BankVision® 環境に対するプログラムレス開発ツール活用

Utilization of Programless Development Tool in BankVision® Environment

武 澤 孝 弘，樋 口 英 之

要　約　BankVision® のバッチ処理開発では，オープンプラットフォームの技術とツールを活用し，データベースアクセスや帳票出力といった部分でコーディングを簡素化する手法を確立している．稼働後 10 年を経過し，オープン技術・ツールを有効活用して更なる生産性向上を図るべく，「プログラムレス」開発を実現する新たなツールの活用を検討し，銀行勘定系バッチ処理への適合性という基準で「ODIP」を選択した．そしてこのツールを BankVision のバッチ処理構造に取り込めるような対応を実施した．

Abstract　In batch processing development in BankVision®, we have established a method that makes it possible to simplify the coding in parts such as database access and form output by making use of the technology and tools in open platform. Ten years after operation, we are considering the use of new tools to realize “program less” development in order to improve productivity by effectively utilizing open technologies and tools; we selected “ODIP” based on the criteria of compatibility with bank accounts batch processing. And implemented measures to incorporate this tool into the BankVision batch processing structure.

1. は じ め に

BankVision® のアプリケーションプログラムは主に COBOL 言語で記述されており，追加開発においても同様である．BankVision の COBOL プログラムは，オープンプラットフォーム上で稼働する利点を活かして，SQL で簡単にデータベースアクセスを可能とし，汎用機ベースの勘定系開発に比べて一定の開発生産性向上が可能な構造となっている．

BankVision のオンラインシステムとバッチシステムは処理構造が異なっており，それぞれ異なる開発規約や開発手法を整備している．近年，SoE 領域の情報システムへデータを連携するニーズの増加により，バッチ処理開発の割合が高くなる傾向にあり[*1]，ユーザからもバッチ処理開発の生産性向上へのニーズが高まってきた．生産性向上は言い換えると開発のスピードアップであり，今後拡大が予想される Fintech 関連の開発でも重視されるニーズである．

これらのニーズに応える目的で，「プログラムレス開発ツール」の活用を検討した．一般的にプログラムレス開発ツールとは，ソースコードを記述することなく業務アプリケーションを開発できるツールのことである．プログラムレス開発ツールの活用によって，COBOL プログラム開発のコーディング作業と，コーディングを確認する単体テストが不要となり，更なる生産性向上が期待できる．

また，現状では BankVision の開発者には金融業務の知識と，COBOL 言語の習得が求められている．プログラムレス開発ツールにより，COBOL 言語を習得しなくても開発することができれば，ユーザ（銀行）による開発も平易になり，開発負荷が軽減されるメリットがある．

プログラムレス開発ツール活用にあたっては，単に新たなツールを使用するだけではなく，これまでミッションクリティカルなシステムとして稼働実績を重ねてきたBankVisionの設計思想や既存資産との融合によるシナジー効果が得られるよう検討した．

本稿ではこのようなBankVisionのバッチ処理開発へのプログラムレス開発ツールの活用について紹介する．まず2章でプログラムレス開発ツール活用への検討内容について述べ，3章でその具体的な実現策，4章で実現策の試行内容について述べる．

2. プログラムレス開発ツール活用に向けて

本章では，BankVisionのバッチ処理構造の特徴と，プログラムレス開発ツールにODIPを選定した理由，ODIPの特徴と活用時の課題について述べる．

2.1 ツール活用の検討

BankVisionのバッチ処理構造は，外部の新たな技術・製品を取り込んで進化できるというオープンプラットフォーム上で稼働する利点を活かして確立されている．具体的には，BankVisionのバッチシステムの中心となるデータベースにRDBMSの「SQL Server」を採用しており，SQL言語により簡単にデータベースアクセスが可能である．また，帳票開発においても，GUI上で開発できる帳票定義ツール「EUR[*2]」を採用しており，汎用機以上の生産性を確保している．また，SQLによるデータベースアクセス，ツールでの帳票定義によって，COBOLでのコーディング内容は簡素化されたものとなり，「規約原則」を定めて自動的にチェックすることで基本的なプログラム構造を保っている．しかしながら，細部においては依然，コーディング内容の自由度は高く，開発生産性が開発者のスキルに依存し，また属人的なコーディングが作成されうるという点に課題がある．プログラムレス開発ツールの活用によってコーディング作業がなくなれば，これらの課題が解消されると期待した．

