平成２１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（SpringBatch, 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

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エア技術面での国際競争力も高まる。

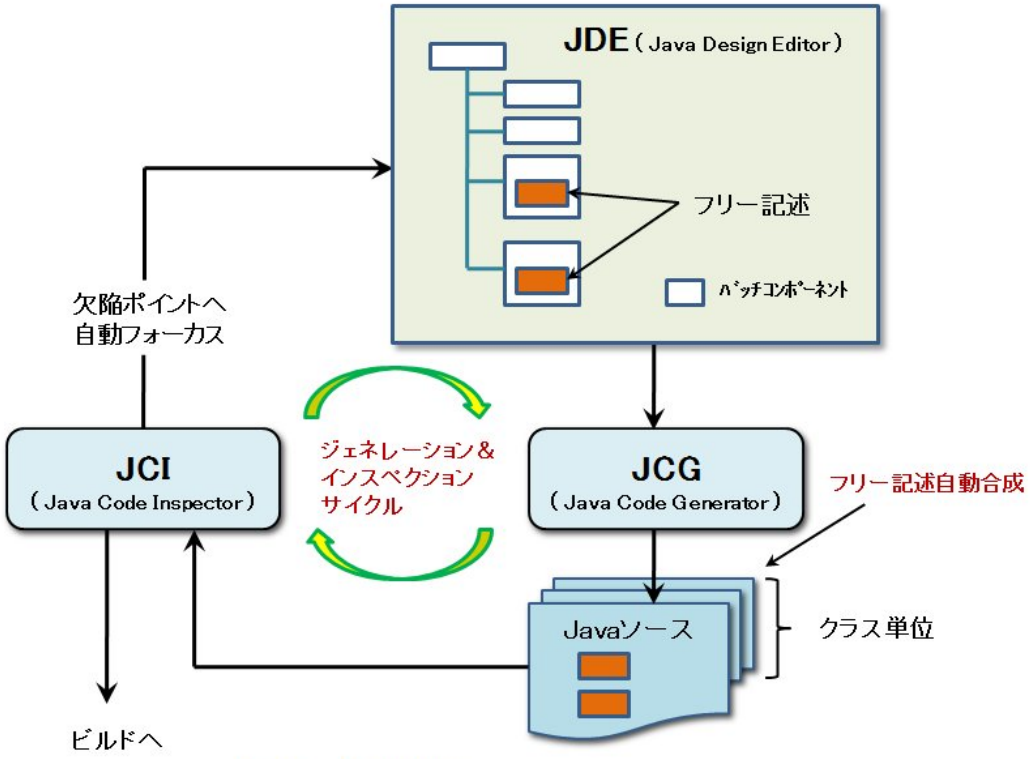


図１　本ツールの概要

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する。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ある。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、実装する。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る。主なカスタマイズされる設計情報を示す。
・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する。以下の点につき十分に配慮した生成をする。
・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 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する。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ある。Eclispe 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。

