管理番号＃２１－０１

平成２３年度
民間基盤技術研究促進制度 成果報告書

Java バッチシステム開発自動化ツール の研究開発

委託先：　㈱キャナリーリサーチ

平成２３年１１月

平成２３年度　成果報告書

「Java バッチシステム開発自動化ツールの研究開発」

目　次

1　研究開発課題の背景

IT サービス市場では、約 10 年前にメインフレームシステムからオープンシステム化の流れが起こり、その開発言語は COBOL からオープンシステム言語(C や VB など)へ、シフトが始まった。ところが、４年前からオブジェクト指向言語の Java が主流となり、2007 年 3 月調査では、利用言語の 37.6%（売上構成比）が Java と報告されている(2003 年での Java 利用は 17%)。この傾向が続くと 2012 年には約 6 兆円の市場規模の約 63.4%が Java を利用すると予測される。つまり、IT サービス市場のシステム開発において、Java の利用が拡大してきている。また、システムは、オンライン画面システムとバッチシステムに大別できる。特に前者においては、Java 画面系フレームワークや Java ジェネレータなどの開発が進み、生産性向上に寄与しているが、バッチシステムにおける Java の利用は始まったばかりであり、Java バッチ系フレームワーク（Spring Batch, TERA バッチ(NTT データ)）の製品化がやっと始まったところである。

バッチシステムの Java 利用が増加した理由を以下に示す。

1) Java VM（仮想マシン）自体の性能向上：これまでは Java によるバッチプログラムの性能が VM に依存して悪いと評されてきたが、近年では大きく改善され、バッチにおいても Java 利用が可能になってきた。
2)Java 以外の言語を使用できるプログラマが非常に少なくなり、Java 以外の言語を選択することが困難になってきた。つまり、Java 要員以外を集めるのが困難になってきた。
3)ハードウェアの切換え時、従来の環境(メインフレーム上で開発された COBOL 資産などハードウエアに依存したり、OS に強く依存している環境)を維持するのが困難になっている。つまり、最新のプラットフォーム（ハードウェアや OS など）上に、プラットフォーム独立な Java で再構築して実行したほうがよほど安価に高速に実行できるため、必然的に Java の選択を強いられる。
4)オンライン画面処理に関するサーバサイドはほとんど Java で記述されているため、あえてバッチシステムを Java 以外の他言語で記述する利点が少なくなっている。これは開発スタイルの変化でもあり、オンライン画面システムとの連携が重視されてきている。

この変化の中で Java を使ったバッチシステム開発に対するニーズが増大しており、Java プログラムジェネレータが開発されているが、適用できる処理モデルが限定されているため全てのバッチシステムの自動生成には至っていない。また、自動生成方式を採用しても、自動生成が適用できない部分は人手による開発となり２種類以上の開発方式が混在し、開発・保守運用管理が複雑となり全体としての生産性向上を阻害する問題となっている。

そこで本研究開発では Java によるすべてのバッチシステム構築を自動化するツールの開発を目標とする。本ツールでは、最小の Java の知識さえあれば利用可能であり、オブジェクト指向プログラミング技術も前提としていない。本ツールの主目的は、1)生産性の飛躍的な向上、2)高品質なソフトウェアの提供、3)保守性の向上にある。

２　研究開発の全体計画

２－１　研究開発課題の概要

Web 系開発では Java が主要言語として定着しているが、バッチ系開発では Java 利用はまだ少なく、システム全体の開発要員の確保や開発生産性・品質確保・保守性に課題がある。システム全体を Java で統一的に効率よく開発できれば、これらの課題が解決され、その社会的効果は極めて大きい。本研究開発の目的は、全てのバッチシステムを Java で自動生成すると同時に、高生産性・高品質・高保守性を実現する汎用的技術を確立して、Java によるシステム全体開発の現実的扉を開くことにある。本研究開発で確立すべき中核技術は、全てのバッチシステムを高品質で汎用的に自動生成できる技術にある。このため、使用領域が限定される自動生成方式ではなく、汎用的な自動生成方式とするために、プログラム部品（バッチコンポーネント）の利用による自動生成技術に加えて、プログラムの随所にフリー記述を可能とする（フリー記述自動合成）技術と、高品質を達成するために、フリー記述自身や、自動生成部との任意の検査項目で静的解析を可能とし自動生成と検証をタイムリーにサイクルで回せる（ジェネレーション＆インスペクション）技術を実現する。

汎用性の高い新たな発想の自動化ツールができれば Java という一つの言語で web 系とバッチ系が作成される時代が到来する。Web 開発技術者もバッチ対応が可能となりプログラミングの効率は著しく向上しかつ安価となる。フリー記述を含めて品質のチェックがすべての段階で自動的に行われることにより、製品の信頼性を著しく向上でき、社会的に大きく貢献できる。世界的に見ても Java のバッチ系開発の自動化ツールは数少なく、汎用性の高いツールができればソフトウェア技術面での国際競争力も高まる。

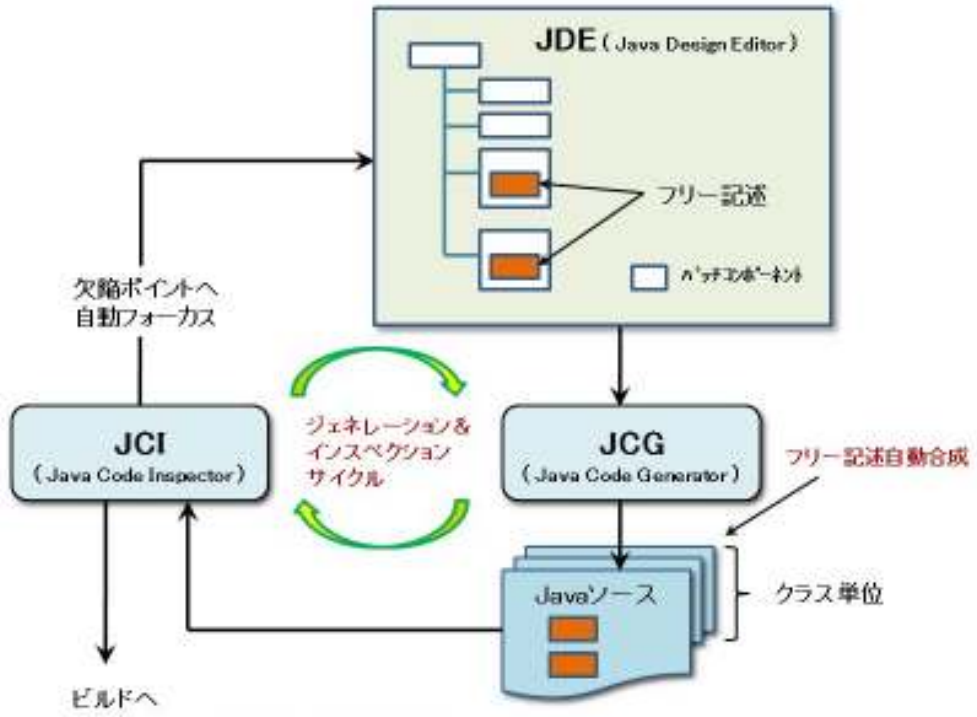


図１　本ツールの概要

図１に本ツールの概要を示す。本ツール実現のために、次の４つのサブテーマごとに研究

開発を実施する。

【バッチコンポーネント】
Java によるバッチプログラムを以下に示す部品に分解します。すべての Java バッチプログラムが、これらの部品の組合せで記述可能です。本システムでは、この部品をバッチコンポーネント（以降、コンポーネント）と定義し、４つのカテゴリに分類する。
・Java 言語の構文要素
・データベース処理コンポーネント
・ファイル入出力コンポーネント
・その他のコンポーネント

＜サブテーマ＞
1) Java Design Editor（JDE）の開発
上記各コンポーネント毎に、そのコンポーネントで定義、設定しなければならない情報（カスタマイズポイントと定義され、生成されるプログラムの設計情報となるもの）を規定し、それらを得るための利用者からの入力が最小限で済むように GUI を設計、実装します。当然、利用者にとっては使いやすい、eclipse の利便性を損なうことのないしかもフレンドリなインターフェイスを追及する必要があります。主なカスタマイズされる設計情報を示します。
・定義ファイルからのスキーマ（データベース表、ファイル）情報の取り込み
・データベース表検索のためのカーソルの定義
・データベース表、ファイルからのレコードのフェッチ
・データベース表、ファイルへのレコードの出力
・初期処理、終了処理、業務ロジック等で利用するフリー記述

2) Java Code Generator(JCG) の開発
バッチコンポーネントおよび JDE で得られる設計情報を入力仕様とし、Java ソースコードに変換するエンジンを実装します。以下の点につき十分に配慮した生成をします。
・オブジェクト指向に基づくクラスとメソッドのコード生成
・安全な SQL 文の生成
・言語ロケールにしたがったコメント文の挿入、さらに Javadoc, XDoclet 等他のドキュメンテーションツールに連携可能なコメントの生成
・可読性の高い、十分にソースコード納品に耐えうるソースの形式
・通常の Java 標準コーディング規約(Sun microsystems 社 Coding 規約)に準拠、または、利用者から提示されたコーディング規約に準拠したコード生成

3) Java Batch Runtime Class (JRC)の開発
実行時に必要な機能と既製のバッチフレームワークとの連携を実装する。
・実行時の機能の実装
バッチプログラムとしては不可欠なデータベースアクセスとファイルアクセスのエンジン部分、および例外処理のクラスから構成される。
・バッチフレームワークとの連携
既存のバッチフレームワークを容易に取り込めるよう、ラッパクラスを用意します。これより、本システムの提供する JRC 以外に、いろいろなフレームワークを利用することができる。

4) Java Code Inspector（JCI)の連携UIの開発

JCIはすでに弊社で開発された、Javaソースコードを入力としてCode Inspection(CDI)を行う静的解析ツールです。Eclipseのプラグインとして簡単にインストールできる製品で、ファイル単位、パッケージ単位、プロジェクト単位で手軽にCDIが行える。ただし、結果はテキスト形式で表示またはファイル出力になる。

本システムでは、生成されたコードにはフリー記述が存在するため、このJCIを組み入れ、いつでもCDIを行うことを可能とするとともに、JCIで指摘されたポイントへ自動でJDE上の設計情報やフリー記述箇所にジャンプし、開発者が容易に修正できるようなGUI連携を行う。実装される主な連携機能を示す。

・JDE上へのジャンプ機能
・ソースにマークをつける機能
・設定画面のプリファレンスページ化

２－２　研究開発の最終目標（平成２３年１０月末）

ア　JDEに関する研究開発【サブテーマ】

(1) 定義したすべてのコンポーネントのカスタマイズポイントが入力できること。
(2) コンポーネントのカスタマイズポイントの編集結果をすべて確認する手段があること。
(3) ジェネレータによって生成されるソースコードとコンポーネントの対応が 100%取れること。
(4) eclipseのプラグインとして、eclipseに慣れている膨大な数の開発者にとっても抵抗感なく使用できるUIを提供すること。

イ　JCGに関する研究開発【サブテーマ】

(1) すべてのバッチコンポーネントおよび各コンポーネントのすべてのカスタマイズポイントについて生成されるJavaソースコードが文法的に正しいこと。
(2) すべてのバッチコンポーネントのすべての可能な組み合わせに対して生成されるJavaソースコードが文法的に正しいこと。
(3) すべてのバッチコンポーネントのすべての可能な組み合わせに対して生成されるJavaソースコードの実行速度が妥当であること。(別途、指標を定義する)

ウ　JRCに関する研究開発【サブテーマ】

(1) バッチ要件となるSQL文が効率よく実行されること。
(2) 定義可能なすべてのファイル操作が効率よく実行されること。
(3) 本ツールのJRCだけでバッチシステムのフレームワーク動作が保障されること。
(4) 既存のフレームワークを使用した場合、そのフレームワーク機能の実行についてはJRCのオーバーヘッドが僅少、できれば無視できるレベルであること。