プログラムレス開発ツールには，株式会社インテリジェント・モデルが提供する「ODIP」を選定した．主な選定理由は，以下に挙げるようなツールの特性が，銀行勘定系バッチ処理への適合性が高いと判断したためである．

・スケジュール実行が可能で，バッチ方式の一括処理に適している．
・他の金融機関での稼働実績があり，かつ特定の勘定系システムに依存していない．
・金融業務で特徴的な，営業日を考慮した日付計算等の演算機能が提供されている．

2.2 プログラムレス開発ツール「ODIP」の概要

ODIPはJavaベースの高速開発ツールであり，以下の特性がある．

（1）モデルドリブン・アーキテクチャ[*3]であることから処理ロジックの実装が簡易

現行開発言語であるCOBOLは，処理ロジックはプログラム・コーディングを行い，コンパイルにより処理ロジックそれぞれを実行体としたうえで実行し，処理結果を得る．それに対してODIPでは処理ロジックをODIPリポジトリへ定義し，実行体は処理ロジック共通の実行エンジンにより，ODIPリポジトリに定義された情報をもとに実行される．ODIPの実装イメージを図1にて図説する．

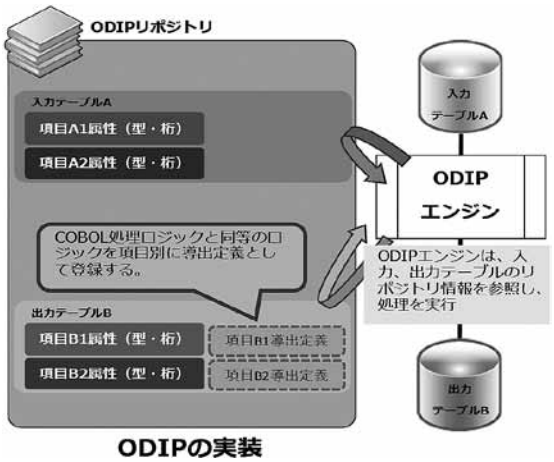


図1　ODIP の実装イメージ図

ODIP の場合，開発作業は主に属性定義と出力の導出定義をするだけでよく，COBOL のように処理ロジックをプログラミングする必要がないため，処理ロジックの実装から実行までの開発作業が COBOL より簡易である．

(2) GUI 画面による処理ロジックの実装が可能

ODIP では，GUI 画面から以下のような内容を定義するだけで，それに応じたバッチ処理を開発することが可能である．

・入力データ単位の属性定義（テーブル，項目），入力データのリレーション，ソート定義
・出力データ単位の属性定義（データ種類，データに出力する項目），出力レイアウト
・出力データ単位抽出条件，出力項目単位に導出演算（計算単位，計算方法，関数演算等）

開発者は，これらの入出力情報，処理情報を ODIP のリポジトリ情報として登録すればよく，従来の構造図設計やプログラム設計を含めたコーディング記述作業に比べ負荷が小さい．集約，集計処理については，集約，集計単位を定義するだけであり，COBOL のような Loop 条件，Break 条件の記述（コーディング）が不要である．また，出力レイアウトを定義するだけで，自動的に出力項目の属性が定義され，出力項目個々の属性定義は不要である．

(3) テーブル・項目の一元管理により項目の追加・廃止や属性変更が簡易

ODIP では，全てのテーブル，ファイルとそれらに定義された各項目について，処理ロジックそれぞれで必要となる入力データ定義，出力定義，導出演算定義を結びつけてメタデータとして ODIP リポジトリ内で一元管理する．そのため，項目間の関連性（項目の廃止や属性変更時に利用している処理ロジック，出力データ等への影響等）が一覧できる．リポジトリ情報を変更すれば，各処理ロジックで使用している項目属性も自動変更される．