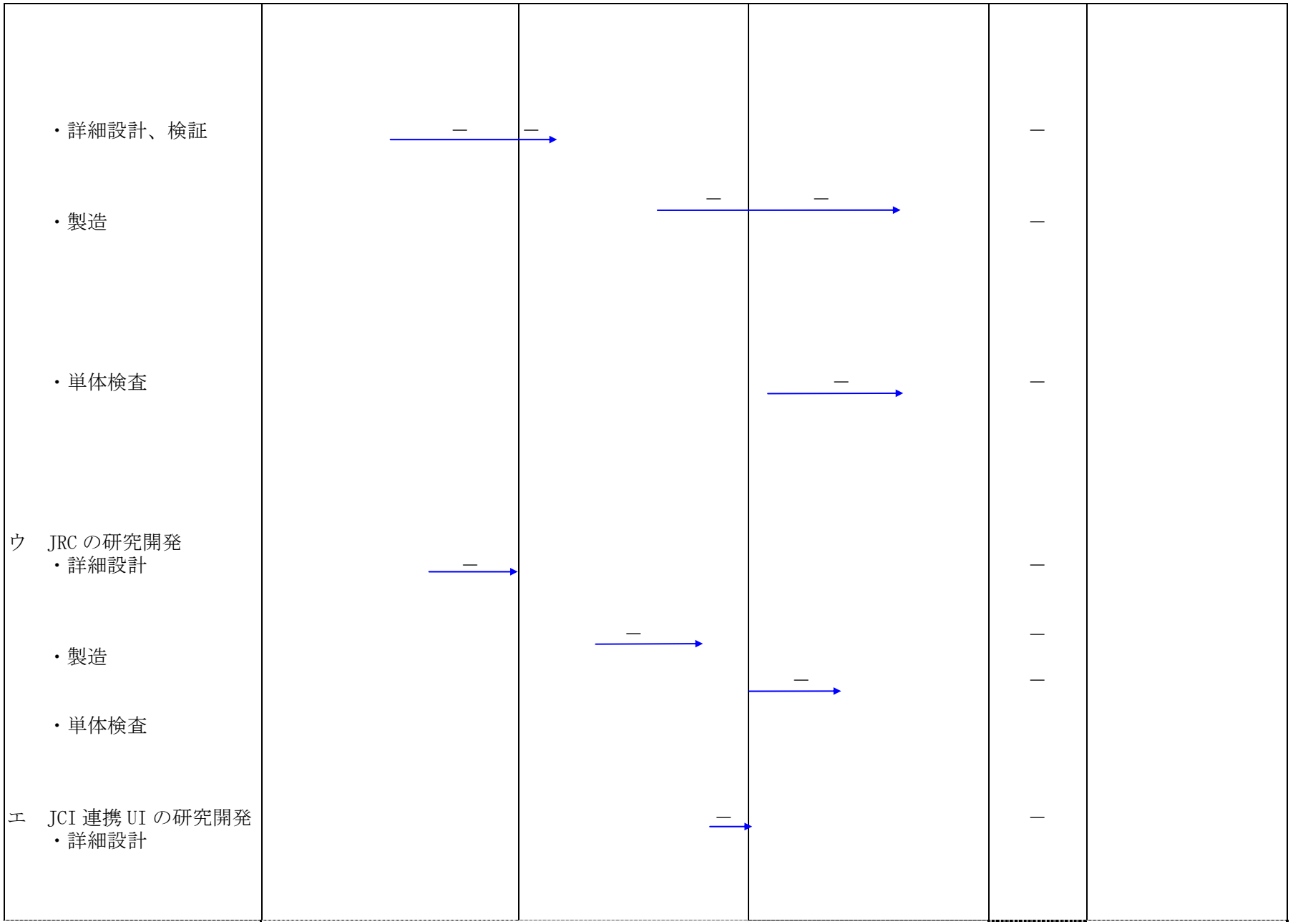
エ　JCI との連携 UI に関する研究開発【サブテーマ】
(1) 既存 Eclipse の優れた GUI を損なうことなく使用できること。
(2) 指摘されたコードの生成元に適切かつ高効率なフォーカス移動ができること。
(3) 設計時に仕様とする、静的に解析可能な全て検査（欠陥）項目を漏れなく検出すること。

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

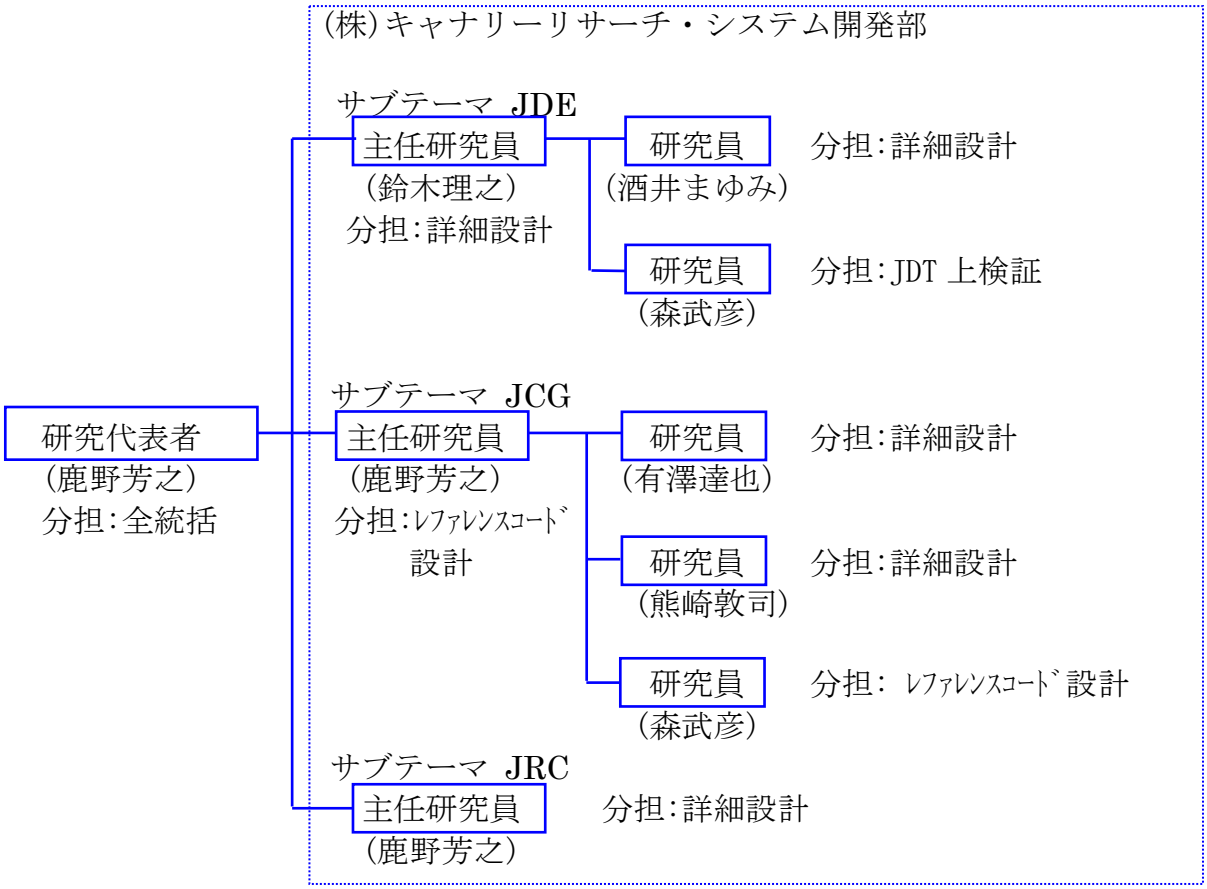
研究開発項目	２１年度	２２年度	２３年度	計	備考
Java バッチシステム開発自動化ツールの研究開発					
ア　JDE の研究開発					
・詳細設計	―	―		―	
・Eclipse JDT 上検証	―			―	
・製造		―	―	―	
・単体検査			―	―	
イ　JCG の研究開発	―				
・レファレンスコード設計		―		―	


