金融機器組込みソフトウェア向けCASE用ジェネレータ*

3K－6

玉木 裕二, 藤巻 昇, 清水 洋子, 岡安 二郎, 小尾 俊之†
株式会社 東芝 研究開発センター システム・ソフトウェア生産技術研究所‡

1 はじめに

我々は、金融機器(現金自動預入支払機)組込みソフトウェア(以下、ATM[1]ソフト)を対象としたドメインCASEの開発を進めている。ATMソフトの中で、銀行毎に仕様の変更の多い、媒体(カードや通帳)を入力する部分に対象を絞り、これまでに、状態遷移モデルをベースとした仕様記述言語の提案、ソフト自動生成の可能性の検討などを行なってきた[1][2]。

本稿では、状態遷移モデルによる仕様記述から、ソフトの自動生成を行うジェネレータの実現方式と、開発したジェネレータをATMソフトに適用し、その評価について述べる。

2 仕様記述モデル

我々のモデルでは、状態遷移モデルをベースに、一つの取引の仕様を、オブジェクトモニタとメインモニタに分けて記述する(図1参照)。

オブジェクトモニタは、一つのデバイスタスクの動作を監視する。デバイスタスクは、ハードウェア(デバイス)を直接制御し、標準タスクとして用意されている。媒体の入力は、複数のデバイスタスクを並行に動作させて行なう。メインモニタは、複数のデバイス間のインタラクションをオブジェクトモニタを使って制御する。どのデバイスを使用するかは、取引によって異なる。メインモニタの上位には、これら全体を制御する親モジュールがある。

この状態遷移モデルにおけるイベント/アクションには、メインモニタとオブジェクトモニタの間の内部的なものと、オブジェクトモニタとデバイスタスクとの間、メインモニタと親モジュールとの間の外部的なものがある。特に、オブジェクトモニタがデバイスタスクに動作指示を与えると、デバイスタスクからは、経過報告を定められた順番で返すという特徴がある。この途中報告の内容や順番は、デバイス毎に定義されている。

3 ジェネレータに対する要求

(1) 実機組込み可能なソースコードを生成できること

実機に組み込む時、生成ソースコードに一切の修正を加えず、コンパイル/リンクできることが望ましい。

まず、生成ソースコードが仕様記述通りに動作するよう、ジェネレートルールを定めなければならない。また、生成ソースコードが、そのままコンパイル可能なためには、ファイルのインクルード文、シンボルの宣言文などを自動的に挿入する機能も必要である。さらに、デバイスなどの外部と定められたインタフェースでやり取りできるようにする必要がある。

この上で、生成ソースコードのコード量、性能の面において、従来のソフトと同等の規模、速度が得られることを目指す。

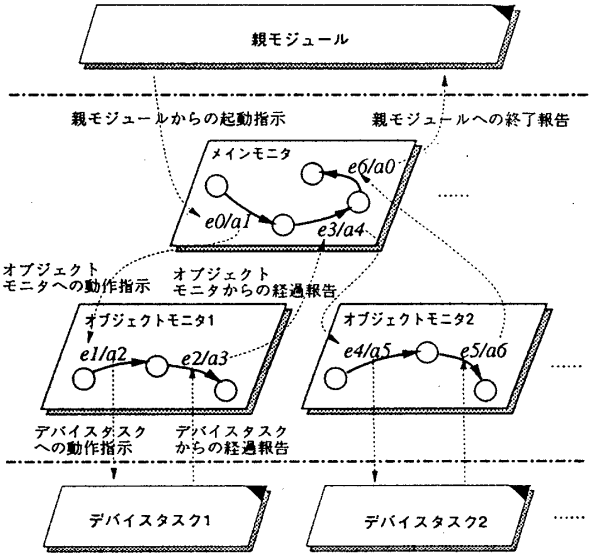

図1: 仕様記述モデルの概念図

(2) 最適化できる構成となっていること

状態遷移図の理解性を高めるために、あえて冗長に記述することがある。これを忠実に生成すると、ソースコードも冗長となり、コード量が増大し、実行速度が低下するなどの問題が生じることも考えられる。そこで、状態遷移図の冗長な部分や、意味的に同一な部分を一つにまとめるなどの最適化ができる構成となっている必要がある。

4 実現方式

4.1 ジェネレートルールの定義

仕様記述からソースコードへの変換規則(ジェネレートルール)を定めた[2]。ここでは、状態遷移図の形状からソースコードの構造への変換規則と、イベント/アクション文字列から条件式/実行文への変換規則を定義している。

(1) ソースコードの構造への変換

変換例を図2に示す。このように、ソースコード上では、状態による分岐と、イベントによる分岐による、二重の分岐構造となる。また、斜体字で示した部分に、変換した条件式/実行文を埋め込む。

(2) 条件式/実行文への変換

外部イベント/外部アクションは、デバイスなど外部とのやり取りであり、インタフェースが定まっているので、プログラム部品として予め用意しておく。変換は、部品名をキーにプログラム部品の実体を取り出すことで行う。

内部アクション/内部イベントは、モニタ間の通信を表し、どの通信が発生したかという二値情報で充分である。そこ

---

*A Generator of CASE especially for ATM software

†Yuji TAMAKI, Noboru FUJIMAKI, Yoko SHIMIZU, Jiro OKAYASU, Toshiyuki OBI

‡Systems and Software Engineering Laboratory, Research and Development Center, TOSHIBA Corp.