2. 3　プログラムレス開発ツール活用時の課題

BankVision で ODIP を活用する場合，次のような課題があった．

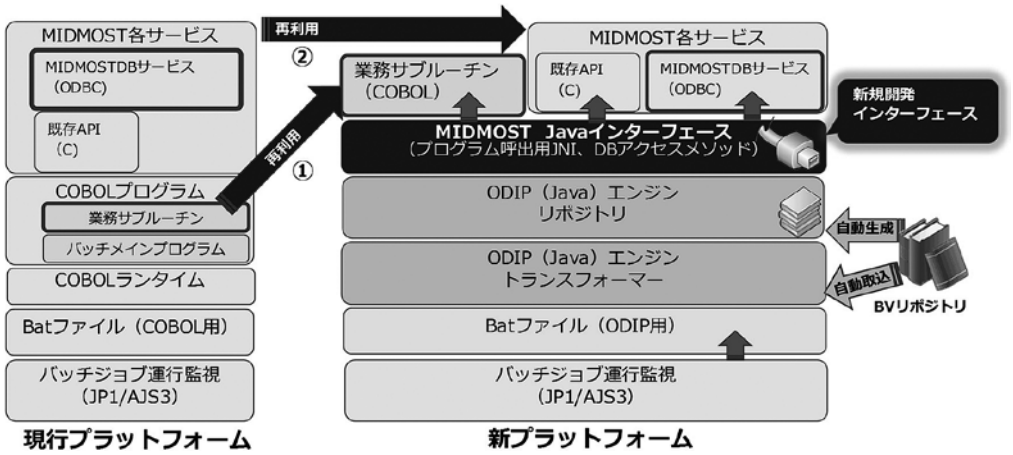
まず，ODIP で開発したアプリケーションから，既存資産である利息計算サブルーチンなどの共通処理を有効活用できない．利息計算サブルーチンは，複雑な業務ロジックで構成されており，また処理の重要度も高く，相応の期間とコストをかけて検証してきた資産である．このような共通処理を，ODIP 用に再度開発することは，保守性の面から得策ではない．

次に，BankVision の COBOL アプリケーションは，オープンミドルウェア「MIDMOST」経由でデータベースにアクセスすることでトランザクションの一貫性が確保されているが，ODIP で開発したアプリケーションのコミット制御の仕組みでは，同水準のトランザクション一貫性を確保できない．このトランザクション一貫性も，金融機関向け勘定系システムにおいては妥協ができない要件である．

これらの課題に対して実施した対応を 3 章にて紹介する．

3.　既存開発資産とプログラムレス開発ツール「ODIP」の融合

2. 3 節の課題解決とバッチ処理開発の生産性向上を目的として，既存開発資産である「業務サブルーチン（COBOL)」と「MIDMOST　DB サービス」を再利用（図 2 の①と②）するためのプログラム連携インターフェース（図 2 の MIDMOST Java インターフェース）を開発した．本章の各節で説明する．



No.	再利用する開発資産	期待する効果	実現するためのインフラ対応
①	BankVision 業務サブルーチン （COBOL）	再利用で開発不要	Java から COBOL プログラムを呼び出すためのインターフェース 業務サブルーチンを呼び出すための JNI 自動生成機能
②	MIDMOST DB サービス	トランザクションの一貫性保証	JDBC[*4] 用の DB アクセスメソッドから MIDMOST の ODBC[*5] の DB サービスを呼び出すためのインターフェース

図 2　既存開発資産の再利用　全体概要図

3. 1　COBOL サブルーチンの呼び出しインターフェースの提供

ODIP は，外部関数呼出のインターフェースを提供しているが，実装形態としては Java メソッドを前提にしているため，Java と COBOL が連携するインターフェース（JNI[*6]）が必要となった．図 3 内の「Java/COBOL 連携用インターフェース（JNI（C））」が該当する．

