Javaソースコードの CDI(Code Inspection) の開発
～アーキテクチャの構築～

M2007MM005 後藤洋
指導教員：野呂昌満

1　はじめに

ソフトウェアの品質を保証する方法として様々なソフトウェア検証技術が存在する．ソフトウェアの検証技術はソフトウェアを実行し実行結果から誤りを検出する動的検証技術とソフトウェアを実行せず成果物の内容を検査する静的検証技術に分類される．静的検証技術のひとつにインスペクションがある．インスペクションでは，仕様書やプログラムなどのソフトウェアの成果物に形式化された見直しを行うことで，成果物に含まれる欠陥を発見する．インスペクションで発見される欠陥とは不正な処理，手順，データの定義などの論理的な誤りやコーディング規約のような開発上で定められるルールに対する違反などである．特にプログラムに対するインスペクションをコードインスペクションと呼ぶ．コードインスペクションツール (以下，CDI ツール) はソースコードにたいしてプログラムの意味解析，制御フロー解析，データフロー解析などの静的解析を行うことで欠陥を発見するソフトウェアである．CDI ツールが行う検査はリソースリークといったプログラムの不正な処理の問題を発見するものやコーディングスタイルを確認するものなど様々な種類が存在する．一方，それぞれの CDI ツールで扱うデータは抽象構文木，フローグラフと共通している．CDI ツールは製品系列で共通部分と変動部分が存在する．

製品系列の共通点と相異点に着目した開発方法論としてプロダクトラインソフトウェアエンジニアリング (以下，PLSE)[1] がある．PLSE では，製品群の共通部分と変動部分を踏まえて製品系列の統一アーキテクチャを構築し，アーキテクチャに基づいて再利用可能部品を構築し体系化する．個々の製品であるソフトウェアは，統一アーキテクチャにもとづいて再利用可能部品を選択し組み合わせることで開発する．CDI ツールが扱うデータや検査処理の特徴から CDI ツールは PLSE に適したドメインであると言える．PLSE に基づいて様々な種類の CDI ツール開発を行うためには CDI ツールの共通部分と変動部分が明確な構造が求められる．

本研究の目的は，CDI ツールのソフトウェアアーキテクチャを構築し製品系列での共通部分と変動部分を明確化することである．CDI ツールの製品系列の要求を分析し抽出した CDI ツールの共通部分と変動部分をもとに，ソフトウェアアーキテクチャを構築する．ソフトウェアアーキテクチャを構築することによって，CDI ツールの共通部分となる要素，変動部分となる要素が明確になる．CDI ツールの共通部分と変動部分が明確になったことで，ユーザ要求の変更に対するソフトウェアの変更箇所の特定が容易となり，再利用性の高い CDI ツールの開発が可能となる．

本研究は CDI ツールのソフトウェアアーキテクチャを構築し，CDI ツールの製品系列における共通部分と変動部分の考察を行う．CDI ツールの製品系列の分析を行い，製品系列の共通点と変動点を分析し，CDI ツールに求められる構造を明らかにする．ソフトウェアアーキテクチャの構築では，内部データの柔軟な変更を可能にするためにオブジェクト指向によるデータ構造定義を行う．さらに変更に対する柔軟性の高い構造を実現するために定義したデータ構造にデザインパターンを適用する．

本 OJL は以下の手順で進めた．

1. CDI ツールのドメイン分析
2. オブジェクト指向によるデータ構造定義
3. デザインパターン適用によるアーキテクチャ構築
4. ソフトウェアアーキテクチャの妥当性の考察

2　メタ技術とベース技術

CDI ツールの開発で用いたベース技術とメタ技術について述べる．本 OJL で適用したベース技術をソフトウェアアーキテクチャに関する技術とドメイン特有の技術に分類し以下に列挙する．

- ソフトウェアアーキテクチャに関する技術
  - オブジェクト指向アーキテクチャ
  - デザインパターン
- ドメイン特有の技術
  - 静的検証技術
  - 言語処理技術
  - プログラミング言語

本 OJL でのメタ技術はプロダクトラインソフトウェアエンジニアリングである．

2.1　プロダクトラインソフトウェアエンジニアリング

系統的に再利用を行う開発方法論としてプロダクトラインソフトウェアエンジニアリング (PLSE) がある．PLSE は対象ドメインの分析結果をもとに核資産を定義し改良を繰り返しながら，核資産から部品を組み合わせて様々な製品開発を行うプロセスを規定している．核資産とは，要求分析の結果，再利用可能部品，部品化の手法およびアーキテクチャなどの開発に必要な項目全ての総称である．

