ソフトウェア工学 91－3
（1993. 3. 2）

金融機器組み込みソフト向けCASE

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

清水洋子　藤巻昇　小尾俊之　岡安二郎　玉木裕二

ソフトウェア生産において，分野を指向したドメインCASEによる高度な支援の期待が高まっている．この中で我々は，金融機器組み込みソフト向けのCASEを開発中である．本論では，この一環として行っているソフト自動生成について報告する．

まず対象分野を分析し，そこから仕様表現のモデルを構築した．このモデルでは，ソフトが扱う実体や情報をオブジェクトとして捉える．そしてこのオブジェクトを中心に，オブジェクトの表現，オブジェクトの動作の表現，機能の表現の3つの観点から仕様を表現する．また，このモデルに基づいたソフト自動生成の実現方式を提案した．これは，仕様変更の可能性があるか否かに留意した方式である．

さらに，提案方式に対しモデルの表現能力，生成ソフトの実用性を評価するため，2つの試行実験を行った．その結果，提案モデルを基にした生成ソフトの機能面，実行速度面，コード量面での実現可能性が確認できた．

CASE especially for Automatic Teller Machine Software

Yoko SHIMIZU　Noboru FUJIMAKI　Toshiyuki OBI　Jiro OKAYASU　Yuji TAMAKI

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

70 Yanagi-cho Saiwai-ku KAWASAKI,210,Japan

In software development, advanced support with Domain-CASE, that is CASE especially for particular domain, is expected. Under such situation, we are trying to construct a CASE especially for Automatic Teller Machine Software. This paper describes about automatic code generation, which is a kernel of this ATM-CASE.

At first, we analyzed the features of the target domain, and built a specification description model. In this model, entities and information operated in the software considered as objects. And specifications are described with three view points; object, object action, and function. Secondly, we proposed a code generation mechanism follows the model, paying attention to be easy to customize.

Furthermore, we made two feasibility studies of our code generation mechanism. The result proved that generated software has satisfactory functionality and performance.

1 はじめに

1.1 背景

ソフトウェアの高生産性，高品質の確保を目的とし開発手法やツールの導入を行う動きの中で，対象分野に特化させることでより高度な支援を行うドメイン CASE の効果が期待されている．この中で我々は，銀行等の現金自動支払／預入／振込をサービスする金融機器の組み込みソフト (以下 ATM[1] ソフトと呼ぶ) を対象としたドメイン CASE の開発を進めている．

1.2 アプローチ

ATM ソフト開発では，同一シリーズの機種に対して複数銀行の動作仕様を実現するため，類似した複数のソフト開発を行う．つまり，類似していて共通に扱える部分と，仕様に応じて変更する部分とが存在する．

そこで我々は，共通に扱える部分は標準ソフトを準備し，仕様により変更する部分は高級言語からのソフト自動生成を実現する，という 2 つの支援アプローチを採っている．標準ソフトにより再利用を促進し，ソフト自動生成によりソフト開発の負荷を軽減し，最終的に高生産性，高品質の確保につなげるのが目的である．

本論では，ソフト自動生成のアプローチについて述べる．

1.3 対象範囲

我々が扱う ATM ソフト構造の概略を図 1に示す．

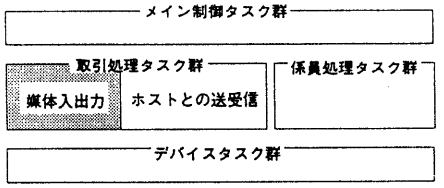

図 1: ATM ソフト構造

「デバイスタスク群」,「係員処理タスク群」は共通に扱える割合が高いため，標準ソフトにより対応している．変更が多いのは，顧客やホストコンピュータとやりとりをして入金，支払等の取引を実現する「取り引き処理タスク群」である．この中で我々は，特に媒体 (カードや通帳など) を入出力する部分を対象範囲として設定し，ソフトの自動生成を目指している．