エ　JCIとの連携UIに関する研究開発【サブテーマ】

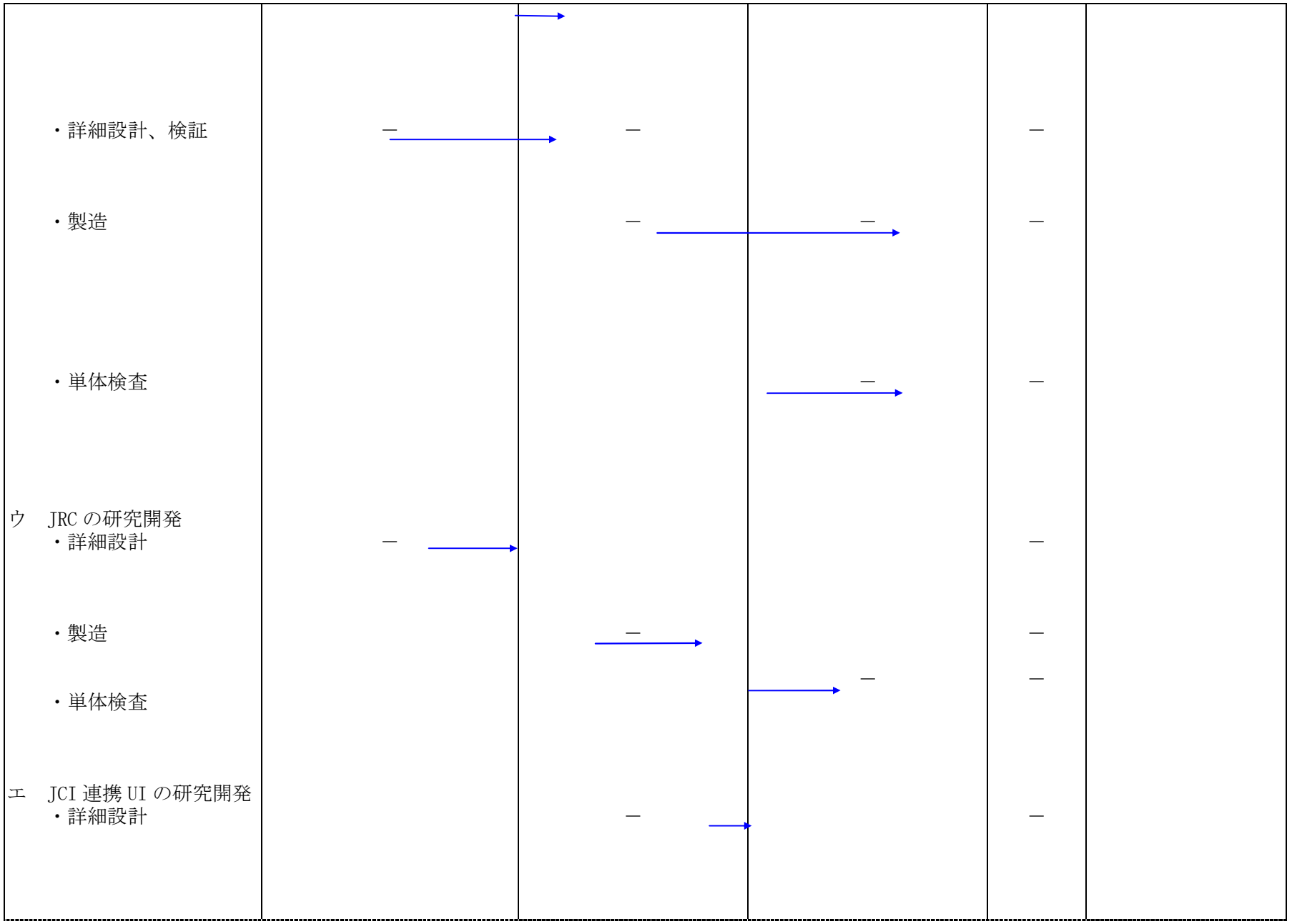
(1) 既存eclipseの優れたGUIを損なうことなく使用できること。
(2) 指摘されたコードの生成元に適切かつ高効率なフォーカス移動ができること。
(3) 設計時に仕様とする、静的に解析可能な全て検査（欠陥）項目を漏れなく検出すること。

オ　本ツール全体のテスト･評価を完了すること。製品レベルの出荷判定に合格すること。

２－３　研究開発の年度別計画

金額は非公表

研究開発項目	２１年度	２２年度	２３年度	計	備考
Java バッチシステム開発自動化ツールの研究開発					
ア　JDE の研究開発					
・詳細設計	―	―		―	
・Eclipse JDT 上検証	―			―	
・製造		―	―	―	
・単体検査			―	―	
イ　JCG の研究開発					
・レファレンスコード設計	―			―	

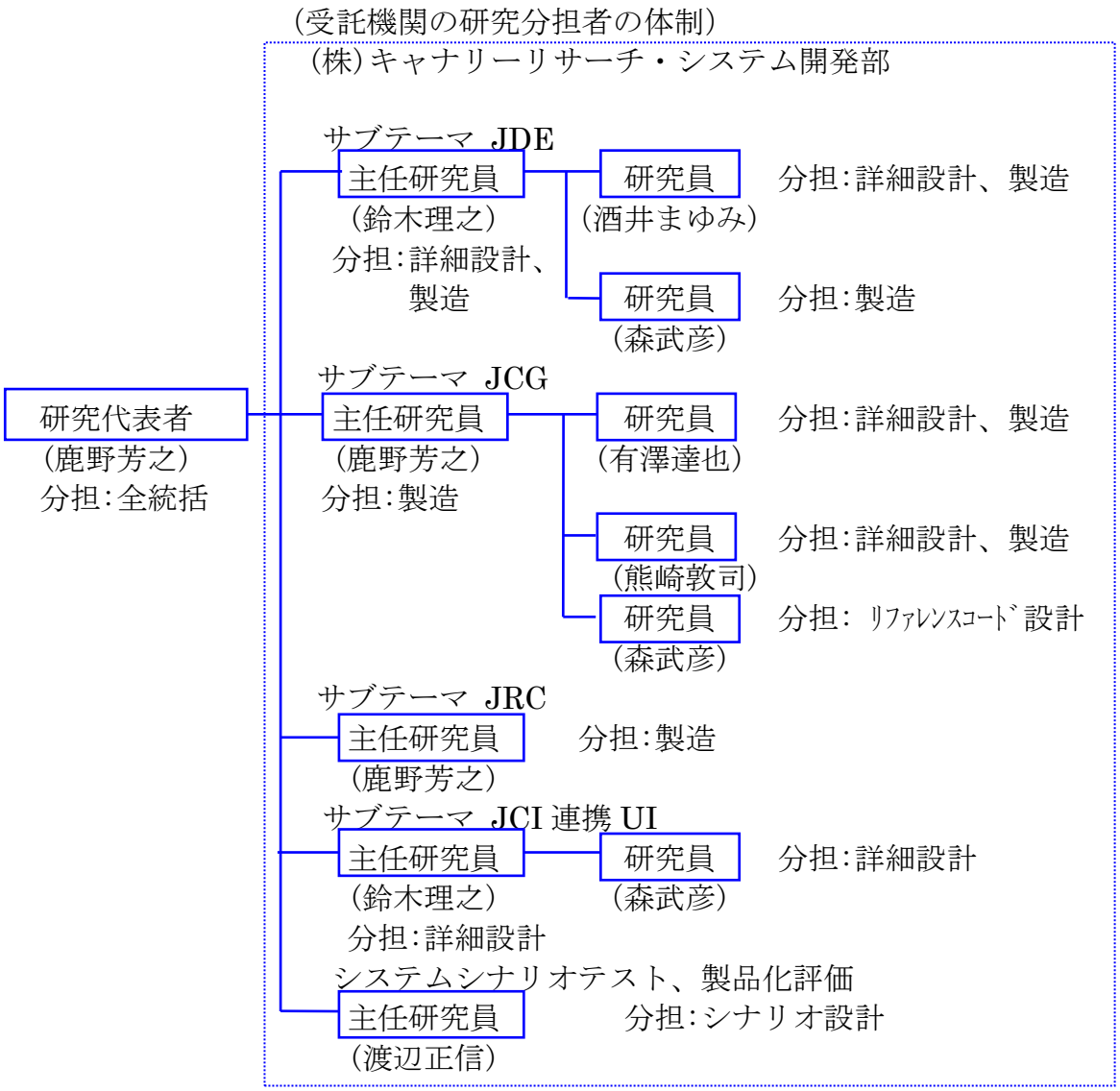

・詳細設計、検証
・製造
・単体検査
ウ　JRC の研究開発
・詳細設計
・製造
・単体検査
エ　JCI 連携 UI の研究開発
・詳細設計

・製造 ・単体検査 オ　システムシナリオ テスト、製品化評価 ・シナリオ設計 ・システムテスト、 製品化評価		―	― ― ―	― ― ―	
間接経費額（税込み）	―	―	―	―	
合　計	―	―	―	―	

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
　　２　備考欄に再委託先機関名を記載
　　３　年度の欄は研究開発期間の当初年度から記載。

3　研究開発体制

3－1　研究開発実施体制



４　研究開発実施状況

次の４つサブテーマを研究開発とした。
　JDE(Java Design Editor)
　JCG(Java Code Generator)
　JRC(Java Runtime Class)
　JCI(Java Code Inspector)の連携 UI
そして、本システムの製品化評価のために検査シナリオの設計、作成を行い、シナリオによるシステムテストを行った。

本システムは MVC モデル(※補足説明)によって設計、構築されている。本システムの JDE は MVC の View, 抽象構文木および Internal Form Generator（中間形生成系） は MVC の Model と Model 生成系、そして eclipse 連携機能と JDE との Interface 機能は MVC の Controller に相当する。この考え方、設計方針によって、1 年度から２年度までは JDE, JCG の基本機能を中心に開発を進めることができた。最終的には JDE の View と Controller, Model 間をコマンドで実装し、Model と JCG 間を API で実装することにより、より高品質な結合を実現することができた。１～２年度までと３年度の開発分担（製造内容）は、図２のように表現することができる。

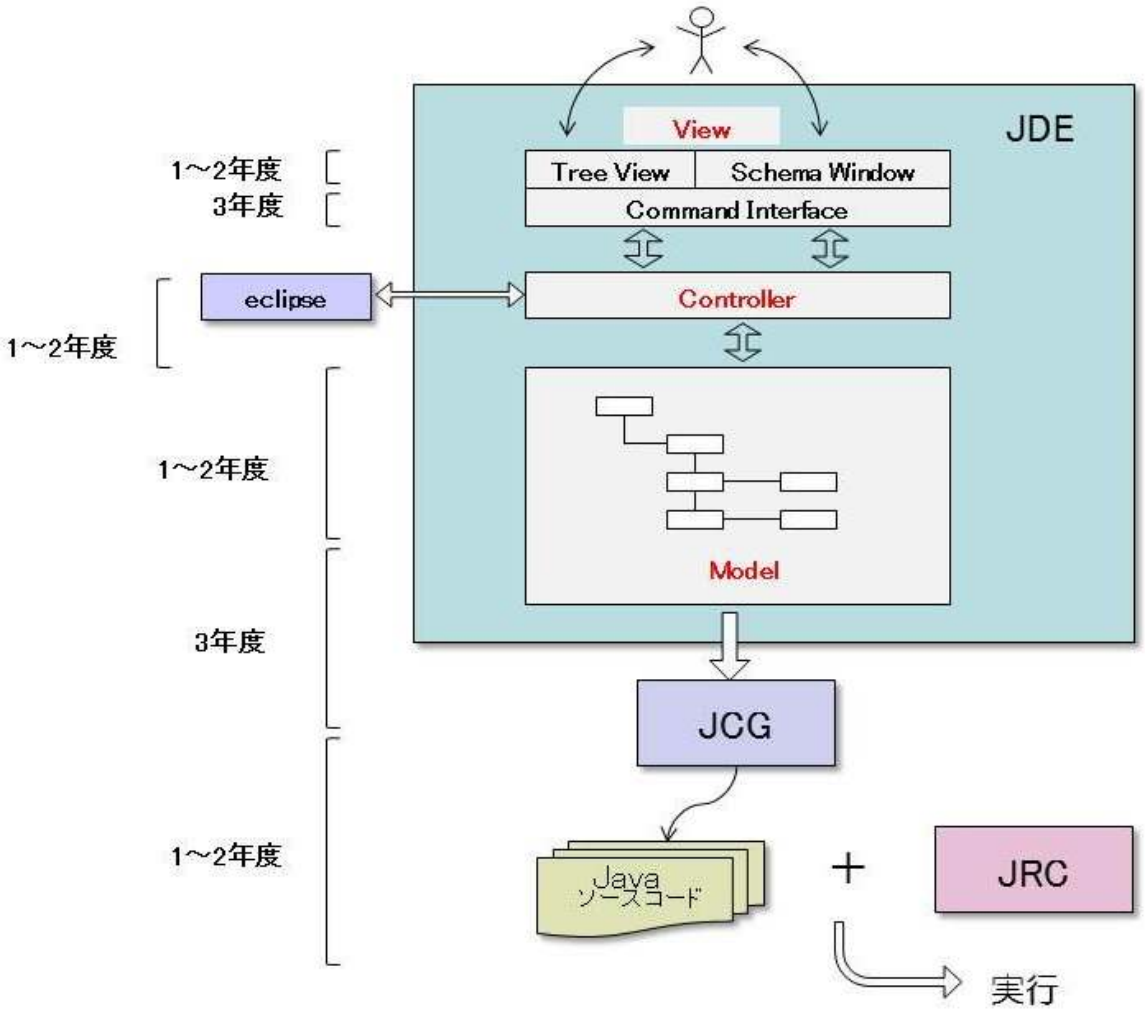
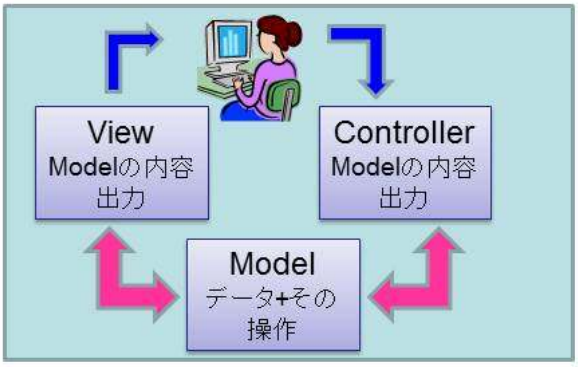