PLSE は以下の 3 つの活動で構成される．

- 核資産開発
- 製品開発
- 管理

核資産開発はドメインエンジニアリングとも言い，対象ドメインの分析，ソフトウェアアーキテクチャの構築，再利用可能部品の構築などを行う．製品開発はアプリケーションエンジニアリングとも言い，製品の仕様をもとに核資産から必要となる部品を抽出し，抽出した部品を統合し製品となるソフトウェアを開発する．管理ではPLSEに基づいた開発を行うための組織やプロジェクトなどの体制の管理を行う．

2.2 オブジェクト指向アーキテクチャ

オブジェクト指向アーキテクチャ[2] はデータと手続きを一体とした抽象データで構成され，抽象データ同士のメッセージ通信によって互いに関連している．オブジェクト指向アーキテクチャは抽象データの内部の変更に対する柔軟性が高い特長を持つ．

2.3 デザインパターン

デザインパターンとは設計において鍵となる側面に名前を付け，抽象化，識別したものである．よく知られたデザインパターンにはGoFの23のパターン [3] がある．GoFの23のパターンでは提案しているデザインパターンを生成に関するパターン，構造に関するパターン，振舞いに関するパターンの3つに分類している．本研究では，GoFの23のパターンからCompositeパターン，Interpreterパターン，Visitorパターンを適用した．

2.4 CDIツール

CDIツールとはソースコードに静的解析を行い，プログラムの実行前にそのプログラムに含まれる欠陥を検出するソフトウェアである．CDIツールが発見する欠陥とは不正な処理，手順，データの定義などの論理的な誤りやコーディング規約のような開発上で定められるルールに対する違反である．欠陥の具体例として，リソースリークがあげられる．CDIツールの概要を図1に示す．

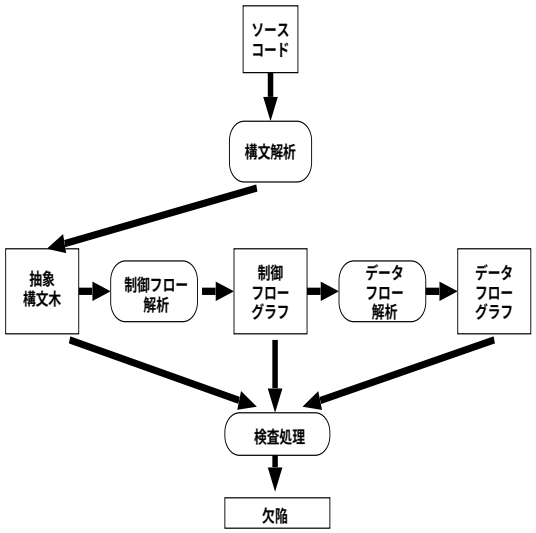

図1 CDIツールの概要

CDIツールは入力されたソースコードを字句，構文解析し，抽象構文木を構築する．さらに構築した抽象構文木をもとに制御フロー解析，データフロー解析を行い，制御フローグラフ，データフローグラフを構築する．CDIツールは抽象構文木を走査しプログラムの意味を解析した結果やフローグラフを走査し処理を行った結果を組み合わせる事で欠陥を発見する．

3 PLSEに基づいたCDIツール開発プロセス

CDIツールは扱うデータ構造の共通部分や製品ごとに様々な種類の検査を行う特徴を持つことから，PLSEに適したドメインである．我々はCDIツール開発において，PLSEに基づいた開発を行った．CDIツール開発における核資産を以下に示す．

- 再利用可能な要求
- ソフトウェアアーキテクチャ
- アプリケーションフレームワーク
- 検査機能を実現した検査モジュール

我々はPLSEのプロセスに従って，これらの核資産の改良をくり返しながら，核資産から必要となる部品を選択統合し製品開発を行った．我々が実施したPLSEに基づいたCDIツール開発のプロセスを図2に示す．