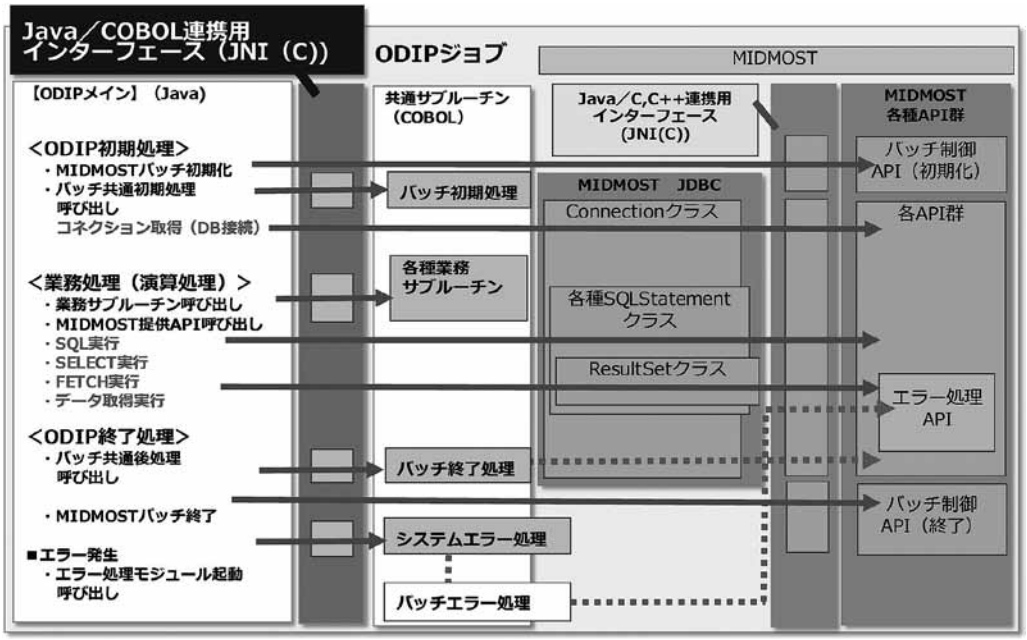


図 3　Java/COBOL モジュール連携概要図

この構造を実装するにあたり，開発者の負荷軽減を目的として工夫した点を二つ挙げる．

（1）インターフェースプログラム（JNI）自動生成

COBOL 呼び出しのためのインターフェース（図 3 内「Java/COBOL 連携用インターフェース（JNI（C））」）をプログラミングすると，製造工数と保守対象成果物が増え，バッチ処理開発全体の生産性が低下する．そのため，このインターフェースプログラムには，MIDMOST/DE[*7] のサブルーチン定義書から自動生成する補完ツールを提供し，開発者の負荷軽減を図る．なお，図 3 内の Java/C，C++ 連携用インターフェース（JNI（C））は，MIDMOST/DE から自動生成するのではなく，MIDMOST が提供する．

（2）COBOL 呼び出しインターフェースの改善

BankVision の COBOL プログラム内で COBOL サブルーチンを呼び出す場合は，USING 句を使用してパケットで複数の引数を定義する．一方，ODIP は，COBOL サブルーチン呼び出しを実行したときに受け取る戻り値は，整数値一つである．開発者が COBOL と同じように複数のサブルーチン実行結果を取得して後続のロジックで利用できるよう，COBOL と ODIP のインターフェースの相違を吸収する共通化したプログラム構造が必要となった．