[1]Automatic Teller Machine

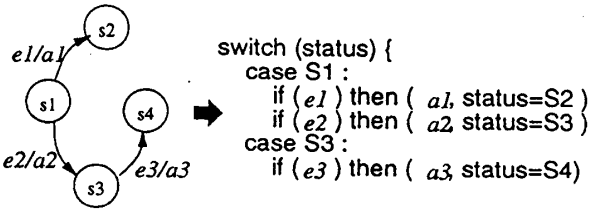

図 2: 生成ソースコードの例

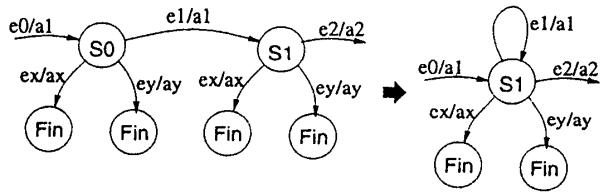

図 3: 最適化の例 パタン 1

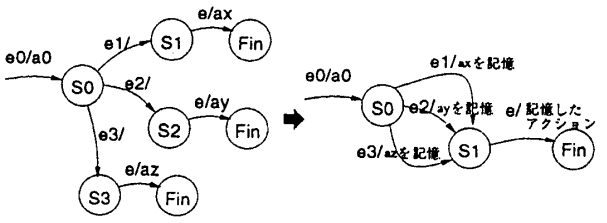

図 4: 最適化の例 パタン 2

で、これをフラグで実現し、その割当を自動的に行う。ソースコード上では、内部アクションとしてフラグをオンする実行文に、内部イベントとしてフラグがオンであるか評価する条件式に変換する。

4.2 ジェネレータの実現

(1) 実機組込み可能なソースコードを生成する機能

前節で定めたジェネレートルールに従って、ソースコードの本体部分の生成が可能である。また、ファイルのインクルード文、シンボルの宣言は、プログラム部品に実体とともに登録しておき、自動的に取り出す方式とした。外部とのやり取りは、プログラム部品として実現したので、そのインタフェースが変更されても、プログラム部品を修正/更新するだけで良い。

これによって、生成したソースコードに一切の修正を加えず、コンパイル/リンク可能となった。

(2) 最適化

実際のソースコードを生成する前に、一旦、状態遷移図における「意味」を持つトークンの並びで構成される、中間的なコードに変換する構成とした。最適化は、この中間的なコードに対して行う構成とした。最適化の例を図 3、4 に示す。

図 3のパタンは、あるイベントが発生する順番が定まっており (e0, e1, e2 の順)、それらのイベントによる、一つのパスでつながった複数の状態がある場合である。このとき、仕様上ではイベントの発生する順序を正しく記述するが、ソースコード上では、S1, S2 を一つにまとめるという最適化が可能である。ATM ソフトでは、デバイスタスクからのイベントの順序が定まっており、オブジェクトモニタで多く現われる。

図 4のパタンは、S0 で三種のイベント (e1, e2, e3) のいずれかを待ち、その後、ある共通のイベント (e) を受け取った時、最初にどのイベントが発生したかによって、実行するアクションが異なる場合である。このとき、ソースコード上では、最初のイベントを受け取った時、最後に実行するアクションを内部に記憶することにより、S1, S2, S3 を一つにまとめるという最適化が可能である。これは、ATM ソフトのメインモニタにおいて、複数のオブジェクトモニタから異常系のイベントを待つ時、多く現われる。

5 評価

開発したジェネレータを、ATM ソフトに適用し、評価を行なった。

5.1 ジェネレータに対する評価

ジェネレータにより実際にソースコードを生成し、実機上で動作させた。この結果、仕様記述通りに動作することを確認した。これは、ジェネレートルールの正当性を実証したことにもなる。

5.2 生成ソースコードに対する評価

生成ソースコードを実機に組み込み、その性能を従来のソフトと比較した。その結果、人間の操作が入らない処理単位を抽出した部分の処理時間において、± 0.3%の性能 (スピード) であった。従って、現行の方式で充分適用可能であると判断できる。

コード量は、ソースコードレベルで従来の約 1.3 倍となった。これは、状態遷移図の理解性を高めるために、あえて冗長に記述したことにも起因する。

しかし、中間的なコードを設け、これを最適化できるようになっているので、これを行うオブチマイザを開発するなどにより、コード量を減らせることも確認済みである。試算したところ、現状ソフトの約 1.1 倍程度に押えられる見通しを得た。

6 おわりに

状態遷移モデルで記述された仕様記述から、ソフトウェアを自動生成するジェネレータの実現方式と、これを ATM ソフトに適用した結果について述べた。

ジェネレータを開発したことにより、仕様記述からソフトウェアを自動生成し、実機上で動作できるレベルまで達した。さらに、生成ソフトウェアの実行速度面で実適用可能であることを確認した。コード量の面でも、オブチマイザを開発するなどによって、ほぼ現状通りの規模で押えられる見込みである。

今後は、オブチマイザの開発や、プログラム部品の作成支援、仕様記述レベルでの検証などを行っていく予定である。

参考文献

[1] 藤巻 他,“金融機器組込みソフト向け POL の開発”, 情報処理学会第第 45 回全国大会予稿集,1992.10

[2] 清水 他,“金融機器組込みソフト向け CASE”, 情報処理学会ソフトウェア工学研究会, 93-SE-91, 1993