プログラム・シーケンス・ジェネレータを用いたソフトウェアの開発

2N－2

鈴木 大作、 藤森 秀樹、 花浦 敏孝
(株) 松下電器情報システム広島研究所

1. はじめに

近年、コンピュータ技術、ソフトウェア技術などの発展にともない、コンピュータに搭載するソフトウェアが大規模化、複雑化してきており、ソフトウェア構造化技法を用いた設計手法が重要視されるようになってきた。

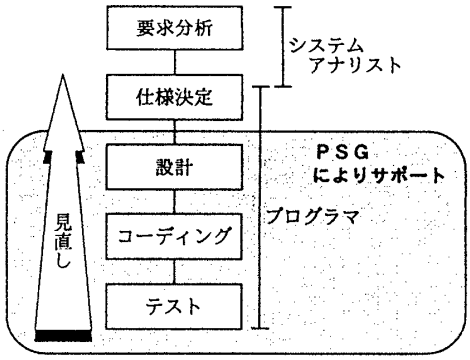
ソフトウェア開発は、上記のような設計手法を用いて綿密な設計を行なったとしても開発/テスト段階でプログラム仕様、プログラム・シーケンスなどの変更が頻繁に発生し、ソフトウェア開発者はこれらの変更に対して柔軟に対処しなければならない。また、これらの変更に対処していくことで開発対象となるソフトウェアが進化していく。

しかし、これらの変更に対して全て対応するということは膨大な開発時間が必要となることが多く、開発期間から考えて全てに対応せず、必要最小限の変更にとどめていくことが現状である。

そこで、このような課題を解決するために、我々は状態遷移設計手法に着目し、状態遷移図の作成、プログラム・シーケンスの設定/変更をコンピュータ上で視覚的に行ない、プログラム仕様、プログラム・シーケンスなどの変更に柔軟に対処する「プログラム・シーケンス・ジェネレータ」(以下ＰＳＧと略す) を開発したので報告する。

2. 開発工程におけるＰＳＧの役割

ソフトウェア開発における一般的なライフサイクルは、第１図に示すように「要求分析」、「仕様決定」、「設計」、「コーディング」、「テスト」に分けられ、要求分析を行なうシステムアナリスト、及び仕様決定からテストまでを行なうプログラマが思考錯誤しながら手作業で開発を行なっている。



第１図 ソフトウェア開発のライフサイクル

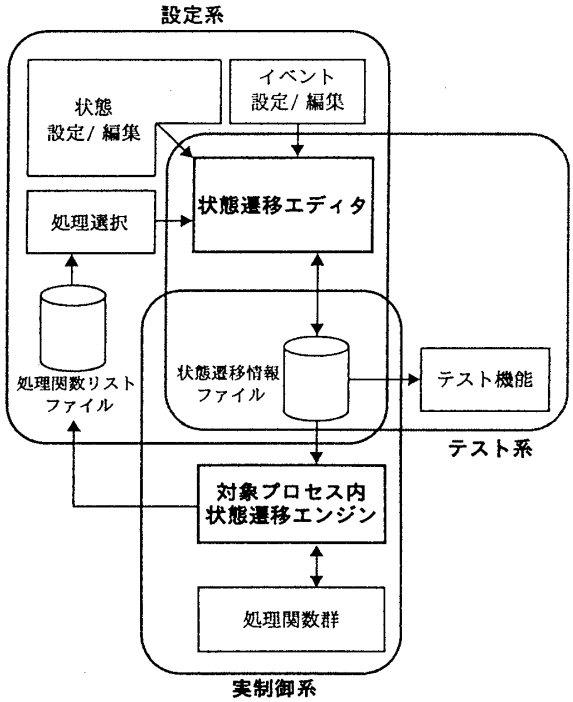
しかし、設計段階では潜在的な間違いや洩れが埋もれやすく、コーディング、テスト段階ではじめてこれらの潜在バグが発見され、前工程を見直す場合が多い。潜在バグは、早期工程のものほど後期工程に深刻な影響を及ぼし、理想的には早期工程の潜在バグは少ないほうが良い。しかし、現実的には潜在バグを少なくする決定的な有効手段はなく、最終的には人間の経験に頼らざるを得ないのが現状である。

よって、我々が提案するＰＳＧを用いたソフトウェア開発では、設計からテストまでの見直し作業により発生するプログラム・シーケンスの変更、及びテスト、また状態遷移図等のドキュメントの修正作業を効率的に行なうことを役割とする。

3. 構成と概要

ＰＳＧの構成を第２図に示す。

ＰＳＧを大別すると"設定系"、"実制御系"、"テスト系"の3つで構成され、各々が関連を持ちながら開発の効率化を図っていく。



第２図 ＰＳＧの構成

3.1 設定系

設定系では以下の機能を有し、それぞれ状態遷移エディタを使用し実現される。

Software Development with Program Sequence Generator
Taisaku Suzuki, Hideki Fujimori, Toshitaka Hanaura
MATSUSHITA Information Systems Research Laboratory
HIROSHIMA Co.,Ltd.