2 特性分析

ソフト自動生成の実現に向け，仕様表現のモデルを構築する必要がある．そしてこのモデルが対象の特性を表現しやすいものであることで，より高度な支援が可能となる．そこで ATM ソフトの特性を導出するため，(1) 仕様，(2) ソフト構造の 2 つの視点から分析を行う．

2.1 仕様の視点から

取引処理タスクは顧客が選択した取引 (入金，支払等) を行う．この取引は成立する場合もしない場合もあるが，いずれにせよ，ある取引の途中で他の取引に移行することのない閉じた処理である．したがって，機能を取引で分割することは妥当である．

一方処理の内容を考えると，カード，通帳，紙幣，コインといった実体の入出力を行う．また暗証，金額，振込先口座等の情報を入力し，接客のために画面にメッセージとしての情報を出力する．この実体と情報を総称してオブジェクトと呼ぶことにすると，各取引の内容はオブジェクトの入出力を扱うことであるといえる．

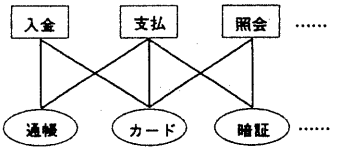

図 2: ある銀行での取引とオブジェクトの関係

図 2にある銀行を仮定して，取引とオブジェクトの関係を示した．入金取引ではカードと通帳を扱い，支払取引ではカード，通帳，暗証を扱う．つまり，取引ごとに扱うオブジェクトを限定できることが分かる．

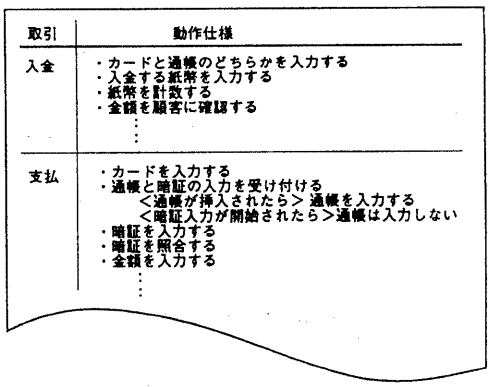

図 3: ある銀行各取引の動作仕様

そして各取引の動作仕様は，図 3に示すようになる．具体的には,「カードの後に紙幣を入力する」といった順序関係,「通帳と暗証を入力する」といった並列関係等を規定している．

言い換えると，各取引はオブジェクト間のインタラクションを規定することで表現できる．

[1] Automatic Teller Machine

一方，各オブジェクトの動きについても調べてみる．例えばカードを取り上げてみると，動作仕様では「カードを入力する」と表現されている部分で，実は図4のような動作を行っている．この一連の動作 (以下動作列と呼ぶ) は，入金取引の場合でも支払取引の場合でも変わらない．つまり，**オブジェクトが持つ動作列は，各取引で共通である．**

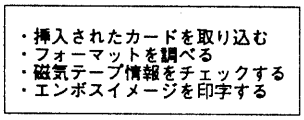

図 4: カード入力の動作列

銀行ごとの仕様の違いについても分析を行った．その結果，オブジェクトの動作列については，ほぼ共通であることが分かった．違いが現れるのは，**各取引で扱うオブジェクトの種類，及びそれらの入出力に関するインタラクションに集中している．**