図２　MVC モデルによる開発分担

※補足説明：MVC(Model-View-Controller)モデル

- Model
  - 何らかのデータとそのデータを操作するためのメソッドを提供する役割を持つ。
    - Internal Form
- View
  - モデルのデータを加工、出力し、ユーザに見せる役割を持つ。
    - JDE全体（TreeView、スキーマウィンドウ、各種UI表示系）
- Controller
  - ボタンを押すなどのユーザからの入力を受け取り、それにしたがってモデルを操作したりする。
    - JDEとeclipse UIのインターフェイス
    - eclipse既存機能との連携



- Generator
  - **Model上に構築されたすべての情報からコード生成を行う**

2

４－１　JDE の研究開発

JDE の主機能となる、バッチコンポーネントのカスタマイズ機能を実現する UI および、JDE 基本機能の詳細設計を行い、これらの UI を eclipse 上に実装するために、eclipse の既存機能との連携、および実装のための UI 部品について調査、検証を行い UI ガイドラインを作成した。その後、それにしがって、JDE の基本機能について実装（製造）を行う。
次に、JDE が Model をアクセスする動作をすべてコマンドで実装する。JDE の保持するデータからコマンドにより、Controller 経由で Model のデータを作成、変更できるようにする。

４－１－１　詳細設計と製造

(1) 詳細設計
・バッチコンポーネントの詳細設計
　4 つのカテゴリに分類し、それぞれについて次の項目について詳細設計を行った。
　　・コンポーネント名
　　・アイコンのデザインおよび色
　　・リンク可否
　　・カスタマイズポイントおよびその UI
・JDE 基本機能の詳細設計
　一般的なエディタとしての機能の実装、およびバッチプログラムを設計するためのエディタを実装するための設計を行った。

(2) 定義ファイルの定義
JDE へのスキーマ入力となる、各種スキーマの定義ファイルについて、書式および定義内容を決定した。定義ファイルには次のファイルがある。
・表定義ファイル（データベースの DB 表のスキーマを定義）
・ファイル定義ファイル（外部フラットファイルの形式・構成を定義）
・項目置換ファイル（設計情報上の項目名置換方法を定義）

(3) 基本機能の製造

詳細設計書および UI ガイドラインにしたがい、JDE の基本機能の製造を行った。
主な製造内容について記す。

・TreeView
プログラム構造表示ツリー(TreeView)は、設計するプログラム構造をツリー状に表示するもので、基本的にはこのツリー上でプログラム構造の追加、削除等を行う。プログラムツリーの１つ以上のロジックノードがひとつのバッチコンポーネントを表す。アイコンや色で適切に識別でき、全体の構造が明確に把握できるようにした。

・スキーマウィンドウ
プログラムが使用するスキーマを、データベース表やファイル等に関係なく同一概念でしかも一覧表示するように実装する。入力側・出力側に分けて表示され、コンテキストメニューにより、そのスキーマの設定画面（カーソル定義の編集画面など）に移動することができる。

・項目値処理
入力項目の値を出力項目に代入する処理を簡単な UI で実装する。利用者はコーディングや変数名のタイピングといった作業をすることなく変数への代入や値の参照が可能となる。項目値代入、処理式代入、同名項目値代入がある。

・nbx ファイル管理
本ツールで作成した設計情報を保存するファイルを nbx ファイルとして定義する。JDE 終了時の表示位置、個人の環境等も保存の対象になる。JDE が読込み、作成、更新するためのファイル管理を実装する。eclipse のパッケージエキスプローラに反映される。

(4) バッチコンポーネントのカスタマイズ UI の製造

バッチコンポーネント毎に規定されているカスタマイズポイントを入力、編集するための UI を実装する。本ツールを用いての設計作業が効率よく行えるために重要な UI となるので、数名による使用評価（プロトタイピング）を十分に行った。

(5) 定義ファイルの読込と JDE 連携の製造

JCG の一機能として、定義ファイルの読込み、検索、JDE への結果送出について実装を行う。

(6) eclipse 既存機能との連携の製造

eclipse の既存機能を損なうことなく JDE の UI 機能を実装するために用いる、主な重要な連携動作の製造項目を示す。

✧ アクション、ウィザード
✧ メニューバー・ツールバー
✧ Undo, Redo
✧ 検索、カット・コピー・ペースト、ヘルプ

(7) JDE コマンド API、API による JDE 機能製造

すべての UI で必要な操作、必要なデータの種類、個数、属性等はすべて明らかである。これらをコマンド API(アクション指示書)として文書に定義し、それらの動作を Controller 部分に実装する。ここで、JDE 側のコマンド発行部分を Command Interface と定義する。すでに実現できている UI で取得したデータや表示すべき View のデータを、Controller 部分のコマンド API を用い、JDE 機能（Model とのデータのやりとり）を実装する。

(8) JDE View 部、JDE Controller 部単体検査

Treeview, スキーマウィンドウ、バッチコンポーネントカスタマイズ UI など、UI 部分の単体検査を行う。最初に JDE 機能仕様書から単体検査仕様書を作成し、レビュー後に単体検査を実施する。次にコマンド API による実装部分の単体検査を行う。アクション指示書から単体検査仕様書を作成し、レビュー後に単体検査を実施する。

(9) JDE デバッグと改修

単体検査ならびに、システムテストおよび製品化評価（後述）によって検出された不具合や不適切な仕様や実装を改修する。すべての検査が終了するまで行われる作業となる。

４－１－２　実施状況

(1) 詳細設計

詳細設計はすべて終了し、UI ガイドラインにしたがった実装方式を追記し、次の仕様書としてまとめた。これにより、本ツールのすべての UI 操作が統一されるので、操作に一貫性を保つことができる。

“バッチコンポーネント機能仕様書“
バッチコンポーネントの機能仕様を具体的な例を挙げ記述し、それぞれの実装を規定するために使用する UI 部品を指定している。表 1 に主な UI 部品を示す。

カテゴリ	コンポーネント	実装方式
Java 言語の構文要素	クラス定義	name box、修飾子:選択、型:選択
	インスタンス変数宣言	変数名追加、削除、変更、順序変更
	getter 定義	変数名追加、削除、変更
	setter 定義	変数名追加、削除、変更
	メソッド定義	name box, 型:選択、修飾子:選択

	if 文	if 式：フリー記述
	switch 文	switch 式：フリー記述
	for 文	形態:選択、for 式：フリー記述
	while 文	while 式：フリー記述
	do while 文	while 式：フリー記述
	continue 文	なし
	break 文	なし
	return 文	式：フリー記述
	synchronized	式：フリー記述
	ブロック	なし
	assert 文	評価式、エラー情報：フリー記述
データベース処理	DB 表入力ループ	スキーマ選択：menu, スキーマ名：menu
	コントロールブレーク DB 表入力ループ	スキーマ選択：menu, スキーマ名：menu ブレークキー：name box, pull down, menu
	DB 表出力	スキーマ選択：menu, スキーマ名：menu 種別選択、条件：フリー記述、項目値処理
ファイル入出力	FILE 入力ループ	スキーマ選択：menu, スキーマ名：menu open/close 指定：選択、フリー記述
	コントロールブレーク FILE 入力ループ	スキーマ選択：menu, スキーマ名：menu ブレークキー：name box, pull down, menu
	FILE 出力	スキーマ選択：menu, スキーマ名：menu 種別選択、open/close 指定：選択、フリー記述 項目値処理
	マルチレイアウト FILE 入力ループ	スキーマ選択：menu, スキーマ名：menu open/close 指定：選択、フリー記述
	マルチレイアウト FILE 出力	スキーマ選択：menu, スキーマ名：menu 種別選択、open/close 指定：選択、フリー記述 項目値処理
	マッチング FILE 入力	スキーマ選択：menu, スキーマ名：menu 種別選択、マッチキー：menu open/close 指定：選択、フリー記述
その他	フリー記述	フリー記述
	初期処理	なし
	終了処理	なし

表１　バッチコンポーネントの UI 部品の指定

“スキーマウィンドウ機能仕様書”

・スキーマ参照

スキーマを参照するすべてのコンポーネントから参照、アクセス可能とするために、参照ウィンドウを並列に表示する。スキーマ選択は一覧表示から選択、スキーマ項目はドラッグ&ドロップ対象とする。

・出力処理情報

スキーマ名は選択表示、処理種別は選択方式、条件はフリー記述とし、新たな

出力スキーマ名は別ウィンドウで選択する。

・項目値処理

代入処理を指定するUIなので、すべての指定がマウス操作で可能になるようにする。

項目値代入：入力側のスキーマと、出力側のスキーマを選択後、項目値代入ボタンで決定

処理式代入：出力側のスキーマを選択後、このボタンでフリー記述

同名項目値代入：入力側のスキーマと、出力側のスキーマで同名の項目すべてに対し、項目値代入を設定

(2) 定義ファイルの定義

フラットファイルについて、本システムで扱う書式を定義した。これにより、スキーマとして次の形式を使用することができる。定義する内容を記す。

表定義（Oracle, SQLServer, PostgreSQL DB表）

項目名、コメント名（日本語名)、DB型、長さ、Nullable、Primary Key

ファイル定義（テキスト、バイナリ、CSV、XML 各々について固定長、可変長形式)

項目名、コメント名、データ型、長さ（、精度)、書式1、書式2、…

(3) 基本機能の製造

JDEとして、スキーマ情報を選択、読込、バッチコンポーネントを組み合わせることによりプログラム構造を作成し、必要なカスタマイズポイントを入力、編集、そしてnbxファイルに保存する、という基本機能の実装(UIを中心とした製造)が完了した。
図３にJDEの基本画面を示す。通常は次の6個のビューから構成される。

1) Package Explorer

通常のeclipseでの環境同様に、作業対象のパッケージ構成図が表示される。作業対象のファイルをここから選択するとメインのエディタ画面に表示される。次の要素から構成される；

src

nbxファイル毎に自動生成されたメインjavaソースファイル名

JRE System Library

Referenced Libraries

nbxファイル毎に自動生成されたクラスライブラリファイル名

使用するJRCファイル名

nbx Files

JDEで作成、保存されたnbxファイル名

2) DB ツリー・ビュー

作業対象がデータベースに接続される場合、使用あるいは参照するDB表に関する情報が表示される。

3) スキーマウィンドウ

スキーマデータを用いてバッチコンポーネントをカスタマイズする場合に多く用いられるビューで、DB表、ファイル項目などを同一の概念(Javaの変数と同一視する)で、しかもマウスのドラッグ＆ドロップ操作で選択、指定を可能にしている。例えば、DB表やファ

イルのレコード読込後、任意の項目値を他の変数や別の DB 表、ファイルのレコード項目に代入するときは、項目代入ウィンドウにより、入力と出力項目を選択し、前者をドラッグし、後者にドロップするだけでその処理（操作）を完了できる。この時、代入の左辺と右辺の型チェックやレングスチェックも同時に行われるので、semantics の事前チェックも可能にしている。スキーマウィンドウのうち主なものを図４～図６に示す。