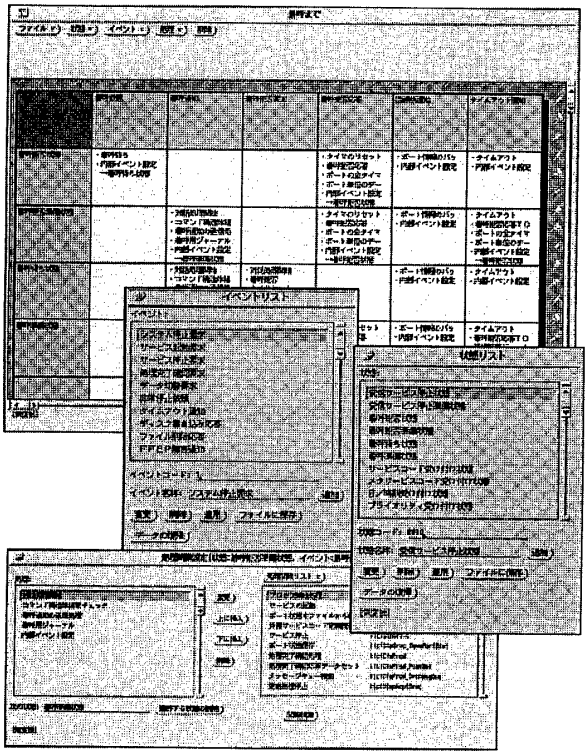
(1) プロセスが受け付け可能なイベントの設定/ 編集
(2) プロセスが取り得る状態の設定/ 編集
(3) 任意の状態、イベント時における処理の選択
(4) 上記 (1)(2) に対応したプログラム・シーケンスを作成し状態遷移情報ファイルに保存する。

(3) で選択される処理は、対象プロセス起動時に状態遷移エンジン (後述) が作成する処理関数リストファイルを参照する。

状態遷移図の作成例を第 3 図に示す。

状態遷移図では対象プロセスの任意の状態、イベントを見出しに設定し、状態とイベントにより一意に定まる事象に処理を設定する。その際、処理は複数設定可能であり、設定された順に従って、プロセス状態、イベントに対応した処理の流れが決定される。また、各状態、イベントに対応した処理群の最後に次に遷移する状態を設定することにより、対象プロセスの状態の遷移を実現する。

以上のように、プロセス内で想定される全ての状態、イベントに対応させて処理手順を選択していくことで、対象プロセスのプログラム・シーケンスが構築される。



第 3 図 状態遷移図例

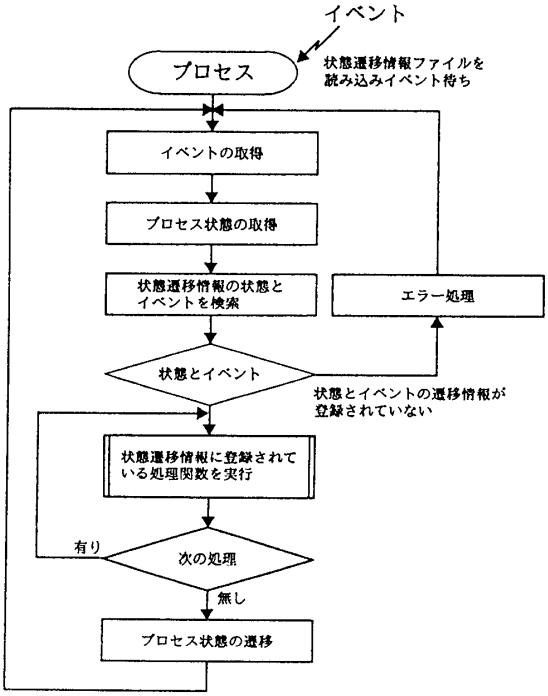
3.2 実制御系

実制御系では、対象プロセス内に組み込まれた状態遷移エンジンで対象プロセスのプログラム・シーケンスの制御を行なう。制御に必要な情報は、状態遷移エディタで作成された状態遷移情報ファイルを介し取得する。

状態遷移エンジンのフローを第 4 図に示す。

まず状態遷移情報ファイルを読み込み、メッセージループで外部プロセスからのイベントの発生を待つ。外部プロセスからのイベントを受け取ると、イベントと対象プロセスの状態をキーに状態遷移情報を検索し、該当処理の有無を調べる。該当処理が存在する場合、処理関数を登録順に実行し、全処理関数実行後に設定されている次遷移状態を現在のプロセス状態とする。

該当処理が存在しない場合はエラー処理を行ない、次のイベントの発生を待つ。



第 4 図 状態遷移エンジンフロー

3.3 テスト系

テスト系では、作成した状態遷移情報をもとに、対象プロセスの状態遷移テストを行なう機能を提供し、単純ミスなどの削減を可能とする。

4. まとめと今後の課題

今回、対象となったプロセスの開発では、対象プロセスのモジュール設計から処理関数の実装部の作成に至るまでを新規に行なう必要があった。その為、ＰＳＧの適用による開発効率の向上は、処理関数がある程度作成された開発終盤に図ることができた。

対象プロセスの開発に際し、ＰＳＧを使用した開発方法に適合して作成された処理関数を蓄積/ 再利用することにより、今回の開発により得られた開発効率の向上よりもさらに多くの効果が期待できると思われる。

今後の課題としては、以下の点が挙げられる。

(1) 設定系において、任意の状態、イベントに対する各処理関数の復帰値ごとに処理の振り分けが行なえるようにする。
(2) テスト系において、各処理関数の実行を可能とすることによりテスト環境の充実を図る。