以上に述べた，仕様の視点からの特性をまとめると，図5になる．

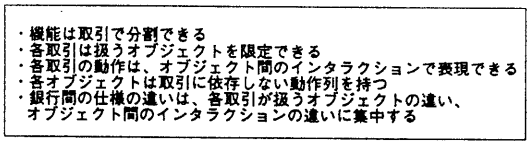

図 5: 仕様の視点からの特性

2.2 ソフト構造の視点から

ここでは特に，オブジェクトがソフト構造でどう実現されるかに注目する．

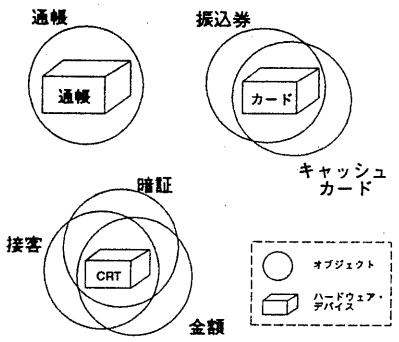

図 6: オブジェクトとデバイスの違い

まず，オブジェクトとハードウェア・デバイスとの違いを明らかにする (図6参照)．

例えば通帳オブジェクトは，通帳デバイスと1対1に対応すると考えて良い．但しカードデバイスは，キャッシュカードと振込券の両方を扱うこともあるため，図6に示すように，2つのオブジェクトが1つのデバイスを共有する形となる．

一方，暗証オブジェクトや金額オブジェクトは，実際にはCRTからの入力となるためCRTデバイスを共有する．また接客用のメッセージもCRT上に出力される．したがって接客オブジェクトもCRTを共有する．

つまり，オブジェクトは何らかのデバイスと対応しており，オブジェクトとデバイスは多対1の関係にある．

次に，オブジェクトの実現法について考察する．これは図7に要約される．

各オブジェクトには，いくつかの動作列が存在するが，これらは変更を加えられることがないため，標準ソフトとして隠ぺいするのが望ましい．隠ぺいの際は，動作列がハードウェア・デバイスの制御と密接な関係にあることから，デバイス単位にまとめるのが良いと考えられる (図7参照)．例えば，カードデバイスを中心にカード入力の動作列，振込券入力の動作列等をまとめ，CRTデバイスを中心に暗証入力／照合の動作列，金額入力／確認の動作列をまとめるのである．

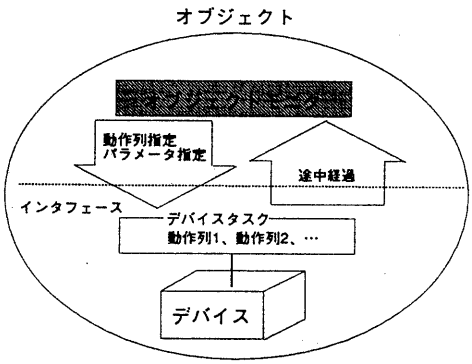

図 7: オブジェクトの実現概要

こうしてまとめた標準ソフトは，図1の中のデバイスタスク群に相当し，デバイス単位にデバイスに関わる操作を集約するという，一般的なアプローチと同じである．

動作列を隠ぺいした一方で，この動作列を順番に実行したり並行に実行するというインタラクション (各取引の動作仕様) は，変更が多くあることから標準ソフトとして隠ぺいできない．そこで，このインタラクションを外部から制御できるよう，次のインタフェースをデバイスタスクに準備する．(図7内に矢印で示す)．

動作列指定

どの動作列を実行するかの指定

パラメータ指定
　動作列の中で可変な値の指定（例:入金限度額）

途中経過情報
　動作列のどこまで実行が進んでいるかの報告

これらを使うことにより，適切なタイミングで動作列の実行を指示できると共に，この動作の経過に依存したオブジェクト間のインタラクションを扱うことが可能となる．

さらに，このインタフェース上に オブジェクトモニタを導入する (図 7中の斜線部)．これは，デバイスタスクからの途中経過情報を監視 (モニタ) しながら記憶する．また，デバイスタスクに対して動作列実行を指令するための簡易な手段を提供する．つまりオブジェクトモニタは，デバイスタスク制御を容易に行うための枠組であり，各取引の動作仕様を実現する際に，オブジェクトのインタフェースとして機能する．