4) ジョブステップ
編集対象となっている nbx ファイルが使用しているスキーマ名およびそれらの入出力関係図。DB 表、ファイルおよび nbx ファイルから構成されるジョブフロー図とも言う。

5) TreeView
バッチコンポーネントから成るプログラム木を作成するための View を JDE 画面のほぼ中央に配置し、eclipse の基本操作で作成、編集ができるようにした。木構造を構成する最小単位はノードになるので、バッチコンポーネントを１つ以上のノードで構成、定義し、任意のバッチコンポーネントに対して「直前に挿入」「直後に挿入」「階層下に挿入」などのコントロールによってプログラム木を簡単に作成、編集可能とした。
バッチコンポーネントによっては、その直前、直後および階層下に挿入できるバッチコンポーネントが言語的に限定される場合は、挿入可能なものだけをメニュー表示することにより、syntax 的なエラーを事前排除している。

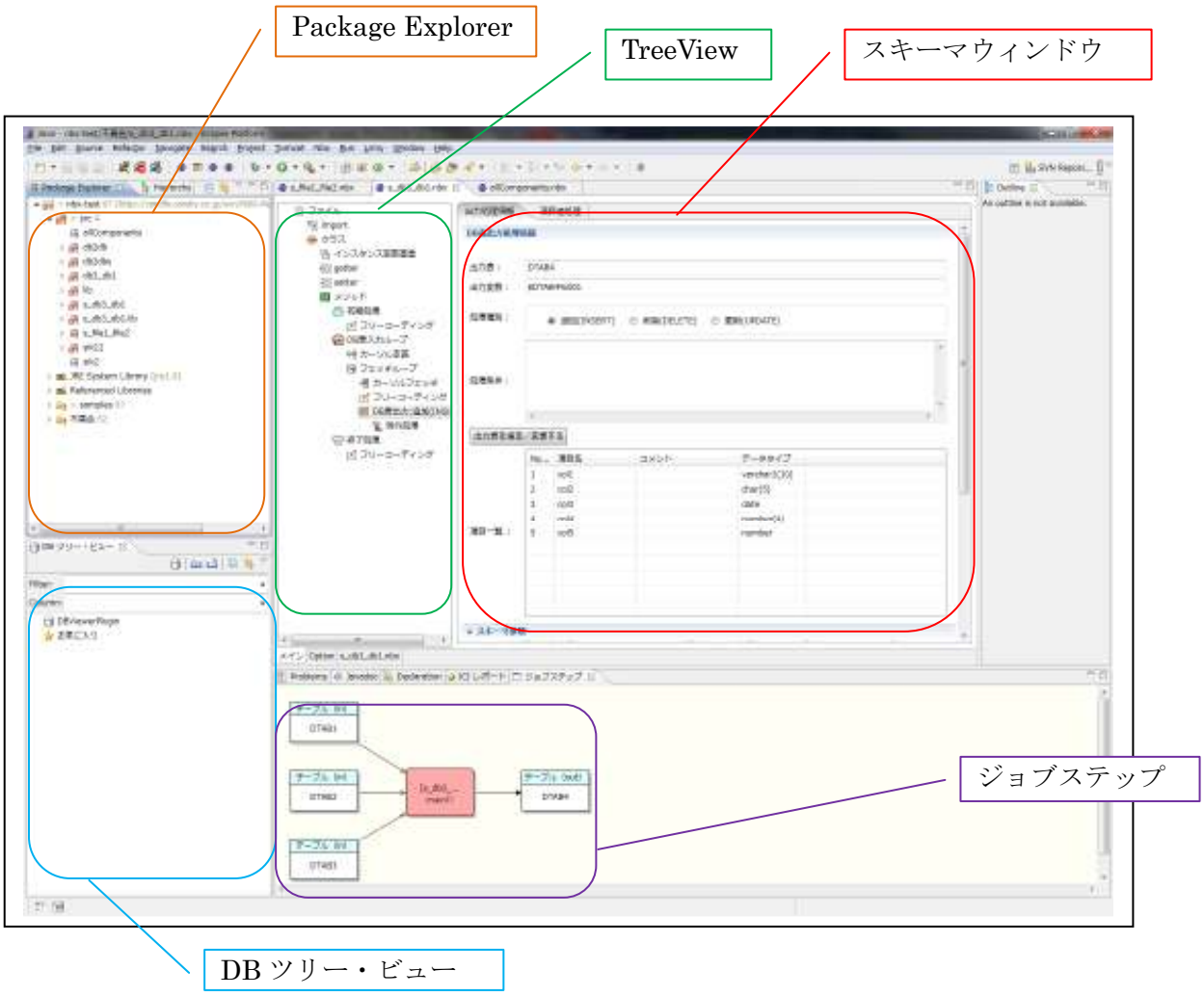


図３　JDE 基本画面

✧　スキーマ参照（図４）
スキーマ項目を参照、指定するバッチコンポーネントで、スキーマ項目を参照、選択するためのウィンドウ。次のスキーマを表示し、必要なスキーマを選択する。

登録済 DB 表
　スキーマ登録によって nbx ファイルに登録された DB 表
登録済 FILE
　スキーマ登録によって nbx ファイルに登録された FILE
入力変数
　スキーマを入力側に指定したとき自動生成される入力用に使用される変数
出力変数
　スキーマを出力側に指定したとき自動生成される出力用に使用される変数
インスタンス変数
　編集対象となっているクラスのインスタンス変数
ローカル変数
　編集対象となっているメソッド内で宣言された変数
メソッド引数
　編集対象となっているメソッドの引数

図４　スキーマ参照ウィンドウ

✧　出力処理情報（図５）
DB 表やファイルに出力する処理を指定するウィンドウ。次の要素から構成される；

出力表(FILE)
　出力対象のスキーマ（DB 表または FILE）
出力変数
　この出力処理に使用される自動生成された変数
処理種別
　DB 表の場合、INSERT or DELETE or UPDATE
　FILE の場合、OUTPUT or APPEND

図５　出力処理情報ウィンドウ

✧　項目値処理（図６）

対象とする出力スキーマに、出力値を代入する処理を行うウィンドウ。出力する項目とその出力値を設定するところで、次の要素から構成される。

入力スキーマ

　入力値があるスキーマを指定。スキーマ種別の選択に同期して、その種別に含まれるスキーマ名が表示される。

出力・入力スキーマ

　それぞれのスキーマがもつ項目名がデータ型、コメントとともに一覧表示される。

項目値代入

　代入処理、即ち、出力スキーマ.出力項目←入力スキーマ.入力項目を指定する。

処理式代入

　出力項目を指定して、代入文の右辺となる任意の式をフリー記述する。

同名項目値代入

　出力・入力で項目名が同名のペアについて、自動で項目値代入を設定する。

項目値一覧

　項目値処理ウィンドウで設定した項目値代入の一覧が表示される。オプション機能として、“代入取消し”、“すべて選択”、“すべて解除”をサポートする。

図６　項目値処理ウィンドウ

(4) バッチコンポーネントのカスタマイズUIの実装

バッチコンポーネントにはそれぞれ固有のカスタマイズポイントが用意されている。例えば、「出力DB表処理」では、出力表を指定し、その表の出力項目に値を代入するために、入力項目値等の値を指定する、などである。そして、それぞれのカスタマイズを必要最小限の操作で設定ができ、わかりやすくしかも容易にできることが望ましい。これらのポリシーにしたがい、これまで定義されているすべてのバッチコンポーネントのカスタマイズUIの実装を行った。例として、図７にassert文、図８にfor文のバッチコンポーネントのカスタマイズUIを示す。
(注：for 文のスキーマ参照ウィンドウは省略)

図７　assert 文バッチコンポーネントのカスタマイズ UI

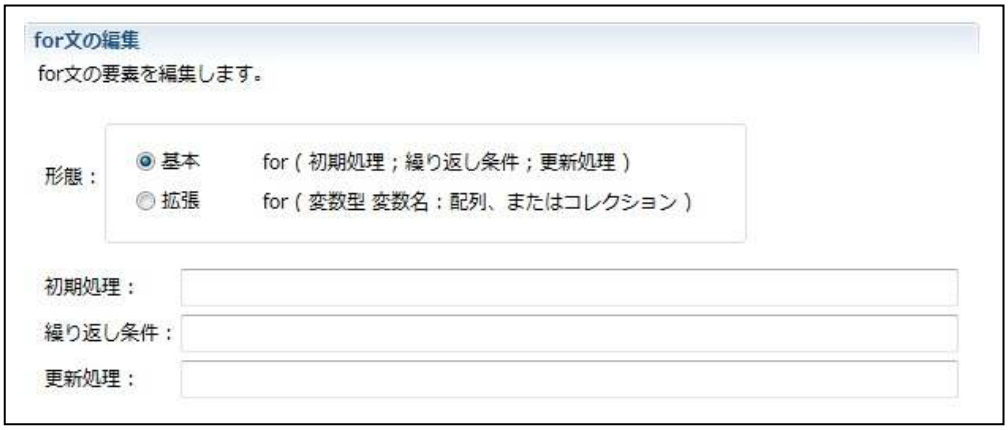


図８　for 文バッチコンポーネントのカスタマイズ UI

(5) 定義ファイルの読込と JDE 連携の実装

DB 表およびファイルのスキーマ情報を JDE に与えるための、JCG 内に用意される JDE 連携部分の実装を行った。

(6) eclipse 既存機能との連携の実装

eclipse ユーザは、もともとある eclipse の既存機能を変更したり、損なわれることを最も嫌う。そこで、既存機能を損なうことなく JDE の UI 機能を実装した。主な重要な連携動

作を行うための製造項目は以下のものである。

- ✧ アクション、ウィザード
  - ➢ nbx ファイルの新規作成、プロジェクトへの挿入、ビューの切替、カスタマイズなどを実装。eclipse の基本ウィザードを知っていれば容易に JDE 新規環境、nbx ファイル読込/保存などが行える。
- ✧ メニューバー・ツールバー
  - ➢ ファイルメニュー、基本操作、ツールの起動メニューなど、eclipse 既存メニューに追加するかたちで実装した。
- ✧ Undo, Redo
  - ➢ 連携機能の中で、最も重要な機能であり、BATOOL などの既存ツールにはないものである。バッチコンポーネントや TreeView, 項目値代入処理など、一連の意味のある動作のかたまりを単位とした Undo(操作の取消し)、Redo(操作の再実行)を実装した。
- ✧ 検索、カット・コピー・ペースト、ヘルプ
  - ➢ 検索については、検索対象が検出されたバッチコンポーネント単位で、結果を表示していく、という視覚的に明確な検索機能を実装した。
  - ➢ カット・コピー・ペーストもバッチコンポーネントやスキーマ項目単位に行えるよう、意味のある単位での操作を実装した。
  - ➢ ヘルプ機能は、本ツールをマニュアルなしでも使いこなせるように操作方法、順序を簡潔に記述したものを表示している。

(7) JDE コマンド API、API による JDE 機能製造

コマンド API(アクション指示書)として定義されたすべての動作を Controller 部分に実装し、JDE 側のコマンド発行部分を Command Interface と定義、実現した。すでに実装できている UI で取得したデータや表示すべき View のデータを､Controller 部分のコマンド API 経由で、JDE 編集機能（Model 自体の編集や Model とのデータのやりとり）を実装することによってエディタとしてのアクションをすべてコマンドの実行に置き換えることができた。この View と Model の分離により、View の変更や改善を Model に、Model の変更を View に影響なく行うことができるようになった。

(8) JDE View 部、JDE Controller 部単体検査

View と Controller の分離は、単体検査、および不具合の修正作業も効率化することができた。前者についてはエディタの機能仕様書から単体検査仕様書を作成、レビュー後単体検査を行った。後者は、コマンド API(アクション指示書)から単体検査仕様書を作成し、同様にレビュー後単体検査作業を行った。【JDE 部総検査項目数: 6,176】

(9) JDE デバッグと改修

前述の単体検査、システムテストおよび製品化評価（後述）に検出された不具合や不適切な仕様や実装方式を改修した。改修作業の度に類似箇所も含め再検査を行い、機能のダウングレードやエンバグ（バグを増加させること）がないことを都度確認していく。この作業を本研究開発において、「デグレード検査」と定義する。

４－１－３　実施計画に対する達成状況

JDE として必要とされる UI 部分の機能仕様、UI ガイドラインの設計後、JDE として計画された作業はすべて達成できた。特に、eclipse 連携の Undo, Redo については、予定工数の 2 倍以上の時間を費やしたが、要求機能を上回る実装を完了できた。今後の改版では、使い勝手を中心に、さらに UI 機能を改善していく予定である。