・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	―	―	―	―	

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

平成 21 年度は次の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)

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った。そして、これらの UI を eclipse 上に実装するために、eclipse の既存機能（主に JDT とその UI）との連携、および JDE を実装するための 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。

４－１－１　詳細設計および UI ガイドラインの作成

(1)　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
　英語名、日本語名の決定
・アイコンのデザインおよび色
　すべて新規デザインとし、色に関しては別途実装時に再定義する
・リンク可否
　プログラム木上での親リンク、子リンク、兄弟リンクそれぞれの定義
・カスタマイズポイントおよびその UI
　デフォルト値
　入力要素
　入力属性
　編集可否および編集可能領域
　入力方法
　　TextBox, ComboBox, RadioButton, MenuSelect, FreeEditor, その他
　保存方法
　　ファイル保存、レジストリ保存、プロパティ保存

(2)　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詳細設計を行った。設計ポイントは次のとおりである。

・全体デザイン、構成、配置、色
後出の eclipse 上の検証時にも関連するが、本ツールの“顔”の部分に相当する、非常に大事なポイントである。このベース技術を確立して以降の設計、実装が実現されていくことを前提に設計を行った。すなわち、JDE 全体に関係する UI ガイドラインを定義、作成することである。

・プログラム木の表現方法、実装方法
・バッチコンポーネント連携

・JCG 実行環境の定義、実装方法
・JCI 実行環境の定義、JDE との連携、実装方法
・保存ファイルの定義、入出力機能の定義、実装方法
これらはエディタの基本機能およびプログラムを設計するツールを実装するための指針となるように設計を行った。

・スキーマウィンドウのデザイン、構成、表現方法、実装方法
本ツールの大きな特徴の１つになり、ツールとしての使用勝手を左右するスキーマウィンドウに関係するすべての設計を行った。

(3)　定義ファイルの書式および定義
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する。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(DB)表を定義する
　　(Oracle, SQLServer を当面の対象とする)
・ファイル定義ファイル
　　外部フラットファイルを定義する
・項目置換ファイル
　　表名、表項目名、データタイプ、データ属性の変更を定義する

(4)　Eclipse 上の検証
JDE の UI を eclipse 上に実装するために、eclipse の既存機能（主に JDT とその UI）と JDE を実装するための UI 部品について調査、検証を行った。特に UI 部品に関しては、既存製品の広範囲かつ十分なサーベイおよび調査を実施、決定し、検証をした。さらに、上記(2)の実装方法が eclipse ユーザにとって常に親和性を保つことを検証、確認した。

４－１－２　実施状況

(1)　バッチコンポーネントの詳細設計
詳細設計の成果として、次に示すすべてのコンポーネントについて、機能仕様を作成、定義し、”バッチコンポーネント機能仕様書”としてまとめた。