MIDMOST から以下の三つのメソッドを提供し，ODIP から業務サブルーチン（COBOL）を呼び出すときのフレームワークを構築した．

ⅰ）COBOL サブルーチン呼び出し時のパケット引数の組み立て（引数設定メソッド）

ⅱ）COBOL サブルーチン実行メソッド

ⅲ）戻り値（out 引数）の取得（引数取得メソッド）

実装イメージは図 4 のとおりである．（提供機能は，図 4 の点線枠内である．）

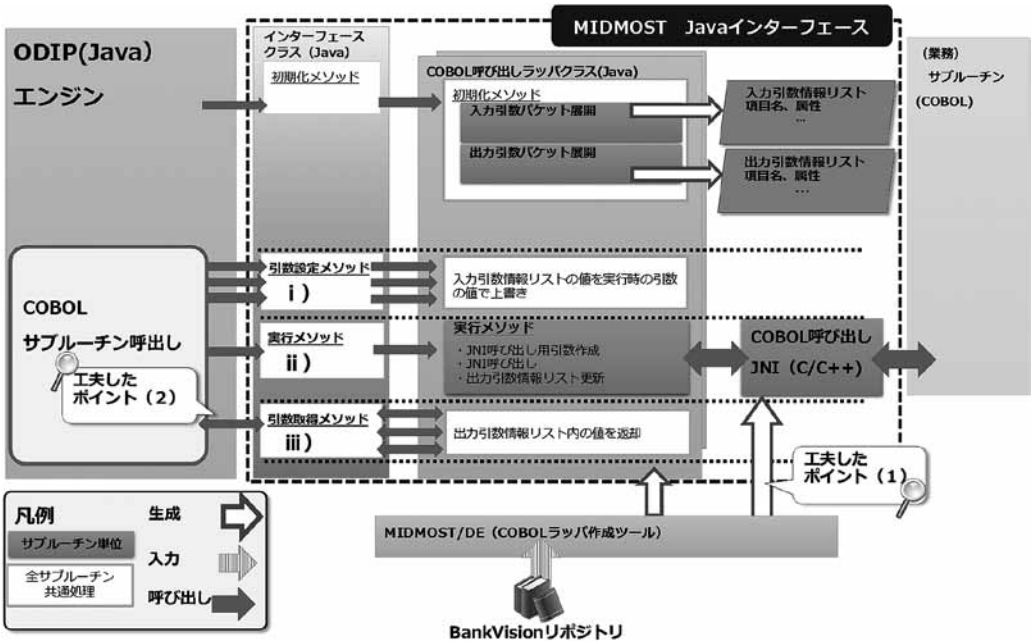


図 4　業務サブルーチン（COBOL）呼び出しイメージ

3. 2　MIDMOST によるトランザクション一貫性

既存の ODIP では，MIDMOST と異なり JDBC により DB 接続が実装されており，SQL 実行の都度，SQL 実行用のコネクションを取得し，その SQL の終了でコネクションを解放するとともにコミットを発行している．つまり，同一バッチプロセス内でトランザクションの一貫性が保証されていない実装となっている．ODIP にデータソースとして「MIDMOST」を追加し，MIDMOST の DB サービスを活用してコネクションを引き継ぎながら DB アクセスするように ODIP を改善し，トランザクションの一貫性を確保した．既存の ODIP と改善後の ODIP のトランザクション制御の実装イメージを図 5 に記載する．MIDMOST に ODIP が利用する JDBC 相当の DB アクセスメソッドを準備することにより，ODIP に大きな改修をせずに対応することができた．

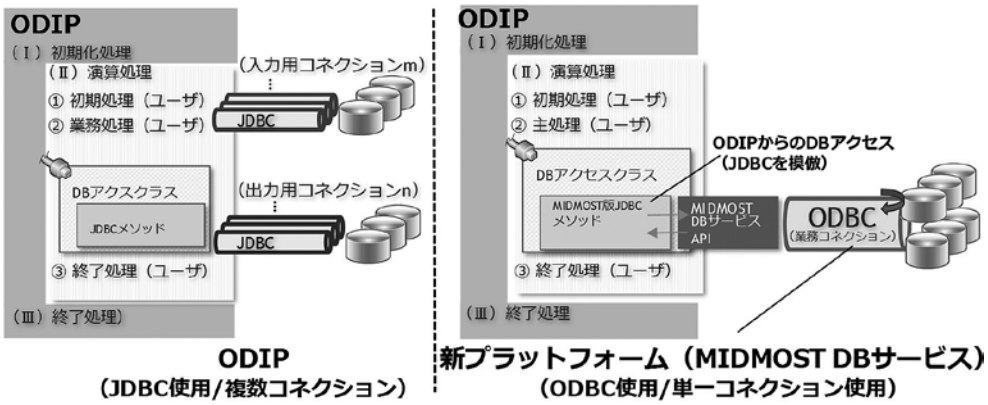


図 5　トランザクション一貫性を保証する MIDMOST DB サービス（MIDMOST JDBC）利用

4.　プログラムレス開発の試行評価

ODIP 活用の有効性を確認するため，実際に次のような前提での試行開発を実施した.

・前章に記載した，「COBOL サブルーチン呼び出しインターフェース」，「MIDMOST DB サービスを利用するインターフェース」を使用

・BankVision の既存のバッチ処理（各業務分野より標準的な内容のプログラムを 10 本選定）を題材として，同内容の処理を再構築

・試行開発担当者として，ODIP を用いた開発の未経験者を 5 名選定し，2 日間の基本的な技術習得を実施

4. 1　試行開発結果

試行開発結果は表 1 のとおりとなり，COBOL 開発における物理設計（詳細設計）・製造～単体テストの作業工数に比べ，一定の生産性向上が確認された.