４－２　JCG の研究開発

JCG の詳細設計にしたがって JCG の実装を行った。目的は、JDE から与えられる設計情報からターゲットとなるコードを作成し、設計、試作、検証を行ったレファレンスコードに帰着するコード生成を行う。Model から実際に値を取りだし、コードテンプレートを実コードに変換し最終の Java ソースコードを生成する。さらに、生成コードの処理性能を計測、検証するための指標を定義する。

４－２－１　詳細設計と製造

(1) 詳細設計

・レファレンスコードの設計と検証
　コード生成系（以降、ジェネレータという）はその設計に先立ち、ターゲットとなる生成コードを決める必要がある。これをレファレンスコードといい、本ジェネレータが対象とするバッチパターンすべてについてコードを設計した。次に、そのコードの正当性を保証するために実際に試作し、ビルドを行い、実環境に近い状況で動作、性能の検証を行った。
・JCG の詳細設計
　次の項目を JCG の詳細設計の内容とした。
　✧　JDE から受け渡される全データの設計、決定
　✧　オブジェクト指向に基づくクラス抽出の設計
　✧　ジェネレータ基本機能の設計

(2) 中間形と抽象構文木の設計

JDE から与えられる設計情報を内部表現する中間形、コード生成の元となるプログラム構造の核部分を表現した抽象構文木の設計を完了する。

(3) JDE Interface の実装

JDE との情報をやり取りする部分で、次のような機能を実装する。
・データベース操作のうち、DDL に記述する値の構文解析
・SQL 文、SQL カーソルの組立てと分解と解析
・フリー記述部分に記述された Java ソースコードの構文解析
・生成コードの JCI による Code Inspection の実行および、JCI との連携

(4) Internal Form Generator の実装

JDE および外部ファイルからの設計情報とプログラム木から、コード生成に必要な中間形を作成する。次の機能を実装する。MVC モデルの Model（以降、M:モデル）の生成系である。

・オブジェクト指向に基づくクラス抽出
・クラスおよびメソッドの作成と、メソッド内のプログラム木の作成
・レファレンスコードを生成するための中間形の作成
・下位層の Code Generator との連携

(5) Code Generator 機能の実装

上位の中間形からコードを生成するために、コードテンプレートを利用したジェネレータ機能を次の 3 つに分けて実装する。
・ジェネレータ基本機能部分
・コード出力部
・バッチ一括生成部

(6) JCG Model API、API による最終の JCG 製造

Model からデータ構造や値を取りだす API を定義し、それらを Model 内に実装する。JDE の View と Model の関係のように、Model とコード生成部を疎結合とすることにより、互いに変更の影響を受けない構造が完成する。Model API を利用した最終的なコード生成部を実装する。Model が持つプログラム構造とデータ構造を API 経由で取得し、(5)で実装するコードテンプレートに沿って実値を与えることにより最終的な Java ソースコードを生成する。

(7) JCG Code Generator 部、JCG Model API 部単体検査

Internal Form Generator、コードテンプレート生成部の単体検査を行う。最初に、抽象構文木定義書、ジェネレータ定義-機能仕様書から単体検査仕様書を作成し、レビュー後に単体検査を実施する。続いて Model API による実装部分の単体検査を行う。単体検査仕様書を作成し、レビュー後に単体検査を実施する。

(8) JCG デバッグと改修

単体検査ならびに、システムテストおよび製品化評価（後述）によって検出された不具合や不適切な仕様や実装を改修する。すべての検査が終了するまで行われる作業となる。

(9) 動作検証と処理性能の計測のための指標の定義

動作検証と処理性能の計測のためにつぎの２つの指標を定義する。実際の検証作業は、後述のシステムテスト、製品化評価で行う。
【指標１】すべての種類のバッチコンポーネントを縦と横軸に取り、その可能なすべての組合せで　実行可能なコードを生成し、停止や無限ループ等の不正な実行パターンがないことを、実 DB 表や実ファイルを用いて検証する。
【指標２】シナリオ設計に基づき行われる製品化評価時に作成するプログラムと、既存製品である BATOOL 等で作成する同一シナリオによるプログラムとについて、実行速度、使用メモリを比較し、性能問題（手書きコード等と比較して、処理速度が劣るとか使用するメモリが多すぎるというような問題）がないことを検証する。

４－２－２　実施状況

(1) 詳細設計

JCG の設計では、入力データの変更に柔軟に対応可能であること、出力コード（レファレンスコード）の変更に柔軟に対応可能であること、が求められる。前者を実現するためにはデータを自然なクラス階層で表現することが必要である。また，後者を実現するためには前述のデータに対する処理を適切な形で分離することが必要である。したがって，JCG の構造としては、データを表現する内部形(Internal Form Generator)のクラス階層に、コード生成(Code Generator)処理を分離して持たせた構造が適切だと結論付けられる。JDE Interface 連携、Internal Form Generator, Code Generator について詳細設計を完了し、“ジェネレータ定義・機能仕様書”を作成した。

(2) 中間形と抽象構文木の設計

JDE の主機能を精査した結果、バッチコンポーネントの情報と、スキーマの情報に大別できた。バッチコンポーネントは画面上ではプログラム木として表現されるので、内部形としても木構造で表現することが自然である。また、スキーマの場合、ツール上では、種類によらない一様な操作性が求められるので、内部形としても種類によらない一様なインターフェイスを提供できることが望ましい。

まず、バッチコンポーネントについてはプログラム木への操作（ノードの追加・削除等）が容易に行えるように、コンポジット構造とし、各バッチコンポーネントに対応するクラスをサブクラスとして表現するクラス階層とした。これにより、プログラム木の各ノードはクラスで表現され、同じクラスとして扱うことができるので、木構造に関する操作は容易である。また、今後のバッチコンポーネントの追加・変更にも柔軟に対応することができる。
つぎに、スキーマについては各スキーマに対応するクラスを一様に扱うためのインターフェイスを定義し、スキーマの種類毎にサブクラスを作成する構造とした。また、各スキーマの有効範囲を管理するクラスを設け、プログラム木との対応も管理している。以上により、こちらについてもバッチコンポーネントと同様にスキーマの追加・変更に強い構造とすることができた。

以上の観点で、JDE から与えられる設計情報を内部形で保持するための M:モデルの設計を完了した。M:モデルは、中間形と抽象構文木から構成されることになる。

(3) JDE Interface の実装

次の機能について実装を完了した。
・データベースの DDL に記述する値の構文解析
　INSERT 文、UPDATE 文の value 句に記述される式を解析し、文法的に文か値かを構文解析するパーザ。
　　例：INSERT (a, b, c) into Table-X values(“xyz”, jx*10+hh, 15);
　　ここで、”xyz”, 15 が値で、jx*10+hh が式であると解析される。

・SQL 文、SQL カーソルの組立てと分解と解析
　カーソル定義やDB 表出力バッチコンポーネント等に指定される SQL 文を組み立てたり、

ユーザが記述したSQL文を分解や解析することにより、最適化されたSQL文を実行するためのクラスを生成するための準備を行う。
例：SQL文の最適化
複数の表が結合(join)される時、joinされる表をうまく選択すると実行効率が高まる。そのためのHINT句もサポートしている。

・フリー記述部分に記述されたJavaソースコードの構文解析
フリー記述コードから、本システムが扱うスキーマ項目を抽出し、内部変数に変換するパーザ。構文解析をすることにより実装することができる。
例：xx = $table1.abc + abc / $instance.jj; について xx, abc はユーザ定義の変数で、その他はスキーマ項目であることを解析する。

・生成コードのJCIによるCode Inspectionの実行および、JCIとの連携
自社開発による製品であるJCI を組入れ、コード生成後のJavaソースに対してCode Inspection を実施、不適切なコードや記述を検出した場合、JDE の対応画面で表示させる、JCIとの連携動作のためのJDEとのインターフェイスの実装を行った。

(4) Internal Form Generator の実装

・オブジェクト指向に基づくクラス抽出
オブジェクト指向に基づいたコード生成ができるように、設計情報からクラスを抽出する。
・クラスおよびメソッドの作成と、メソッド内のプログラム木の作成
実際に生成されるクラス、そのクラスに必要なメソッド、およびメソッド内の実行文のもとになるプログラム木を生成する。
・レファレンスコードを生成するための中間形の作成
プログラム木から、内部でもつレファレンスコードテンプレートを選択し、M:モデルとしてもつべき情報を保持する中間形を作成する。
・下位層のCode Generatorとの連携
M:モデルから実際のJavaコード生成を行うため、下位層のCode Generatorを起動するための環境の作成を行う。

実際にJCG内に作成されるM:モデルを図9に示す。

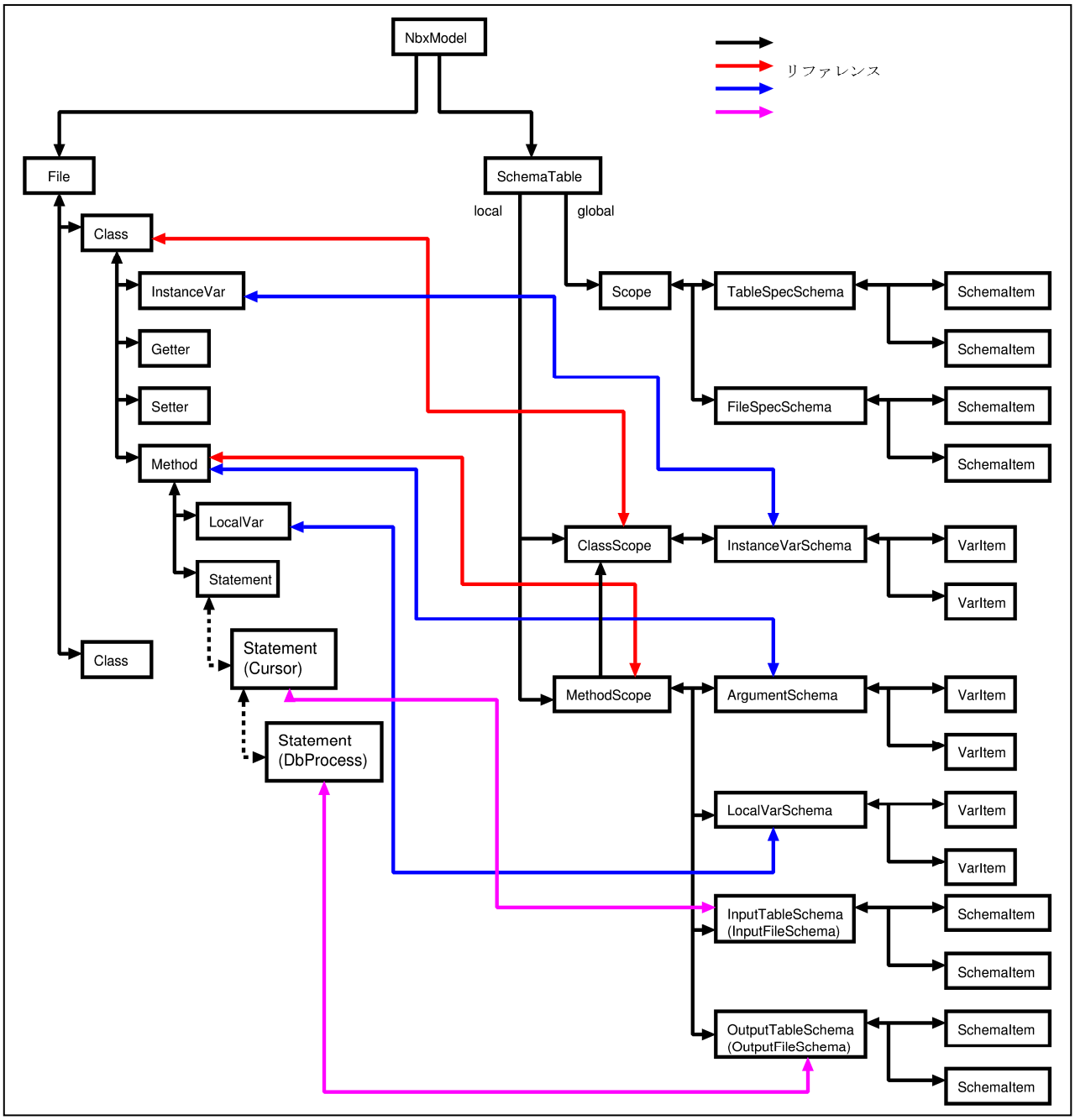


図９　本システムの M:モデル

(5) Code Generator 機能の実装

コード生成処理は前述の各クラスにそれぞれ分離した形で配置できる。これにより、レファレンスコードに修正が必要な場合でも容易に対応することが可能である。また、将来的に異なるレファレンスコードを生成する場合でも、大きく形を変えることなく対応することが可能である。以上の結果，JCG は，バッチコンポーネントを表現するクラス階層とスキーマを表現するクラス階層で自然に構成することができた。