1) Java 言語の構文要素
　　クラス定義コンポーネント
　　インスタンス変数宣言コンポーネント
　　Getter 定義コンポーネント
　　Setter 定義コンポーネント
　　メソッド定義コンポーネント
　　制御構造コンポーネント
　　　　if コンポーネント
　　　　switch コンポーネント
　　　　for コンポーネント
　　　　while コンポーネント
　　　　do while コンポーネント
　　　　continue コンポーネント
　　　　break コンポーネント
　　　　return コンポーネント

synchronized コンポーネント
ブロック・コンポーネント
ラベル
例外処理コンポーネント
try catch コンポーネント
throw コンポーネント
assert コンポーネント
Import 定義コンポーネント
インタフェイス定義コンポーネント
変数宣言コンポーネント
メソッド宣言コンポーネント

2) データベース処理
DB 駆動表ループ・コンポーネント
コントロールブレイク DB 駆動表ループ・コンポーネント
DB 出力処理コンポーネント

3) ファイル入出力
FILE 駆動表ループ・コンポーネント
コントロールブレイク FILE 駆動表ループ・コンポーネント
FILE 出力処理コンポーネント
マルチレイアウト FILE 駆動表ループ・コンポーネント
マルチレイアウト FILE 出力処理コンポーネント
マッチング処理ループ・コンポーネント

4) その他
フリー記述
初期処理
終了処理

(2) JDE 基本機能の詳細設計

まず、グランドデザインを行い、その結果を“JDE グランドデザイン定義書”としてまとめ、JDE 設計者全員にプレゼンおよび全員によるレビューを行い、3 回に渡り改版を行い完成した。図２に、JDE のメイン画面イメージを示す。その後、この定義書にもとづき、詳細設計の成果として、次の機能仕様書を作成した。

“バッチコンポーネントアイコン仕様書”
“バッチコンポーネント機能仕様書”
“メニュー機能仕様書”
“スキーマウィンドウ機能仕様書”