まとめると，図 7全体は 1 つのオブジェクトを表現しているが，一番下はハードウェア・デバイスで，その上に標準ソフトとしてのデバイスタスクを載せ，さらにその上にオブジェクトのインタフェースとしてオブジェクトモニタを置く．そしてデバイスタスクは標準ソフトとして準備し，オブジェクトモニタ部分だけをソフト自動生成の対象として扱う．

3　モデルの構築

前章の分析から導きだした特性を踏まえ，ソフト自動生成のベースとなるモデルを構築した．以下ではこのモデルを (1) オブジェクトの表現，(2) オブジェクトの動作の表現，(3) 機能の表現という側面から説明し，最後にその全体関連を示す．

3.1　オブジェクトの表現

オブジェクトの表現として必要なのは，(1) どれだけのオブジェクトがあり，(2) それらがどのような関係にあるかである．どれだけのオブジェクトがあるかについては，対象を限定しているためほぼ予め列挙することができる．但し，新たなデバイスが追加されたり，同じデバイスでも新たな実体や情報を扱うことになった場合は，新しいオブジェクトが追加される．

一方オブジェクト間の関係については，取引を介して表現できる．なぜなら，オブジェクトは既に図 2で示したように，各取引で使用するか否かを限定することができるからである．この関係もほぼ固定できるが，銀行ごとに多少異なる可能性はある．

以上をまとめると，オブジェクトの表現として必要なのは図 8である．

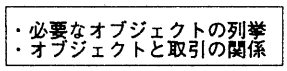

図 8: オブジェクトの表現

3.2　オブジェクトの動作の表現

オブジェクトの動作表現には次の 2 つの要素がある．(1) 動作の基本である動作列の進行状況を監視すること，(2) 他のオブジェクトとのインタラクションを可能にすることである．

動作列の監視のためには，デバイスタスクへ動作列の実行指令を出し，デバイスタスクからの途中経過報告を受け取ることが表現でき，さらに進行状況を記憶しておけることが必要となる．この要件を満たすという理由から，オブジェクトの動作の表現には状態遷移モデルを採用した．これを図 9を用いて具体的に説明する．

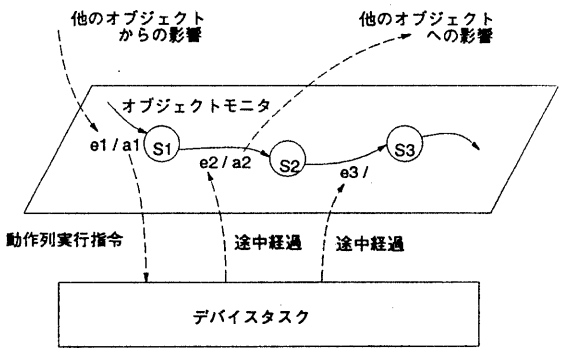

図 9: オブジェクトの動作表現

アクション a1 によりデバイスタスクに動作列実行を指令し，状態は S1 に遷移する．デバイスタスクは処理を進め，その途中経過を随時報告してくる．1 つめの報告をイベント e2 として受け取り状態は S2 へ，2 つめの報告をイベント e3 として受け取り状態は S3 へと遷移していく．このように，動作列の進行と共に状態が遷移していくため，この進行状況を状態という形で記憶していることになる．つまり，状態遷移モデルにより動作列を監視することが表現できるのである．

一方他のオブジェクトとのインタラクションについて考える．これについては，他のオブジェクトからの影響をイベントで，他のオブジェクトへの影響をアクションで表現する．

これも図 9で説明する．他のオブジェクトからの影響としてイベント e1 を受け取り，これをトリガーにして動作列の実行を開始する．また，イベント e2 を受け取った後のアクション a2 は，他のオブジェクトにおいてイベントとして機能し，これにより他のオブジェクトへ影響

を与えることができる．

このようにイベントとアクションを用いることで，オブジェクト間のインタラクションが表現可能となるのである．

以上述べたように，オブジェクトの動作表現の２つの要素である，動作列の監視及び他のオブジェクトとのインタラクションは，状態遷移モデルによって表現できることが分かった．