表 1　生産性比較結果

題材とした COBOL プログラム		ODIP 試行開発所要時間(h)			COBOL 開発想定所要時間(h)		
処理特性(業務分野)	STEP 数	設計製造	単体テスト	合計	設計製造	単体テスト	合計
サブシステム IF 処理(業共外接)	881	21	17	38	36	27	63
サブシステム IF 処理(業共外接)	818	22	8	30	36	27	63
帳票作成処理(業共外接)	1,197	53	45	98	44	33	77
帳票作成処理(預金)	1,651	39	20	59	44	33	77
帳票作成処理(融資)	1,172	22	18	40	40	30	70
帳票作成処理(業共外接)	1,621	22	26	48	44	33	77
帳票作成処理(融資)	1,596	23	11	34	44	33	77
帳票作成処理(業共外接)	1,572	15	6	21	44	33	77
サブシステム IF 処理(業共外接)	640	16	10	26	36	27	63
サブシステム IF 処理(預金)	845	24.5	10	34.5	36	27	63
平均	1,199	26	17	43	40	30	71

題材とした COBOL プログラムは，物理設計・製造～単体テストで 2,375 STEP/人月の生産性が想定されるプログラムであり，標準的な内容である．これに対し ODIP 試行開発では，

COBOL の STEP 数で規模換算すると，物理設計・製造～単体テストで 3,918 STEP/人月の生産性（39%向上）となった.

4. 2　試行開発結果の評価

前節の試行開発結果（39%の生産性向上）の妥当性を，物理設計・製造と単体テスト別に確認した.

・物理設計・製造

COBOL 開発に比べ，ODIP 開発では平均 14 時間の作業時間削減（35%の生産性向上）となった．削減時間に相当する COBOL 開発時の作業は，業務ロジック以外のプログラムロジック検討を含めたコーディング作業であり，その作業量削減割合が 35%という想定は，コンパイル等を含めた COBOL 開発の作業実態からすると妥当性がある.

・単体テスト

COBOL 開発に比べ，ODIP 開発では平均 13 時間の作業時間削減（43%の生産性向上）となった．削減時間に相当する COBOL 開発時の作業は，業務ロジックに直接関係のない部分も含めた全プログラムロジックの網羅的な確認作業（カバレッジテスト）であり，その確認量削減割合が 43%という想定は，テストデータ準備等を含めた COBOL 開発の作業実態からするとこちらも妥当性がある，

生産性向上の度合いについては，「業務ロジック以外のプログラムロジック」がどの程度の量の処理であるかや，プログラムレス開発ツールに対する開発者の習熟度にも依存するため，これらの生産性変動要素を見極めて作業計画をコントロールすることが必要となる．いずれにせよ ODIP 開発では，業務ロジックの作成に専念できる点が COBOL 開発に比べて省力化につながり，一定の生産性向上が期待できるという点が試行開発によって確認できた.

4. 3　更なる生産性向上に向けた検討

プログラムレス開発ツールを活用すると，前節で述べたとおり「作成量，確認量の削減」によって物理設計・製造～単体テスト工程の生産性が直接的に向上する．その上で更なる生産性向上・品質向上につなげるために今後取り組み予定としている検討課題を 2 点挙げる.

（1）論理設計・物理設計：業務ロジックをより簡素化するデータベース設計

COBOL 開発に比べ ODIP 開発の生産性が高いことは前節で述べたが，ODIP で定義する業務ロジックをより簡素化することができれば，開発および保守の生産性をより高めることができる．そのための方策として，バッチ処理の入力となるデータベースに，個々のバッチ処理で複雑な加工編集が必要となるデータを，データベース項目としてあらかじめ用意しておくという対応が考えられる.