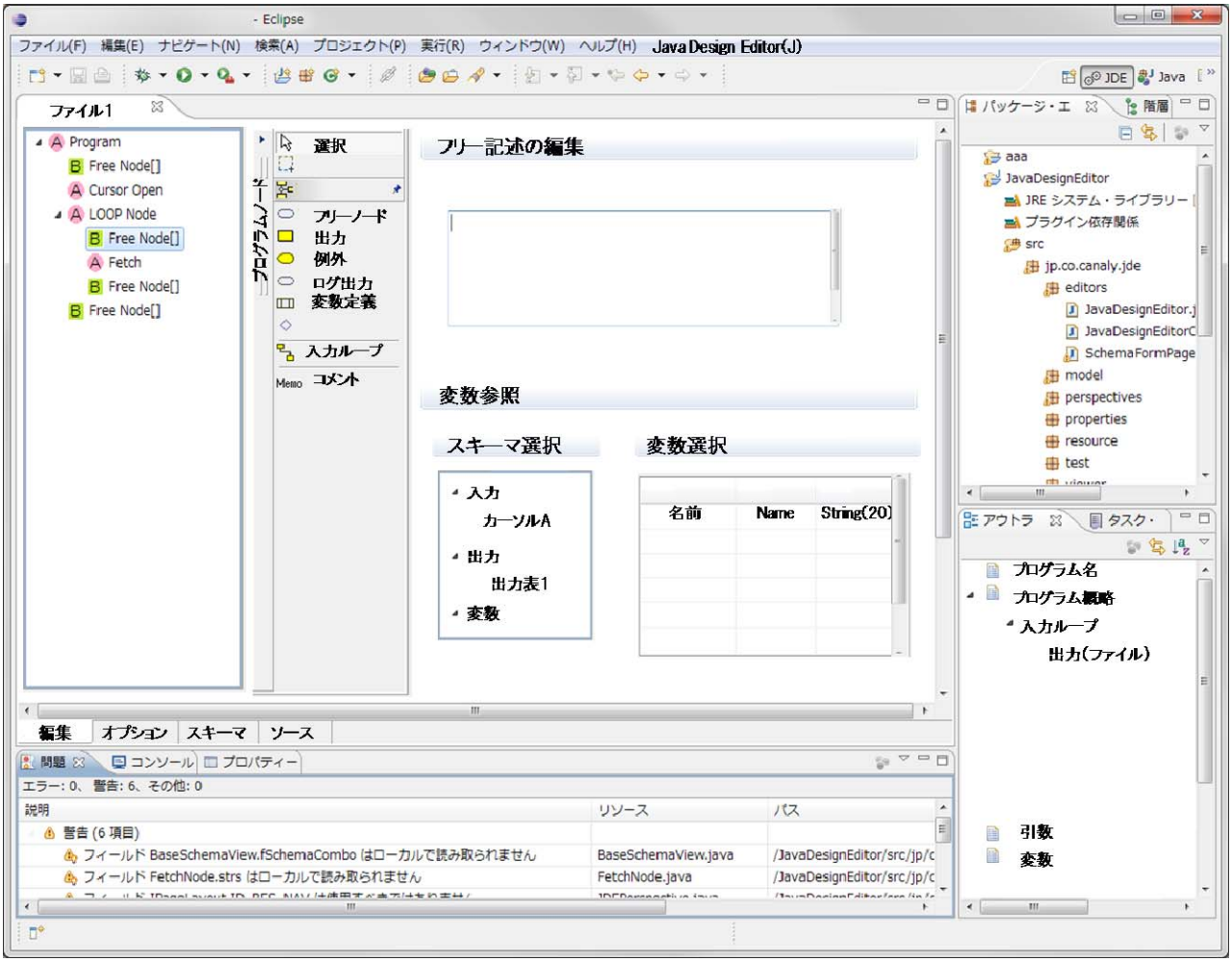


図２　JDE のメイン画面イメージ

(3)　　定義ファイルの書式および定義

本ツールの入力・出力スキーマとして適用するスキーマ定義ファイルについて、書式および定義内容を定義し、“スキーマ定義ファイル仕様書”としてまとめた。

(4)　　Eclipse 上の検証

eclipse 上のプラグインにふさわしく、eclipse ユーザにとって自然な操作性となるよう、eclipse の主要な既存機能（JDT とその UI）を JDE 実装のために使用するためのガイドと JDE 実装方式（このように実装すべきであるという意味で）をまとめ、“JDE-UI ガイドライン”を作成した。

４－１－３　実施計画に対する達成状況

本年度として計画した作業はすべて達成できた。特に、UI ガイドラインについては予想を上回る工数がかかってしまったが、JDE 実装に向けての準備としては十分な内容と考えられる。22 年度 4 月期に、このガイドラインにしたがって、作成した仕様書を見直し eclipse プラグインの詳細設計書としてふさわしいものに改版する予定である。

なお、当初予定していた仕様書名は次のように対応する。
・“ノード定義書”は“バッチコンポーネントアイコン仕様書”として作成。
・“プログラム木機能仕様書”は“メニュー機能仕様書”内に包含。
・“グランドデザイン定義書”は“JDE-UI ガイドライン”として作成。

４－２　JCG の研究開発

JCG が生成するレファレンスコードを設計した。ある程度試行錯誤的にコード設計、記述を行い、その都度ビルド、実行を行い、多角的に検証を実施した。その後、このレファレンスコードを生成するために必要な中間形の設計、およびその中間形からコード生成する JCG 本体の実装のための詳細設計を行った。

４－２－１　レファレンスコードの設計、検証と JCG の詳細設計

(1)　　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。
・レファレンスコード設計の目的は、
　　ソースコード的に美しく、高可読性であること、
　　オブジェクト指向を保つこと、
　　メモリ空間においても手書き版に劣らないこと。
・検証する項目は次のとおりである。
　　メモリ空間、処理性能において問題のないこと
　　既存のフレームワークとの親和性が保たれること
　　フレームワークなしでの実行が可能であること
　　RDBMS に非依存であること
　　Linux, Windows 等のプラットフォームに非依存であること
　　32bit, 64bit アーキテクチュアに非依存であること

(2)　　JCG の詳細設計
レファレンスコードが決まるとそれを生成するために必要な情報を決めることができる。本ツールでは、これらの情報を JDE から与えられる設計情報とプログラム木から取得し、内部的に抽象構文木と中間形（合わせて中間形という）という表現形式で保持する。ジェネレータはこれら内部表現からコード生成を行う。次の項目を詳細設計の内容とした。
・レファレンスコードを生成するための中間形の設計
　　生成に必要かつ十分な情報を保持し、かつコード生成がやりやすい形式を決めた。
・JDE から受け渡される全データの設計、決定
　　ジェネレータに引き渡す JDE からの情報の決定とそれらを保持する内部データ構造を設計した。
・オブジェクト指向に基づくクラス抽出の設計
　　本ツールは使用者にオブジェクト指向の考え方やスキルを前提としないので、ジェネレータがクラス構造を決める必要がある。そのためのクラス抽出の設計を行った。非オブジェクト指向構造（手続き型設計方法）もその対象とした。
・ジェネレータ基本機能の設計

中間形からコード生成を行う（換言すれば、中間形をコードに変換する）方式の設計を行った。加えて、実際の コード出力方式の設計、およびバッチ一括生成方式の設計も行った。