ここで，イベントとアクションについて分類整理しておく（図10参照）．

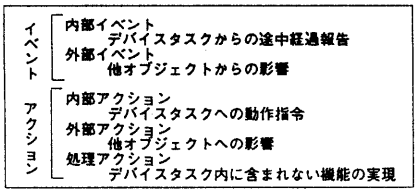

図 10: オブジェクトモニタ上のイベントとアクション

デバイスタスクとのやりとりを，それぞれ内部イベント／内部アクションと呼び，他のオブジェクトとのやりとりは外部イベント／外部アクションと呼ぶ．

処理アクションというのは，デバイス内の動作列に含まれていない機能を実現するものである．例えば，通帳の挿入ミスを２回まで認める仕様があるとすると，ミスの回数をカウントするアクションが必要である．これは「通帳の入力」という動作列の中では行われないため，処理アクションとしてオブジェクトモニタの中で実現することになる．

3.3 機能の表現

機能として表現すべきことは，(1) オブジェクト間のインタラクションにより実現される取引の動作仕様と，(2) オブジェクト内において行うデバイスタスクへの動作実行の指令である．

取引動作仕様の表現法を，図11を用いて説明する．この例は，オブジェクトＡにおいてイベントｅが発生したとき，取引Ｘの場合はオブジェクトＢへ，取引Ｙの場合はオブジェクトＣへ影響を与えたい場合である．

左側は，オブジェクトＡの外部アクションを，直接オブジェクトＢまたはＣの外部イベントと結びつけて表現しようとしている．この方法では，オブジェクトＡに取引による分岐の表現を含めることになり，「取引に依存しない不変の動作列をオブジェクトに隠ぺいする」というそもそもの思想に反する．

そこで，図11の右に示すように取引記述を導入する．これも状態遷移モデルで表現されるものである．オブジェクトＡは外部アクションａを取引記述に対して起こし，

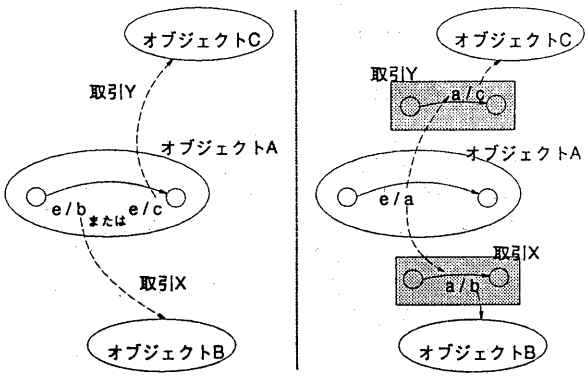

図 11: 取引記述の位置づけ

現在動いている取引記述がこれをイベントとして受け取る．そしてオブジェクトＢまたはＣの適切な方に対しアクションｂまたはｃを起こし影響を与える．

つまり，オブジェクトの外部イベント，外部アクションは，すべて取引記述との間でやりとりされる．これにより，取引ごとに異なるオブジェクト間のインタラクションを，それぞれの取引記述で表現し分けることができ，オブジェクト自身は取引による影響を考慮しなくて済むようになる．

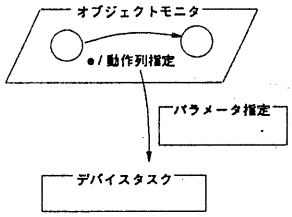

図 12: デバイスタスクへの動作実行指定の位置づけ

動作実行の指令は，図12に示す位置づけで行う．

指令を送るときに指定すべきものは，図7で示した通り動作列とパラメータである．

動作列は，オブジェクモニタに記述するアクションの名前により指定する．一方パラメータは，動作列の内容を指定する (例えば入金限度額の指定) ものであり，動作列に比べて銀行間での変更が生じやすい．そこでこれを局所化するために，オブジェクトモニタ上には表現せず別途指定することにする．