・ジェネレータ基本機能部分
　レファレンスコード毎に、コードテンプレートを生成するコード生成部。
　例として、図 10 にデータベース処理時に生成するクラスのコードテンプレートを示す。

```
public class CusorNameReader extends DatabaseReader{
     public CursorName(){
            stmt = con.prepareStatement("SQL 文");
     }
     public ResultSet getCursorName(){
            return stmt.excuteQuery():
     }
}

public class CursorName extends Cursor{
     private TypeName VarName
     . . .
     public void setData(TypeName ArgName, . . . ){
            this.VarName = ArgName;
            ...
     }

     public TypeName getVarName(){
            return this.VarName;
     }
     ...
  }

public class ProgramNameWriter extends DatabaseWriter{
     private PreparedStatement pstmt;

     public void writeData(Type item, ... ){
            pstmt.setXXX(item);
            ...
            pstmt.addBatch();
     }
     protected void open(){
            pstmt = con.prepareStatement("SQL 文");
     }
}
```

図 10　データベース処理クラス

・コード出力部
　指定された eclipse のプロジェクト内のソースファイルに、実際のソースプログラムを出力する部分。前述した、Package Explorer に表示される src に配置されるメインクラスファイルと Referenced Libraries に配置されるライブラリクラスとして出力される。
・バッチ一括生成部
　本システムを実際のバッチシステム・プロジェクトに適用した場合、多数の nbx ファイルが作成されることになる。SI(System Integration)作業の効率化のために、複数の nbx ファイルから一括でコード生成を行うメカニズムを実装した。

(6) JCG Model API、API による最終の JCG 製造
Model からデータ構造や値を取りだす API を Model 内に実装し、この API を使用した最終

的なコード生成部を実装した。コードテンプレートに沿って API 経由で取得した実値を与えることにより Java のソースコードを生成する。Model と Java 言語にはほとんど依存関係がないので、他の言語での生成も短期間で実装可能になっている。

(7) JCG Code Generator 部、JCG Model API 部単体検査

JDE Interface、 Internal Form Generator、Code Generator、Model API による実装部分の単体検査を実施した。JDE 同様にそれぞれが独立なので、不具合修正や改修作業も並行に行うことができた。【JCG 部総検査項目数: 3,585】

(8) JCG デバッグと改修

前述の単体検査、システムテストおよび製品化評価（後述）に検出された不具合や不適切な仕様や実装方式を改修した。改修作業の度に類似箇所も含めデグレード検査を行った。

(9) 処理性能の計測と検証指標の定義及び検証

・前述の(7)の検査項目の網羅性を規定するために設けたのが指標１である。これは、すべてのバッチコンポーネントを縦と横に取り、その可能なすべての組合せで実行可能なコードを生成し、実行検査評価を検査漏れなく実施するための指標である。
・指標２によって、既存製品である BATOOL と本ツールで同一の業務、処理を設計、実装し、比較評価することにより、現実の使用に十分耐えうる製品かどうかの定量的な評価を実施することになる。同時に処理性能評価のための指標である。

４－２－３　実施計画に対する達成状況

計画した作業はすべて達成できた。最初に、M:モデルと Code Generator は並行分離開発され、次に、M:モデルからデータを取得する API を開発することにより、M:モデルと Code Generator が疎の関係で連結されたことになる。この API を用いることで、異種言語の Code Generator を分離開発することができるので多種多様な JCG を実現することができる。

４－３　JRC の研究開発

JRC クラス群の機能設計、詳細設計を行い、それにしたがって、ランタイムクラスを実装する。実際のデータベース表、外部データファイルを用い実行速度、メモリ空間効率等の検証作業を行う。

４－３－１　詳細設計と製造

(1) JRC の詳細設計

データベース処理、ファイル操作処理、例外処理およびユーティリティに分けられ、各処理について、次のランタイムクラスの詳細設計を行った。

A) データベース処理
B) フラットファイル処理
C) 例外処理およびユーティリティ処理

(2) データベース処理クラスの実装

フェッチクラス、DDL クラス、バルク処理クラス、Connection (Pool)クラスの製造を行う。

(3) フラットファイル処理の実装

Java, C 型バイナリファイル、テキスト、CSV ファイル I/O クラス、および
マルチレイアウトファイル I/O クラス、XML ファイル I/O クラスの製造を行う。

(4) 例外処理およびユーティリティ処理の実装

例外処理およびユーティリティ処理クラスの製造を行う。

(5) 単体検査

ランタイムクラス詳細設計書から単体検査仕様書を作成し、レビュー後に単体検査を実施する。

(6) デバッグと改修

単体検査ならびに、システムテストおよび製品化評価（後述）によって検出された不具合を改修する。特に製品化評価のうち、実行速度、処理性能評価に重点を置く。

４－３－２　実施状況

(1) JRC の詳細設計

すべてのランタイムクラス（データベース処理クラス、ファイル処理クラス、例外処理およびユーティリティ処理クラス）の詳細設計の完了後、本格的に JRC を実装した。

(2) データベース処理クラスの実装

データベースを扱うクラスとして、次のクラスを製造完了した。

Database:
　データベース接続用の抽象クラスである。データベースへの接続、切断等を行う。
　実際のデータの読み書き等は行わず、それらはサブクラスで行われる。
Databaseinfo:
　データベース接続情報を保持するクラスである。下記の情報を保持する必要がある。

- ドライバ情報
- データベースの URL
- 接続ユーザ ID
- 接続パスワード

これの情報はサブクラスで設定を行う。 また、サブクラスはシングルトンとすることを推奨する。Database オブジェクトは，DatabaseInfo オブジェクトの情報を利用してデータベースへの接続を行う。

DatabaseReader:
データ読み込み用のデータベース接続クラスである。カーソル毎にサブクラスを作成する。実際の読み込み処理はサブクラスで行われる。

DatabeseWriter:
データ書き込み用のデータベース接続クラスである． 書き込みを行うデータ集合毎にサブクラスを作成する。実際の書き込み処理はサブクラスで実装される。

Cursor:
データベース/ファイルからデータを取得するためのカーソルのスーパークラスとなりうるクラスである。通常。カーソル毎に対応するクラスを Cursor クラスのサブクラスとして定義する。 データベース/ファイル両アクセス用のメソッド定義されているのが、最低限対応するメソッドだけを定義すればよい。 このメソッドでは、データベースの場合は ResultSet オブジェクト、ファイルの場合は FileLine オブジェクトからカーソル用の値を取得する。それぞれの値の型や個数は異なるので、それらはサブクラスで getter メソッドとともに定義する。

データベース用カーソルクラスの典型的な例はつぎのとおりである。

```
public class CursorName extends Cursor{
    private String ItemName;

    public CursorName(){
        this.ItemName = null;
    }

    public void setData(ResultSet rs) throws SQLException{
        this.ItemName = rs.getString("ItemName");
    }

    public String getItemName() {
        return ItemName;
    }
}
```

MultiCursor:
マルチファイルからデータを取得するためのカーソルのスーパークラスとなるクラスである。

以上のデータベースクラスを図 11 のクラス図で示す。

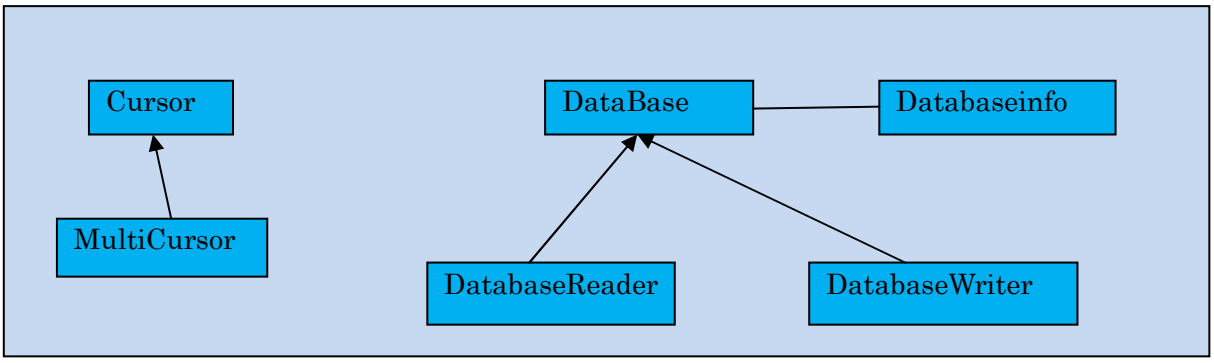


図 11　データベースクラス

(3) フラットファイル処理の実装

本ツールが扱うことができるすべてのファイルの種類について、次のファイル I/O のクラスを製造した。

File:
　入出力用ファイルクラスのスーパークラス
FileDef:
　ファイル定義用クラス。ファイル定義にしたがって設定を行う。
ItemFile:
　項目定義用のクラス
NormalFileDef:
　ノーマルファイル用のファイル定義クラス
MultiFileDef:
　マルチファイル用のファイル定義クラス
FileReader:
　読み込み用のファイルクラス。
　読み込み用のファイル毎にサブクラスを用意する。各サブクラスではファイル定義
　クラスを利用して、ファイル構造の定義を行う。 ただし、マルチファイルや
　マッチングループで利用されるファイルの場合は専用のクラスを利用する。
MultiFileReader:
　読み込み用マルチファイルクラスのスーパークラス
MasterTranFile:
　マッチループ用ファイルクラスのスーパークラス。
　各ファイル毎にサブクラスを用意する。
FileWriter:
　書き込み用のファイルクラス。
　書き込み用のファイル毎にサブクラスを用意する。各サブクラスではファイル定義
　クラスを利用して、ファイル構造の定義を行う。 ただし，マルチファイルの場合は
　別のクラスを利用する。
MultiFileWriter:
　書き込み用マルチファイルクラスのスーパークラス

以上のファイル処理クラスを図 12 のクラス図で示す。

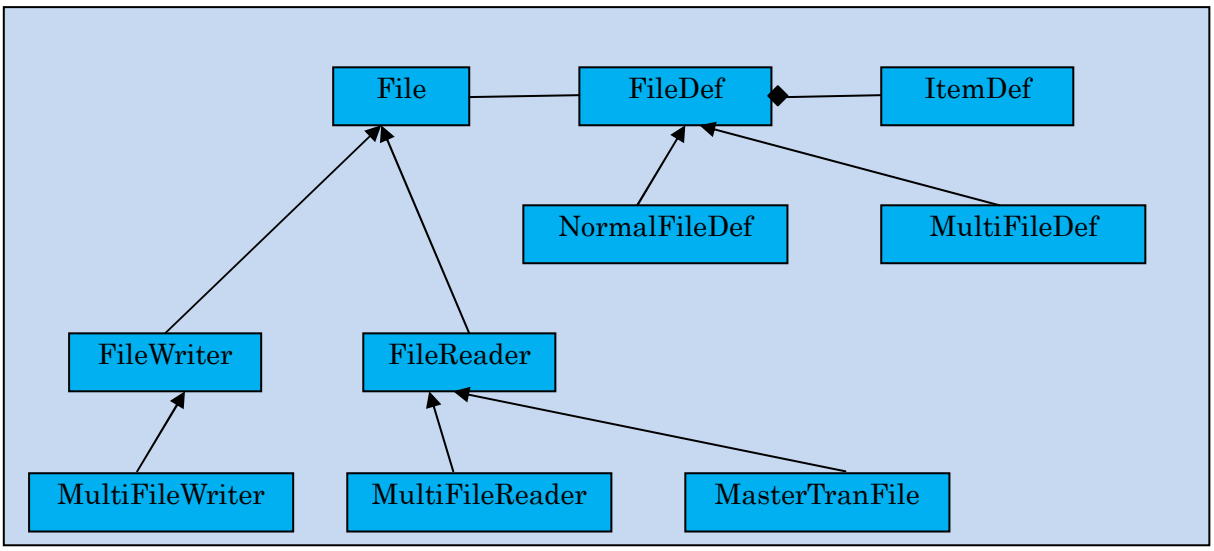


図 12　ファイルクラス

(4) 例外処理およびユーティリティ処理の実装

本ツールで生成されたコード内で発行されるすべての例外を処理するクラスと、その他共通処理（時刻取得、ログ出力、等）のユーティリティクラスを製造した。将来、ランタイム機能としての要求があればこれらのクラスに追加実装することになる。