４－２－２　実施状況

(1)　　レファレンスコードの設計と検証

実際に設計、作成したレファレンスコードパターンとその簡単な説明、および実際の業務例を表１にまとめる。

パターン名	入力	出力	処理内容	業務例
DB-DB	DB	DB	1つ以上のデータベース表を指定条件で検索し、フェッチしたレコード毎に加工処理を行い、1つ以上のデータベース表を更新する	マスタメンテ 日次、月次集計
DB-FILE	DB	FILE	1つ以上のデータベース表を指定条件で検索し、フェッチしたレコード毎に加工処理を行い、1つ以上の集配信ファイルを作成、更新	集計処理 配信ファイル作成
FILE-DB	FILE	DB	配信ファイル等のトランデータを読込み、レコード毎に加工処理を行い、1つ以上のデータベース表を更新する	集計処理
FILE-FILE	FILE	FILE	配信ファイル等のトランデータを読込み、レコード毎に加工処理を行い、1つ以上の集配信ファイルを作成、更新	集計処理 配信ファイル作成
DBコントロール ブレイク	DB	-	あるキーでソートされたデータベース表を検索し、同一キー値のレコードに対して集計や累積処理を行う。ブレイクは多段でも可	ソート処理 多段階集計処理
FILEコントロール ブレイク	FILE	-	あるキーでソートされたファイルを順次読込み、同一キー値のレコードに対して集計や累積処理を行う。ブレイクは多段でも可	ソート処理 多段階集計処理
入力マルチレイアウト	FILE	-	可変長のファイルを読込み、レコード種類に応じた処理を行う	可変長レコード ファイル読込
出力マルチレイアウト	-	FILE	レコード種類に応じた処理を行い、可変長レコードのファイルを作成する	可変長レコード ファイル作成
ノンカーソル	DB	DB	カーソルを使用しないデータベース処理	一括条件更新 一括条件削除
1:1マッチング	FILE	-	マスタファイルとトランザクションファイルを順に見ていって、トランザクションレコードのキーに対応するマスタレコードを見つけて、両者を一緒にして出力する	1対1マッチング
1:Nマッチング	FILE	-	1対1と同様だが、マスタレコードと対応するトランザクションレコードが複数あるケース	1対Nマッチング

DB:データベース表(結合表、パーティション表など)
FILE:外部ファイル(形式:バイナリ、テキスト、CSV、XML、COBOL型)
-:　任意

表１　レファレンスコードパターンと処理内容

(2)　　JCG の詳細設計

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のクラス階層に、コード生成処理を分離して持たせた構造が適切だと結論付けられた。

JDE の主機能を精査した結果、データは大きく、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る。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・変更に強い構造となっている。

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。無論、バッチ一括生成についても対応可能である。

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。また、このアーキテクチュアにより、生成コードも自然なクラス構成を取るようになり、結果的に手書きによる綺麗なソースコードに勝るとも劣らないコードを生成することができるジェネレータの設計ができた。

４－２－３　実施計画に対する達成状況

本年度として計画した作業はすべて達成できた。レファレンスコードの実際のビルドによる検証では、本ツールも対象とする既製のバッチフレームワークである SpringBatch を利用した（最小モデルではあるが）DB-DB パターンで、性能比較を行った結果、本ツールの生成するコードの優位性も証明することができた。1 例を表２で示し、その根拠を記す。

10万件データ転記比較　（単位:ms）

		リファレンスコード					SpringBatch	
COMMIT間隔	No.	フェッチサイズ					No.	デフォルト設定
		デフォルト	1	10	100	1000		
1	1	94141	95687	81984	89281	80000	1	236312
	2	79188	91718	85750	79594	81656	2	237219
	3	86094	101922	74937	85969	78969	3	228203
	AVG.	**86474**	**72332**	**60670**	**63736**	**60406**	AVG.	**233911**
10	1	13718	23750	15359	14172	13547	1	30578
	2	12875	22421	14781	13812	12906	2	29703
	3	13875	22813	14859	13094	13907	3	29719
	AVG.	**13489**	**22995**	**15000**	**13693**	**13453**	AVG.	**30000**
100	1	6297	15094	7281	6500	6578	1	8813
	2	6375	15062	7250	6375	7422	2	8563
	3	6437	14906	7125	6500	6313	3	8594
	AVG.	**6370**	**15021**	**7219**	**6458**	**6771**	AVG.	**8657**
1000	1	5938	14015	6797	6078	6078	1	6532
	2	5781	14437	6719	6081	5906	2	6640
	3	5860	14516	6688	6125	5875	3	6531
	AVG.	**5860**	**14323**	**6735**	**6078**	**5953**	AVG.	**6568**