例を示すと，紙幣オブジェクトに対し「紙幣を入力する」という動作列の実行を指定するときは，アクション名として「紙幣を入力する」を用い，そのパラメータである入金限度額は別途指定し，状態遷移上には表現しない．

まとめると，取引動作仕様は取引記述の状態遷移モデルで表現し，オブジェクト内での動作指令指定は，オブ

ジェクトモニタのアクション名，及び別づけのパラメータ指定により表現する．

3.4 全体関連

以上説明したモデルの全体関連を図 13に示す．

図全体が，オブジェクトと取引の関係を示すオブジェクトの表現を展開したものとなっている．各オブジェクトの斜線以外の部分が，ソフト自動生成で扱う範囲である．**オブジェクトの動作の表現**は，それぞれオブジェクトモニタ中に状態遷移モデルで行う．**機能の表現**は，取引記述中の状態遷移モデル，オブジェクトモニタ内のアクション名，及び別づけのパラメータ指定にて行う．

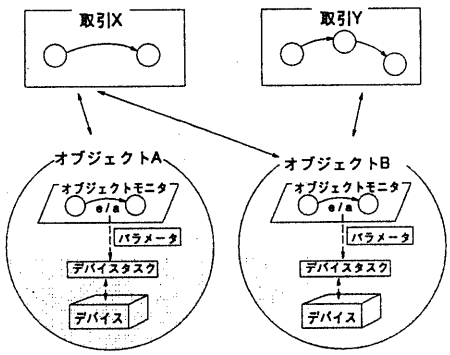

図 13: モデル全体関連図

4 自動生成の実現法

これまでに述べたモデルをベースにして，仕様記述からコードに変換するソフト自動生成を実現する．

4.1 仕様記述

仕様として記述すべきものは以下である．

オブジェクトの表現
図 2のようなオブジェクトと取引の関係を示す情報であり，これは一定形式のテキストにて記述する．

オブジェクトの動作の表現
オブジェクトモニタの記述に相当し，これには状態遷移図を用いる．外部イベントや内部イベント等を区別するため，イベント名やアクション名に特定の記号を冠するという規則を設けている．

機能の表現
取引記述には状態遷移図を用いる．イベント名，アクション名に上記と同様の規則を適用する．パラメータ指定は，内部イベントそれぞれに対し，一定形式のテキストにて記述する．

4.2 コードへの変換

自動生成の対象範囲は，既に述べた通り図 1中に斜線で示す媒体入出力部分である．この部分のソースコード構成を，自動生成対象外の既存ソフトと共存するよう考慮しながら，図 14のように決定した．

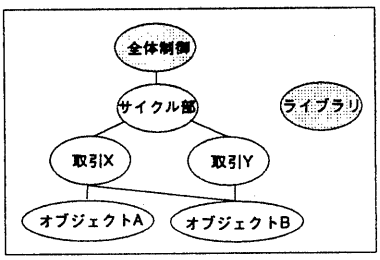

図 14: ソースコード全体構成

全体制御
自動生成対象外のタスクとのインタフェースとして機能する．

サイクル部
これは，それぞれ独立して並行に動作するオブジェクトを制御するための機構である．

現在の取引に応じ，必要な取引モジュール，オブジェクト・モジュールを順番に繰り返しコールする．例えば現在取引 X を行っていれば，「取引 X」，「オブジェクト A」，「オブジェクト B」を順に繰り返しコールする．