(5) 検査と改修

前述したクラス図と“JCI 実行環境仕様書（詳細設計書)”から単体検査仕様書を作成し、レビューの後、検査を実施した。データベースとしては、Oracle 11g と PostgreSQL を用い、プラットフォームは Linux と Windows を利用した。【JRC 部総検査項目数:196】

４－３－３　実施計画に対する達成状況

計画した作業はすべて達成できた。予想よりコンパクトに実装され、構造もオブジェクト指向に基づいているので、バージョンアップ時等に予定されている既製のバッチフレームワーク対応時でも変更はほとんど不要である。当面、このランタイムクラスは、本ツールのデフォルトのフレームワークとして提供（リリース）される。

４－４　JCI の連携 UI の研究開発

すでに弊社にて開発済みである JCI と、本ツールとの連携 UI の詳細設計と実装を行う。本システムの特徴である「ジェネレーション&インスペクションサイクル」を実現する。
達成目標としては、

- JCI はすでに製品化された eclipse プラグインであり、本研究開発のための新たな改造や変更は皆無にすること。
- JCI は本研究とは非同期にバージョンアップ等が行われるので、常に最新版を取込むメカニズムを備えること。
- JCI からのアウトプットのみで JDE との連携が行えるインターフェイスにすること。

４－４－１　詳細設計と製造

(1) 詳細設計
設計項目は次のとおりである。
- ✧ JCI との結合に関する検討および調査
- ✧ JDE 上へのジャンプ機能
- ✧ ソースにマークをつける機能
- ✧ 設定画面のプリファレンスページ化

(2) 製造

次の機能を実装する。
- ✧ JCI が指摘したコード部分を生成する入力となった JDE の画面へのジャンプ機能
- ✧ 上記入力から生成されたコード部分にブロックマークをつける機能
- ✧ JCI の設定画面のプリファレンスページ化機能

(3) 単体検査

JCI の指摘項目がすべて網羅されるソースコードを生成する NBX ファイルを作成し、すべての指摘ポイントから単体検査を実施する。

(4) デバッグと改修

単体検査ならびに、システムテストおよび製品化評価（後述）によって検出された不具合を改修する。

４－４－２　実施状況

(1) 詳細設計

・JCI との結合に関する検討および調査
JCI は本ツールと同様の eclipse のプラグインである。通常は本ツールと同じレベルでインストールされるので、JCI との結合は並列に存在するプラグイン間の JCI からの情報（解析結果）の取得方法が結合に関する検討事項となった。JCI の解析結果は CSV ファイルとして出力されるので、本ツールがこの CSV ファイルから情報を取得することにした。図 13 に示した結果の出力例のように、ソースファイル名、行、記述の情報を読み取ることができるので、生成コードとマッピングを取ることが可能である。

```
開始日時,2011/4/6 19:42:34
終了日時,2011/4/6 19:42:34
総件数,15
give up fork num,0
give up expression num,0
"検査項目","重要度","メッセージ","ソースファイル","行","記述"
"0702","高","無限ループの可能性があります
","/nbx-test/src/s_file1_file2/s_file1_file2.java","31","while(mAT_CSV1.next())"
"0704","低","ループを抜ける箇所が複数存在します
","/nbx-test/src/s_file1_file2/s_file1_file2.java","31","while(mAT_CSV1.next())"
"0706","中","switch の case 文に break 文，return 文が共にありません
","/nbx-test/src/s_file1_file2/s_file1_file2.java","34","case 1: "
"0706","中","switch の case 文に break 文，return 文が共にありません
","/nbx-test/src/s_file1_file2/s_file1_file2.java","45","case 2: "
```

図 13　JCI の解析結果の CSV ファイル

・JDE 上へのジャンプ機能
JCI 指摘ポイントから、そのコード生成する元になったバッチコンポーネントを特定することができれば、JDE にそのコンポーネントを通知し、自動でそのカスタマイズ UI を開くメカニズムを実装すればよい。ジェネレータは、各バッチコンポーネントから生成されたコード部分がどこから、どこまでであるという情報を保持しているので、実現可能である。次に、JDE に自動で指定のバッチコンポーネントのカスタマイズ UI を開くメカニズムの実装が必要である。さらに、JCI の解析結果である″記述″から、そのコンテンツを入力したウィンドウまたは UI コントロールを特定する必要がある。

・ソースにマークをつける機能
カスタマイズ UI を特定できれば、その中の″記述″を検索し、適当なマーキング（反転等）を行えばよい。同一のコンポーネントに、複数の″記述″がある場合、すべてにマーキングをする必要があり、順番に検索できる機能も実現したい。

・設定画面のプリファレンスページ化
これは JCI の実行や解析レベル等の設定を本ツールのプリファレンスページから行えるようにすることである。JCI も本ツールも eclipse のプラグインなので、プリファレンスページを共有することが可能である。つまり、本ツールのプリファレンスページから、JCI のページにジャンプするような機能で実装を行えるはずである。

以上の詳細を、“JCI 連携動作仕様書”としてまとめた。

(2) 製造

JCI の結果から、JDE の画面をアクティブにし、該当コードを生成しているバッチコンポーネントを表示する機能は実装できた。ユーザが手書きしたコードを指摘することが最も重要な目標であり、その機能自体は実装できたものの、JCI からの不要な指摘項目については非表示にする機能が求められた。換言すると、JCI のすべての検出ポイントから JDE にジャンプする必要はないので、この連携動作改善を次期のバージョンアップ機能項目の１つとした。

(3) 検査と改修

JCI の指摘項目がすべて網羅されるソースコードを生成する NBX ファイルを作成し、すべての指摘ポイントから連携動作の単体検査を実施した。これらの検査ならびに、システムテストおよび製品化評価（後述）によって検出された不具合は改修できた。
【JCG 部総検査項目数: 112】

４－４－３　実施計画に対する達成状況

計画した設計作業はすべて達成できた。実際に JCI プラグインを eclipse にインストールし、本ツールで生成した java プログラムに対し、インスペクションの実行を行い、解析結果の指摘ポイントから JDE への該当コンポーネントへジャンプすることも確認できた。これにより、本ツールの目指す「ジェネレーション&インスペクションサイクル」は実現できたといえる。

４－５　システムテスト、製品化評価

本システムのシステム検査、製品化評価時に使用する、実際のバッチ要件に極めて近いプログラム仕様をシナリオとして定義し、本ツール自体を用いて複数の設計情報(nbx ファイル)およびそのシナリオで使用するデータの準備、生成ツール、さらに実行環境から構成されるシナリオキットを作成する。それらを用いシステムテストを実施する。そして、製品としてあるべき機能、性能、使用感などの評価を行う。

４－５－１　設計と作成

(1) シナリオ設計

シナリオには入力・出力データ仕様、データベース仕様、ファイル仕様に加えて、ビジネスロジック、加工処理仕様、処理性能仕様、例外処理仕様などが含まれる。これらを、21 年度に行ったレファレンスコードの全てのパターンについて設定、考案したシナリオを設計する。表２にレファレンスコードのパターンを示す。

パターン	入力	出力	主な業務ロジック
DB-DB	DB	DB	基本、集計、トランザクション
DB-FILE	DB	FILE	集配信ファイル作成
FILE-DB	FILE	DB	集配信ファイル読込、日時処理
CntrlBrk-DB	DB	-	ソート、多段階集計
CntrlBrk-FILE	FILE	-	ファイルソート、多段階集計
MultiLayout-FILE IN	FILE	-	可変長ファイル読込
MultiLayout-FILE OUT	-	FILE	可変長ファイル作成
FileMatching 1:1	FILE	-	ファイル照合 1:1
FileMatching 1:N	FILE	-	ファイル照合 1:N

表２　レファレンスコードのパターン

ここで、

DB:データベース表、FILE:外部ファイル、- :形式問わず

を表す。

(2) システムテスト

シナリオには入力・出力データ仕様、データベース仕様、ファイル仕様に加えて、ビジネスロジック、加工処理仕様、処理性能仕様、例外処理仕様などが含まれる。各シナリオ毎に、レファレンスコードのパターンが該当するすべてについて実行検査を行う。RDBMS には Oracle 11g および PostgreSQL を用いる。

(3) 製品化評価と反映

最終的に製品として問題がないか、前記した性能検証のための指標および以下の観点から評価を行う。

✧ システム全体の GUI の完成度、使用感とツールとしての支援能力
✧ バッチ処理性能（処理速度と処理能力）
✧ バッチ処理記述能力
✧ オンラインヘルプや提供するドキュメントの整備

できるだけ定量的に評価を行い、指摘される問題点、検出される不具合の改修は、
各サブテーマの研究開発最終作業として計画する。

４－５－２　実施状況

まず、実業務をヒアリング、調査を行い、実際のプロジェクトの要件をバッチ要件としてまとめ仕様書にした。ここで対象とした実業務は、あるコンビニエンス・ストアのバッチ処理を中心としたものである。そして、それをもとに本システムのシナリオテストを行うベースとなる方式設計を行った。最後に、この方式設計から、前述したレファレンスコードのパターン毎に、できるだけ実業務に近いシナリオで、一連の処理が連続して実行できるようなシナリオ設計を行った。実際の店舗用に実施されるバッチ処理の代表的なシナリオを例示する。(実際の設計書には、数値や諸元、件数、処理時間等が指定されている)

シナリオー1　集配信ファイルからのマスタ更新
月次処理として行うバッチで、まず 1 つのファイルが送信され、そのコンテンツによって 7 種類のどのマスタ DB 表のどの表を更新するかが通知される。次に、指定マスタの更新データが集配信ファイルとして与えられるので、全件を読出し、該当するマスタ DB 表を更新する。更新に失敗した場合は、全件ロールバックを行い、更新無しとして終了する。本ジョブは、先頭のマスタを決めるジョブステップと 7 種類のマスタ更新ジョブステップから構成される。なお、集配信ファイルのサイズは未定なので、時間内に完了するかどうかの判定も必要である。

シナリオー2　トランザクション処理
バッチとしては典型的なトランザクションの処理。1 日に発生した全売上データから、次の処理、集計を行うこと。

・店舗ごとの総売り上げ、総個数、総返品数
・費目ごとの総売り上げ、総個数、総返品数
　（費目は、13 桁の Jancode の下 4，5 桁目で分類する）
・在庫データベースから、在庫数がある一定数を切ったものに関して、マスタに指定されている量の発注を申請する DB 表を更新する
・日時レポート DB 表を作成する

シナリオー3　マッチング処理
店舗 A の在庫なし品目と、店舗 B の在庫あり品目をすべて照合し、店舗 B から店舗 A に配送可能な品目を抽出し、取寄せる店舗情報と移送数とともに取寄せファイルを生成する。同じ処理を、店舗 A の在庫なし品目がなくなるまで、店舗 C, D, …と照合を実施する。結果を適当な帳票として出力すること。本ツールには帳票を作成するツールが未だ実装されていないので、最低限機能としてフリー記述で生成すること。

シナリオー4　N段のコントロールブレーク処理
バッチ要件に含まれていたコントロールブレーク処理は、ブレークの段数が１～３であることから、最大で３段のブレーク処理を検証する必要がある。そこで、表３のように、3段の分類に応じた集計を行うシナリオを用意した。ただし、バッチコンポーネントのうち、コントロールブレーク入力ループを使用すれば、入力は DB 表でもファイルでもよい。

大分類	中分類	小分類	小計	中計
大分類１	中分類１	小分類１		
		小分類２		
		小分類３		
	中分類２	小分類４		
		小分類５		
		小分類６		
	中分類３	小分類７		
		小分類８		
		小分類９		
大分類２	中分類１	小分類１		
		小分類２		
	中分類２	小分類３		
		小分類４		
			大計	

表３　３段階のコントロールブレーク処理例

４－５－３　実施計画に対する達成状況

計画した作業（シナリオキットとしては 30 セットを実装・作成）はすべて達成できた。シナリオキットとして、
- DB 表使用のケースでは、表の DROP,CREATE の SQL 文、およびテストデータがある場合はデータと INSERT の SQL 文。
- ファイル使用のケースでは、ファイル作成のスクリプト、およびテストデータがある場合はデータとデータを挿入するためのプログラム。
- 出力されるべき結果の DB 表、ファイル、テキストファイルなどの被比較結果媒体と結果比較のためのスクリプトあるいはプログラム、そしてエビデンスを生成するスクリプトあるいはプログラム。

