非常にわかりづらいことや、担当者個人のスキルによってコードの品質に差が大きい事が問題となっていました。そこでフラグ型から状態遷移型にすることにより処理を簡単化してバグを減らし、個人のスキルによらずに高品質なコードが短時間で作成できる設計手法を目指しました。

2．導入過程

まず、無料の ZIPC 製品紹介セミナーを受講しました。このセミナーを受講して ZIPC の導入を決定したのですが、理由は以下のとおりです。

- 状態遷移型にすることにより従来のフラグ型プログラムから脱却できる

- 状態遷移表を使用することによりヌケ、モレを防止できる
- 階層化により表が大きくなりすぎることを防止できる
- リアルタイム OS が MUST ではないため、適応するシステムに応じてユーザが選択できる

次に ZIPC を利用するのは初めてだったため実習セミナーを受講しましたが、その時は偶然にも一人で受講する事ができました。テキストは「拡張階層型 状態遷移表 設計手法」のポイントがわかりやすく解説されていてとてもためになりました。ZIPC の使用経験がなかったので状態、イベントから設計例まで詳しく教えていただき理解を深める事が出来ました。マンツーマンだったので担当するシステムについてどのような構成にすればよいかのアドバイスも頂きました。

3. 導入の効果

ZIPC の適用の対象は既に市場導入されているカラーレーザープリンタの増速機でした。

理由としては基本仕様が既に決まっているので、印字スピード以外は全く同じ動作にすることを目標にできるからです。ただし、処理速度の向上のため CPU は前任機が 8 ビットであったのに対して、今回は 16 ビットを採用しました。適用範囲については状態遷移型の制御が適用可能な部分に関しては全て ZIPC によるコードの自動生成を用いました。関数についても .fnc ファイルを作成しましたのでコードの 95% 以上は自動生成しています。

ZIPC 適用の結果について **表 1** に示します。コードサイズに関しては前任機を 1 とすると、ZIPC を適用した増速機は約 1.4 倍となりました。ただし、処理の修正や未導入機能があり、厳密なサイズ比較は出来ませんでした。コード増加の理由としては ZIPC の適用によるコード増加が +20%、16 ビット CPU 採用によるコード増加が +20% 程度であると予想されます。ZIPC 適用前はもっとコードサイズが増加すると予想していた点と、適用時はある程度サイズの増加を見積らなければならないことから評価は △ としました。

バグの量についてですが、ある期間のバグ件数について比較を行いました。ただし、期間や開発段階が全く同じではないので参考程度にお考えください。それぞれのバグを発生原因に応じて 3 種類に分類しました。

● 構造的問題によるバグ

プログラムの作り方に問題があり、処理にヌケ、モレが存在した。

ZIPC の適用によって構造的問題によるバグを約 1/7 に激減させることに成功しました。これは処理をできるだけシーケンシャルに行うような状態遷移に作

ったためです。ある条件を満たさない限り次の状態に遷移する事はないので、チェックをしながら確実にシーケンスを進める事ができます。まれに条件を満たさず次の状態が遷移できない状況が発生したとしても、状態変数を参照することにより不具合の原因を簡単に特定する事ができました。

仕様変更に対しても、時間的な制約があまりない場合は状態を追加する方法で対応したため、他の処理への影響を最低限に抑えることができて二次不具合を大きく減らす事ができました。

また、ある状態に遷移する時に共通の処理を行いたい場合、今までは処理を関数呼び出しにしていたため呼び出しを忘れたり想定外のシーケンスで状態への遷移が発生したりすると、用紙搬送モータなどへの出力が停止しないといった不具合が発生していましたが、状態のスタートアクティビティとして処理を登録することによりヌケを防止することができました。

以上のことから ZIPC を適用すると安定したソフトウェアを開発初期から作成できる事がわかりました。

● 設計ミスによるバグ

要求仕様を十分検討しなかった結果、想定外の事象に対して正しく動作できなかった。

次に設計ミスによるバグの数についてはほとんど変化が見られませんでした。仕様変更の要求があった場合に、十分な検討をせずにコーディングを行ってしまったことが原因と考えられます。TOP DOWN での設計において仕様の変更があった場合は、最上位からチェックしなおす必要があり、場当たり的に修正を行ってしまってはバグを減らすことはできません。