取引
取引記述の内容を実現する．

オブジェクト
オブジェクトモニタの記述内容を実現する．

ライブラリ
取引記述やオブジェクト・モニタ内でのイベントやアクションの機能の中で，共通的な機能をサブルーチンとして実現する．

この中で，**全体制御**，**ライブラリ**は，予め人手により作成しておく．残りの部分については，仕様記述からの変換を行う．以下にその変換法を述べる．

サイクル部は，「オブジェクトの表現」の仕様記述から生成する．

取引及びオブジェクトは，両方状態遷移図からの変換であるため，その方法は同一である．

まず状態遷移のひな型を，(1) 状態による分岐と (2) イベントによる分岐の二重条件分岐で実現する．図 15に簡

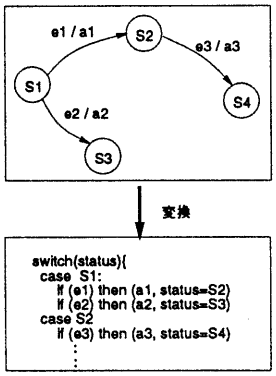

図 15: 状態遷移図ひな型の変換

単な例として，状態遷移図及び変換後の疑似コードを示している．

状態S1から出る2つの遷移は，"CASE S1:"内の2つのif文に変換される．ここで，実行される遷移がイベントの評価順序に依存しないよう，イベントe1とe2は互いに排他的な表現にする必要がある．

この疑似コードでは，ひな型の中身にはe1やa1といった仕様記述上の表現がそのまま入っており，ソースコードには変換されていない．これらの変換は以下のように行う．

内部イベント／アクションは，状態遷移図上の「イベント／アクション名」とこれを実現する「ソースコード表現」の対応表を使って変換する．この対応表は不変のものとして予め準備しておくことが可能である．なぜなら，内部イベント／アクションはデバイスタスクという標準ソフトとのやりとりだからである．但し内部アクションについては，そのパラメータをパラメータ指定記述から生成することになる．

外部イベント／アクションは取引ごとにも銀行ごとにも異なるものであり，予め準備することはできない．これはオブジェクトモニタと取引記述の間の通信であり，通信があるか否かの2値が表現できれば十分である．そこでこれをフラグで実現することとし，フラグの割当をソフト生成時に自動的に行う方法を採る．

処理アクションは，内部イベント／アクションの場合と同様対応表を用いて変換する．但しこの対応表は予め準備できるとは限らない．今までにない処理アクションが必要になった場合は，これを対応表に追加するという，人手の作業を伴うことになる．

5 試行評価

構築したモデル，及びこれに基づくソフト自動生成方式の評価を行うため，2つの試行実験を行った．

5.1 仕様記述実験

概要

主にモデルの評価を目的に，ソフトへの変換を除いた仕様記述までの試行を行った．対象とした範囲は，媒体入力の前半部分(カードや通帳を入力するまで)で，入金，支払，照会，記帳，振込，振替の6取引を扱った．

現状のソースコードやドキュメント類から仕様を興し，仕様記述言語で表現し，これをレビューする形式を採った．

結果と考察

記述結果

記述したオブジェクトモニタは7つ，取引記述は5つであった．これらの関係を図16に示す．

取引＼オブジェクト	カード	通帳	金額	暗証	振込方式	振込券	接客
入金	○	○					○
支払	○	○	○	○			○
照会	○			○			○
記帳		○		○			○
振込／振替	○	○	○		○	○	○

図 16: 実験で扱った取引とオブジェクトの関係

オブジェクトモニタ

複数取引で同じオブジェクトを扱っている場合については，すべて共通のオブジェクトモニタにより記述することができた．例えば，カードオブジェクトは入金，支払，照会，振込／振替の4取引と関係しているが，どの取引も共通のオブジェクトモニタを使用している．つまり，オブジェクトの動作列が取引間で共通に使用できることが確認できた．

但し接客オブジェクトは例外である．これは取引中に画面を通じて顧客にメッセージを伝える役割を持つもので，取引処理への依存度が非常に高い．そのため，取引に依存した動作列を複数持つことになった．

取引記述

一方取引記述に関しても，振込処理と振替処理が類似しているため，共通化して，内部に取引による分岐を埋め込む方が効率的であることが分かった．

記述量

オブジェクトモニタ，取引記述の状態遷移は，それぞれが状態数5～35個程度の記述となった．状態数が多いのは，扱う媒体数が多い取引記述である．このような取引では，基本的な流れが存在するため，記述全体としての理解性には問題はなかった．