を１キット毎に実装、整備した。
将来的に、このシナリオに基づくシステムテストは、システム自体のバージョンアップの度に実行すべきデグレードテストキットとして用いることができる。

(2) 検査実施状況

表４にサブテーマ毎の検査実施状況をまとめる。現時点で不具合はすべて改修済みである。（不具合数には改善項目も含まれ、結果的に改修作業の対象となった項目数である）

サブテーマ	検査項目数、検査実施数	不具合数
JDE	6,167	1,504
JCG	3,585	264
JRC	196	19
JCI 連携 UI	112	13
システムシナリオテスト	22	7

表４　検査実施状況

(3) 処理性能評価結果

既存製品で同じバッチ処理を行うプログラムを作成し、処理性能を比較した。比較対象としては、Java バッチフレームワークである SpingBatch と Terasoluna を用い、次の 2 種類の処理の実行時間を計測比較した。

処理 A：ファイルから全件抽出したデータを全件データベース表に挿入
処理 B：データベース表から全件抽出したデータをファイルに出力

ここでは、SpringBatch との比較結果を、表５に処理件数と実行時間を、図 14 にそのグラフを示す。ここで、Jaime とは本ツールの製品名である。

件数(万件)	処理 A SpringBatch	処理 A Jaime	処理 B SpringBatch	処理 B Jaime
1	3587	740	3353	887
10	11263	1960	9963	3390
20	19853	3310	16697	6207
100	91230	15530	79910	29193

表５　処理件数と実行時間の比較

【評価結果】
・どちらのパターンにおいても本ツールが優位である。
・処理件数の増加において、SpringBatch は N**2 のオーダーに近い形で時間が増加しているが、本ツールはほぼ N のオーダーでしか増加しない。
・JavaVM のメモリ使用量も本ツールのほうが少なかった。

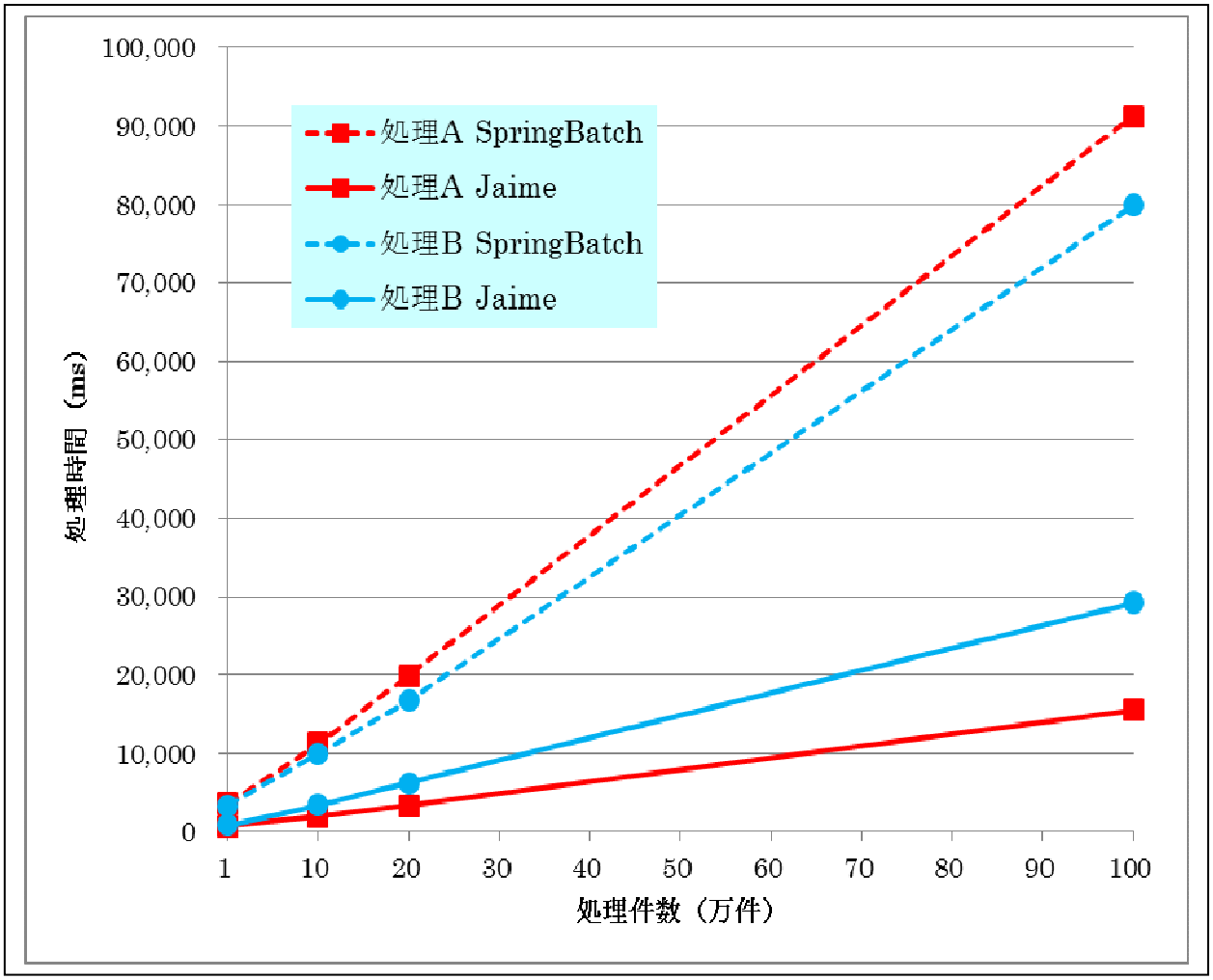


図 14 処理性能比較グラフ

４－６　総括

本研究開発として計画した作業は各サブテーマ毎にすべて達成できた。Eclipse のプラグインとして配布し、簡単なインストール手順にて本ツールをインストールし、実際に nbx ファイルを作成、ジェネレートしたコードを JRC とビルド、実行までの一連の流れを確認することができた。したがって、本ツールの RC(Release Candidate)版が完成したとの認識である。各サブテーマ毎に、その成果と得られた成果物を示す。

※本ツールは、４つのサブテーマで構成され開発されてきたが、これは Java 言語のプログラム開発ツールとして最適な開発方式を提案することでもある。そして、各サブテーマにおいて適用すべき技術を選択し、それを用いた実装を検証することにより、他製品にはない本ツールの優位性と存在価値を示すことができた。これらのことを特許とすべく、出願を行った（後述）。

ア　JDE の研究開発

nbx ファイルを新規作成、保存、設計情報を作成、編集するというエディタの基本機能とそのための UI、およびすべてのバッチコンポーネントのカスタマイズ UI が実装でき、MVC モデルとして V:ビューの部分を完成した。これらの UI 実装には、eclipse 既存機能との連携が必須であるが、結果的に eclipse の基本機能を損なうことなく達成できたので eclipse

との親和性は非常に高いものになった。このことは、今後本ツールが広く展開されるためには絶対に必要な要件なので、その点では満足している。特に、eclipse 連携のうち、Undo, Redo 操作については、単にキー入力やマウスのイベントのような１つの操作単位を対象とせず、JDE 上での設計情報に対する最小単位を扱うことができたので、ユーザにとってはたいへんに有利に働くはずである。実現には予定工数の 2 倍以上の時間を費やしたが、要求機能以上の成果であった。一方、M:モデルに展開された設計情報は、シリアライズされた後、XML 形式の nbx ファイルに保存されるので、本ツールを支援する今後開発されるツールにとっても扱いやすい形式となった。
今後は、さらに良い使い勝手を追及し、UI 機能やオプション機能を追加・改善していく予定である。

成果物：

バッチコンポーネント機能仕様書、スキーマウィンドウ機能仕様書、
コマンド API(アクション指示書)、JDE プログラム本体一式

イ　JCG の研究開発

Code Generator としては、M:モデルの構文木から得られるレファレンスコードのコードテンプレートを作成し、M:モデルの中間形（即ち、バッチコンポーネントのカスタマイズポイント）から可変部分を作り出すという生成方式により、今後の変更や追加に十分に対応できる、強い実装ができた。さらに、今後の性能評価で実行性能向上が要求されるような場面や、将来予定している既製のバッチフレームワーク製品への対応の場面など、バージョンアップも含めて、大きな問題は出ないと確信している。加えて、本来の生成言語は Java であるが、将来的に他の言語での生成要求においても、ターゲット言語のレファレンスコードを設計し、与えてやることにより容易に実装可能な構造である。実際、本ツールのバージョン 3.0 以降では、Oracle のための PL/SQL, Pro*C/C++、SQLServer のための C#でのコード生成が予定されている。
実行性能という観点から、今後はいろいろな入力仕様（設計情報）から生成される Java コードを、具体的にレビュー、評価が可能となり、より性能の高い生成系を目指すことができる。開発においては、まず M:モデルと Code Generator が並行分離開発され、その後に、M:モデルからデータを取得する API を実装することにより、M:モデルと Code Generator が疎の関係で連結され、JCG が完成した。

成果物：

ジェネレータ定義・機能仕様書、
JCG 中間形仕様書、JCG 抽象構文木仕様書、M:モデル構造定義書、
M:モデル API 、JCG プログラム本体一式

ウ　JRC の研究開発

ランタイムに必要なライブラリ機能の製造作業はすべて完了できた。もともとプロトタイプの状態から、特に実行時間性能に関しては評価をしながらの実装であったので、現時点においてもその心配は少ないと言える。実データベース表やいろいろな種類、大きさのファイルを使用した実行効率、メモリ使用量などの具体的な数値を得ることができるようになったので、今後はこれらの評価をベンチマーク方式で行っていくことができる。バッチ処理では、オーダー的に数十万、数百万件のレコードを対象にすることはごく一般的なので、これらの大きいオーダーにおけるさらなる評価も行っていきたい。このランタイムクラスは、本ツールのデフォルトのフレームワークとして提供（リリース）される。

成果物：

JRC 実行時環境仕様書（詳細仕様書)、JRC プログラム本体一式

エ　JCI(Java Code Inspector)の連携UIの研究開発

連携方式および連携機能の実装が完了した。JCI も本ツールと同じレベルにインストールされるeclipseプラグインなので、実際にインストールをして、JCGの生成するJavaプログラムに対し、コードインスペクションの解析実行を行いながら最適な連携方式（情報の取得方法）を検討した。特に重点を置いたのが、既存製品であるJCIに改修を加えることなく連携させるメカニズムであったので、JCIの結果ファイルであるCSVファイルを経由することとした。そして、得られた情報をJDEに通知する機能と、JDEで与えられたバッチコンポーネントを自動で開く機能を並行に実装していくことができた。本ツールの大きな特徴である、コードインスペクションサイクルの効果も評価できた。なお、本サブシステムとしてのソースコードはJDEの中に含まれる。

成果物：

JCI連携動作仕様書

オ　システム、製品化評価のシナリオ設計

バッチ処理要件調査（定義)、シナリオ実行のための方式設計、そして製品化評価のシナリオ設計および実装の作業（シナリオキットの作成）はすべて達成できた。このシナリオに基づき製品化評価を実施したところ、すべてのサブシステムに不具合修正や機能改善要求が発生したが、効率良く各サブシステムの改修作業を行うことができた。これも、MVCモデルによる本ツールの設計方針が寄与している。将来的に、このシナリオに基づくシナリオテストは、システム自体のマイナー改修やバージョンアップ等のメジャー改修時に実行すべきデグレードテストキットとして用いることができる。

成果物：

バッチ要件定義書、シナリオ方式設計書、シナリオ設計書、
シナリオキット一式

５　参考資料

５－１　研究発表・講演等一覧

外部発表分類：その他

・平成 22 年 12 月 9 日、コクヨホールにて行われた NICT 主催の「NICT 民間基盤技術研究促進制度ベンチャー支援制度　成果発表会」にて展示を行った。

　展示題名：

　　Eclipse プラグインによる Java バッチシステム開発自動化ツールの紹介

・平成 23 年 10 月 4 日～10 月 7 日、CEATEC JAPAN 2011 ICT Suite の「NICT の民間基盤技術研究促進制度による委託研究成果」にて出展をした。

　展示題名：

　　Eclipse プラグインによる Java バッチシステム開発自動化ツール