表２　最小 DB-DB パターンによる JCG 生成コードと SpringBatch 利用コードとの性能比較

・フェッチサイズを大きく（バルクフェッチを指定）すれば、コミット間隔に関係なく、本ツールの生成コードのほうが、処理速度が速い。
→レコードを 1 行ずつ加工して、1 行ずつ DB 表に書き込む本ツールのバッチ処理形態では圧倒的に有利である。
・ユーザにバルクフェッチの知識ない場合のデフォルト設定においても、本ツールの生成コードのほうが、処理速度が速い。
→ユーザは、バルクという概念にとらわれずに設計に注力可能である。

４－３　JRC の研究開発

ランタイム環境を提供する JRC クラス群の機能設計、詳細設計を行った。

４－３－１　JRC 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。

(1) データベース処理
フェッチクラス、DDL クラス、バルク処理クラス、Connection (Pool)クラス

(2) フラットファイル処理
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

(3) 例外処理およびユーティリティ処理
例外処理およびユーティリティ処理クラス

４－３－２　実施状況

個々のクラスの詳細設計が完了し、プロトタイプモデルも実装できた。結果的に、JRC クラスライブラリとして設計され、本ツールのデフォルトのフレームワークとして提供出来るはずである。図３に DB 表使用の生成コードのクラス構成を、図４にファイル使用のクラス構成を JRC との関連図で示す。