記述作業

記述作業は1つの取引から着手した．オブジェクトと取引記述を交互に修正，追加しながら試行錯誤で進めていった．しかし2つめの取引からは，記述済みのオブジェクトモニタが存在する部分は比較的容易に作業を進めることができた．

したがって，すべてのオブジェクトモニタの記述が揃った段階で，他の銀行向けに取引記述を行うという作業は，比較的進めやすいと考えられる．変更の多くは，取引記述と接客オブジェクトに集中すると予測される．

5.2 実機組み込み実験

概要

仕様記述実験で記述した仕様の中から，特に入金取引に絞り，実際にソース生成規則に則ってソースコードを(人手で)作成し，実機上で動作確認することを行った．また実行速度，コード量を測定し，現状との比較を行った．

結果と考察

動作確認

実験中発見された問題は，ソースコード生成規則を改良することですべて対応でき，実機上にて仕様通り動作させることができた．これにより，本モデルに基づく自動生成ソフトの，機能面での実現可能性が確認できた．

実行速度

実験用ソフトが動作した時間の中から，人間の処理が入らない単位を切りだし，現状のソフトの場合と実行速度の比較を行った．その結果，10秒弱の処理単位において0.1秒程度の違いが見られただけであった．現状ソフトと同程度の速度が出せるという点で，実行速度面でも十分実現可能であることが確認できた．

コード量

コメントを抜いたソースコードの行数により比較を行った．

組み込み実験を行った部分のみを比較すると，約1割増加した．これは，単純なルールでコードに変換することによって生じる冗長性が原因と考えられる．

一方，仕様記述実験の結果を利用して，この範囲をすべて自動生成した場合のコード量を算出した．具体的には，状態及び遷移の数によりソースコードが線形に増加すると仮定し，その係数を実験時のコードから推定して行ったものである．

この結果を基に比較を行うと，コード量は現状ソフトの80%程度に減少するという結果が導かれた．減少の主な原因は，本モデルにおいてオブジェクトの動作列を切りだし，取引間，銀行間で共通に使える枠組を提供している効果であると考えられる．この動作列がうまく切り出せていないと，1銀行内の複数取引間で冗長さが生じ，さらにこれを他の銀行向けに変更する場合にも，冗長さを増すことになるからである．

以上より，コード量においても実現可能な範囲であることが確認できた．

6 おわりに

ATMソフトの開発に特化して支援を行うATM向けCASE開発の一環として，媒体入力部のソフト自動生成について述べてきた．

特性の分析を行い，そこからベースとなるモデルを構築し，これに基づく自動生成実現方式を提案した．また試行実験により，機能面，実行速度面，コード量面での実現可能性も確認できた．

現在は仕様記述用のエディタ，及び自動生成用のジェネレータの開発を進めている．また，本論で述べたアプローチによる実適用を計画している．

今後は，検証等の周辺支援も検討していく予定である．

謝辞

本研究での分析や試行実験において，有益な助言，多大な助力をいただいた，株式会社東芝柳町工場の高松悟氏，山本武正氏，東芝インテリジェントテクノロジ株式会社の安達倫幸氏に感謝致します．

参考文献

[1] 加地他，“状態遷移をベースとした部品合成システムの試作”，情報処理学会第38回全国大会，1989.

[2] 岸他，“状態遷移モデルに基づくプログラム部品合成システムの開発”，情報処理学会研究報告91-SE-80，1991.

[3] 藤巻他，“金融機器組み込みソフト向けPOLの開発”，情報処理学会第45回全国大会，1992.

[4] S．シュレィアー/S．J．メラー著，本位田真一/伊藤潔監訳，“続・オブジェクト指向システム分析”，啓学出版，1992.