● 確認不足によるバグ

コーディングにおける問題が、確認によって発見できなかった。

確認不足によるバグは逆に増えてしまいましたが、原因として ZIPC に不慣れだったので従来手法に比べケアレスミスが多かったということが挙げられます。さらに今回はシミュレーションを行わなかったので、ある特定のパターンでしか確認をしていなかったことがこのバグを防止できなかった原因だと思います。設計者の ZIPC への理解度が増すこととシミュレーションの導入により減らせると思います。

表 1　ZIPC 適用の結果

	前任機	増速機（ZIPC 適用）	判定
コードサイズ	1	前任機の約 1.4 倍 ×1.2(ZIPC によるコード増加) ×1.2(16 ビット化による増加)	△
構造的問題によるバグ	37	5	◎
設計ミスによるバグ	32	31	△
確認不足によるバグ	15	39	×

4. 改善要求

ZIPC を使用していく中でいくつか気がついた点がありますので、今後改善を検討していただきたいと思います。

- **STM 自動検証のためのシミュレーション機能の充実**
  単体の STM において変数、イベントを変更して網羅的な試験を行えるようにしていただきたいと思います。そしてテストを行った後の変数や状態が設計通りであるかをチェックできる機能があれば、単体としての完成度が上がり全体としての品質も向上すると考えられます。
- **STD において異常系の処理を表示させない/印刷させないオプション**
  異常を検知した時に全ての状態からエラーの状態に遷移させるようにすると、非常に見にくい STD になってしまいます。また、状態の遷移を確認したい場合には、イベントや処理を表示させてしまうと見にくくなる場合があります。遷移やイベント、処理については表示・印刷させる/させないオプションがあると、非常に便利だと思います。
- **ひとつの STM 内でのイベント解析設定を変更したい**
  状態が遷移していく中で IF-IF でイベントを待ちたい場合と、IF-ELSE で待ちたい場合が混在する時があります。どちらかに統一して設計した方が良いと思いますが、切り換えられた方が便利な場合が存在します。
- **コード生成においてサイズ優先/スピード優先を切り換えたい**
  システムの制約からできるだけ処理/遷移を早くしたい場合と、ROM 容量の制約からサイズをできるだけ小さくしたい場合があります。生成のオプションを追加していただけると目的に応じてコードを生成できるため便利だと思います。

5. 感想

今までの設計においても状態遷移という考え方はあったのですが、今回 ZIPC を使用させていただきシステムの状態遷移を強く意識して設計を行いました。その結果、開発初期段階から大きなバグを減らす事が出来ました。さらに開発期間が短い中で新しいツールを導入する事にしたので不安はあったのですが、納期通り市場導入する事ができました。これはツール及び設計方法がわかりやすかったためで、ツールを導入する場合の難易度は比較的低いと思いました。さらにソースコードレビューを実施した時に、従来はコードの流れを追う程度でイベントの発生タイミングが変わっても問題ないかということに関しては作成者だけしかわからないという状況でしたが、状態遷移表は直感的に理解できるため処理の流れが把握しやすくレビューの質を向上させることができました。

しかし、ツールに不慣れだったせいで、意図通りに動作しないことがあったり小さい不具合は逆に多くなったりしました。STM の作り方にはコツが必要で数回作ってはじめて品質の高い設計ができるようになるようです。さらにチームで開発を進める場合は、ある程度状態の分割方法やイベントの待ち方を統一しておくと、品質の高いコードを作成できるだけでなくレビューにおいて相互のチェックを行う場合にも理解しやすいため、共通の基盤は必要なようです。

今回はツールの効果を判定するため、全てのシステムを ZIPC のコード自動生成で実現させました。しかし、コードサイズや処理のスピード、検証の容易さなどを考えると改善の余地があると思われます。特にコードサイズの増大はコスト増加につながる懸念があるため、ZIPC を適応すべき部分は選択したほうが良いと感じました。

また、今回の活動においては時間の制約からシミュレーション活用できなかったのですが、状態遷移表による設計はヌケ、モレを防止する効果はあるものの各処理の誤りを減らす事は出来ません。システム全体の品質を上げるためには、STM 単体としての品質を上げていく事が不可欠でシミュレーションをうまく活用していくことがキーになると思います。

最後に ZIPC 導入のためにお忙しい中サポートをしていただいた CATS（株）の皆様にこの場をお借りしてお礼を申し上げます。