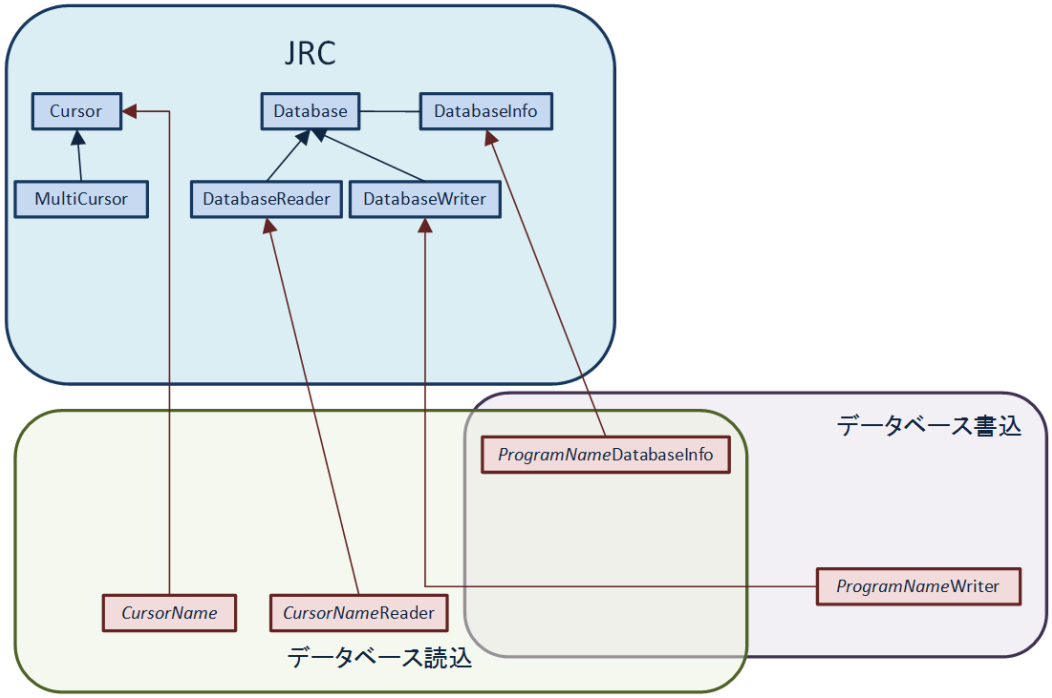


図３　生成コードのクラス図（DB 表使用モデル）

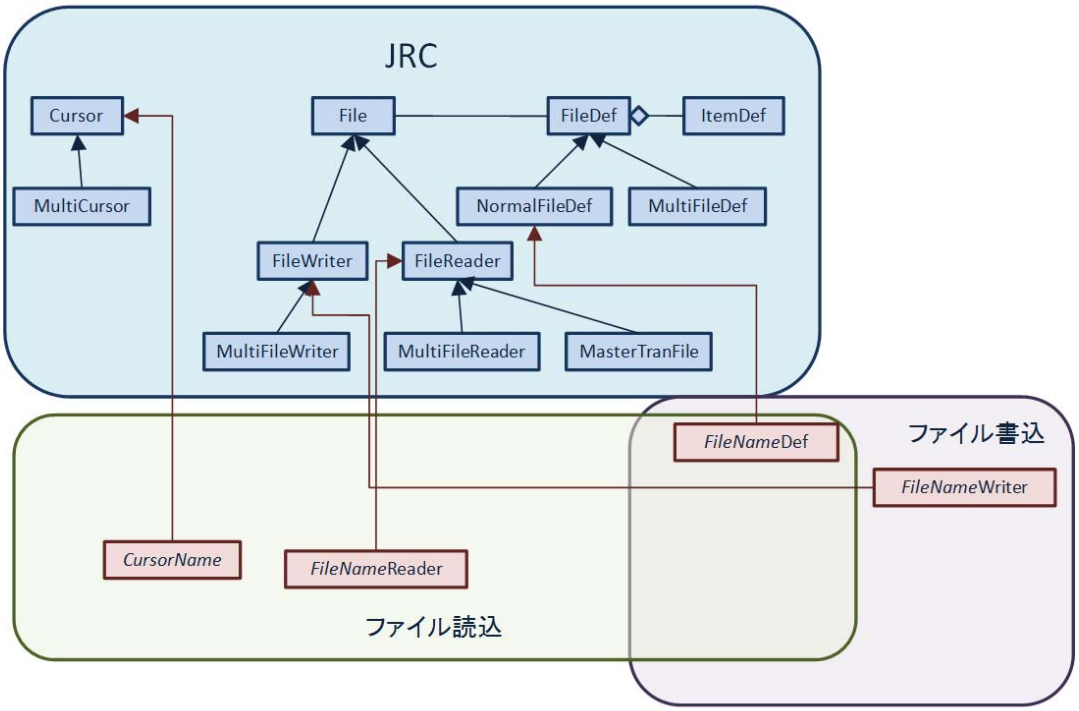


図４　生成コードのクラス図（ファイル使用モデル）

４－３－３　実施計画に対する達成状況

本年度として計画した作業はすべて達成できた。レファレンスコードを動作、実行させる

ために必要なクラスについては、プロトタイプで仮実装ではあるが作成し、生成コードとのビルドおよび実行までの確認ができた。今後の JCG の生成コードが実業務に耐えうる品質になるにつれ、JRC も並行して改版が続けられなければならない。そして、他の既製バッチフレームワークとの親和性や連携も考慮する必要がある。

４－４　総括

ア　JDE の研究開発【サブテーマ】
21 年度に予定していた､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作業は完了し、成果物として以下の仕様書を作成した。

バッチコンポーネント機能仕様書、JDE グランドデザイン定義書、
スキーマ定義ファイル仕様書、JDE-UI ガイドライン、
バッチコンポーネントアイコン仕様書、メニュー機能仕様書

特にカスタマイズポイントに使用する UI についてはとてもよく設計できている｡しかし、詳細設計の上での議論なので、次に示すような課題は次年度の実装フェーズで優先的に解決していく予定である。
・eclipse への親和性および eclipse ユーザにとって自然な UI が設計レベルであり、予想の域を脱しない部分が散在する。
・ユーザにオブジェクト指向を求めないという前提で Java エディタとして成り立つかどうかの検証。
・JCG, JCI とのインターフェイスにおいても実装してみて初めて判明する情報の不足の補足の方法。

イ　JCG の研究開発【サブテーマ】
同様に、21 年度に予定していた、レファレンスコードの設計・試作、JCG の詳細設計作業は完了し、成果物として以下のものを作成した。

レファレンスコード集、中間形定義書、
抽象構文木定義書、ジェネレータ定義-機能仕様書、
コード検証報告書

特にレファレンスコードを設計した上で、簡単ではあるが仮の環境でビルド、実行ができたパターンがあり、より具体性のある詳細設計を行うことができた。また、ジェネレータ・アーキテクチュアについても柔軟性のある、理論的に綺麗な設計をすることができたことは大きな収穫であった。

ウ　JRC の研究開発【サブテーマ】
同様に、21 年度に予定していた、JRC クラス群の機能設計、詳細設計作業は完了し、成果物として“ランタイムクラス詳細設計書”を作成した。JCG のレファレンスコードの検証のために、JRC も仮実装が行われ、基本動作の検証を行うことができ、より堅実なフレームワークを設計することができた。しかし、次に示すように課題がいくつかあるので、次年度の実装フェーズですべて解決する予定である。
・既製のフレームワークとの連携方法については、公開情報が足りなかったり、特殊用途の目的をもつものがあるため、どこまで本ツールの対応範囲とするかの判断が困難である。
・特に、SpringBatch との性能比較を行ったが、最小モデル 1 つに過ぎないので、まだまだ検証モデルを設計、構築しなければ優位性を謳うことはできない。

・今年度はデータベースとしてフリーのものしか利用していないため、今後ターゲットとする商用 RDBMS である Oracle と SQLServer 上での実行検証を、次年度のどの時期に行うかが未定であるため、早めの対応が要求される。

5　参考資料

5－1　研究発表・講演等一覧

該当なし

5－2　産業財産権

該当なし