BankVision は「基幹系 DWH」と呼ぶバッチ専用環境を設けており，この基幹系 DWH 上で稼働するバッチ処理が参照するデータベースは，オンラインデータベースの素データを日次鮮度で保持し一部バッチ処理に有用な派生属性を追加した「共用明細データベース」と，共用明細データベースと外部データ等を特定のバッチ処理のために保持する「目的別データベース」である．ODIP を適用する場合に業務ロジックが複雑になってしまうような出力項

目があれば，例えばその出力項目そのものを「共用明細データベース」あるいは「目的別データベース」にあらかじめ保持して，複雑な業務ロジックをこれらデータベース作成処理に局所化することで，全体的な開発および保守の生産性が高まることが期待できる．

(2) テスト・保守：単体テストと結合テスト範囲の最適化

ODIP を適用する場合，単体テストが業務ロジックに関係のある部分の確認に集中できるようになることから，単体テストの省力化が実現されることは 4.1 節で述べたとおりであるが，単体テストと結合テストを合わせて品質を作りこんでいくことを考えた場合，単体テストと結合テストの目的やバランスを見直して最適化することも必要である．

例えば，ODIP の単体テスト環境で結合テスト環境のテストデータ（本番データをマスク化したもの等テーブル間のデータ整合性が保たれているテストデータ）が使用できれば，従来はテスト担当者が手作業で作成したテストデータでホワイトボックステストを行っていた単体テスト工程で，一部結合テスト工程のブラックボックステストを前倒しで実施できるようになり，品質の作りこみを前倒しすることでの品質向上が期待できる．

5. お わ り に

本稿で検討した結果，プログラムレス開発ツールの採用により，BankVision バッチ処理開発の生産性向上が期待できることが分かった．この生産性向上を定着させるために，ツールの特性を活かした設計パターン（処理全体をシンプルな処理に分解し，同種の処理やデータを共用しながら，それらの組み合わせで業務ロジックを構成していく事例集）等のノウハウの蓄積を進めていく．

「プログラムレス開発」は，開発を平易化して開発要員の育成と調達を容易にすることによって，開発力の向上とスピードアップの要請に応えることができる．この要請は今後も強まっていくため，引き続き BankVision の進化の一方向としてプログラムレス開発ツールの活用検討に取り組んでいく．

＊ 1　事例では，BankVision 適用時のアプリケーション開発工数の約 6 割，保守開発時の約 7 割をバッチ処理開発工数が占めている．

＊ 2　EUR：株式会社日立製作所が提供する帳票作成ツール．

＊ 3　モデルドリブン・アーキテクチャ：仕様（モデル図）をもとにビジネス要件や，ソフトウェアの機能や構造をプログラムに落とし込んでいく開発手法．ODIP は，本開発手法に基づき，GUI や仕様（モデル図）によって直感的に成果物を自動生成していく開発ツールである．

＊ 4　JDBC（Java Database Connectivity）：Java プログラムからデータベースにアクセスするための標準インターフェース（API）である．

＊ 5　ODBC（Open Database Connectivity）：アプリケーションソフトがデータベース管理システム（DBMS）などに接続し，データの取得や書き込み，操作などを行う方法の標準を定めたもの．Microsoft 社が制定したもので，主に同社の Windows で動作するデータベース関連ソフトウェアで用いられる．

＊ 6　JNI（Java Native Interface）：Java プラットフォームにおいて，Java で記述されたプログラムと，他の言語（C や C++ など）で書かれた，実際の CPU の上で動作するコード（ネイティブコード）とを連携するためのインターフェース仕様である．JNI の場合，MIDMOST API を呼び出すために，API 単位にラッパーのプログラムが必要となる．

＊ 7　MIDMOST/DE: MIDMOST を基盤とする COBOL による AP 開発のための開発支援ツールで，「オブジェクト（DB）定義」，「コード定義」，「サブルーチン定義」，「登録集」等各種定義情報のリポジトリを保有する．

執筆者紹介　武　澤　孝　弘（Takahiro Takezawa）

1997年日本ユニシス(株)入社．信用金庫勘定系ユーザ担当の営業職に従事．その後，BankVisionの開発/適用/保守を担当．現在，金融システム第三本部開発三部に所属し，BankVisionの適用作業に従事．

樋　口　英　之（Hideyuki Higuchi）

1997年日本ユニシス(株)入社．都市銀行証券関連プロジェクトの開発後，オープン系金融外接パッケージの開発/適用に従事．ネットバンク勘定系システムの構築/本番適用後，BankVisionの開発/適用/保守を担当．現在，アウトソーシング本部インフラサービス部に所属．