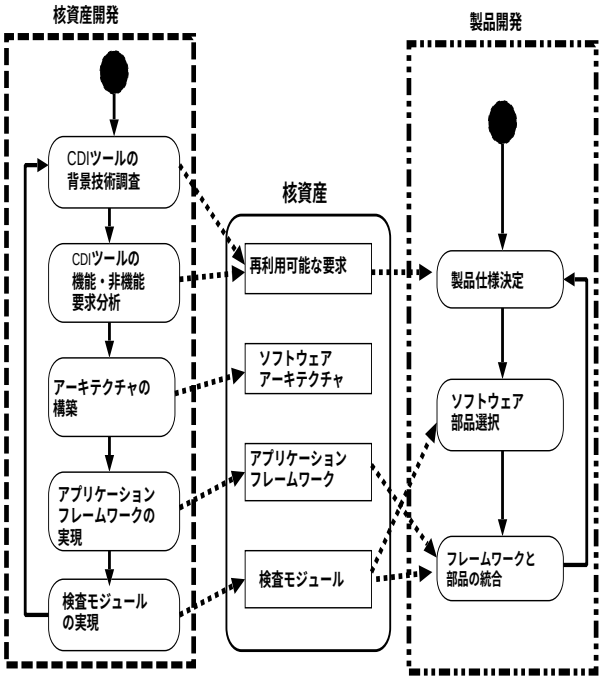

図2 開発プロセス

4 CDIツールのドメインの特徴

PLSEに基づいたCDIツールの開発を行うために，CDIツールのドメインを分析しその特徴を明らかにする．ドメイン分析では，CDIツールのドメインで共通する要求と変動する要求を分類する．さらに，CDIツールのソフトウェアアーキテクチャに求められる構造を明らかにするためにCDIツールに求められる機能要求・非機能要求の分析，CDIツールの背景技術の調査を行う．

CDIツールの機能はプログラムの誤りを発見する各種検

査である．CDI ツールが行う検査の種類は要求ごとに様々に存在し，検査の追加，変更の頻度が高いと考えられる．以上のことからから CDI ツールのアーキテクチャには処理の追加，変更に柔軟な構造が求められる．CDI ツールの非機能要求は処理に対する柔軟性である．

5　検査の追加変更に柔軟なアーキテクチャ

CDI ツールのドメイン分析結果をもとに，検査処理の追加，変更に柔軟なアーキテクチャの構築を行う．アーキテクチャの構築ではオブジェクト指向アーキテクチャスタイルに基づいて CDI ツールで扱う抽象構文木をオブジェクトとして定義する．オブジェクト指向に基づいて定義した抽象構文木にデザインパターンを適用し構造を整理する．

5.1　データ構造の定義

CDI ツールでは欠陥を発見するためにプログラムの意味解析，フロー解析を行う．意味解析やフロー解析を行うには，ソースコードを抽象化した抽象構文木が必要である．抽象構文木のデータの構造は CDI ツールが対象とするプログラミング言語によって決まることから，対象プログラミング言語の文法などの仕様をもとに定義を行う．本 OJL で開発した CDI ツールの対象とするプログラミング言語は Java である．抽象構文木は Java の言語仕様 [5][6]，Sun から提供されている抽象構文木のライブラリ，EclipseJDT[7] を参考にした．
我々は構文要素の内部データ変更に対する柔軟性を考慮し抽象構文木の構文要素をオブジェクト指向に基づいて抽象データ型として定義した．定義した抽象構文木のクラス図を図 3 に示す．

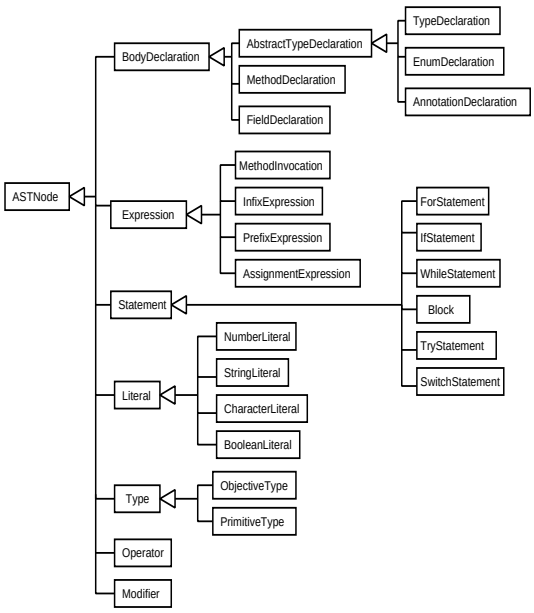

図 3　抽象構文木のクラス図

5.2　デザインパターンの適用

データ構造走査処理の再利用性や検査の追加，変更に対する柔軟性を確保するためにデザインパターンを適用する．

Interpreter パターン
Interpreter パターンは，言語の文法表現をオブジェクト構造として構成し，その文法表言の解釈するための処理を定義する．

Composite パターン
木構造を表現する．

Visitor パターン
Visitor パターンオブジェクトの構造に散在する処理を分離，局所化する．処理を局所化することでデータ構造を変更する事なく処理の追加，変更が可能になる．

これらのパターンから適切な組合せを検討する．以下の組合せを検討する．

- Interpreter パターン，Composite パターン
- Interpreter パターン，Composite パターン，Visitor パターン

Interpreter パターンと Composite パターンを組合せて適用した場合の構造を図 4 に示す．

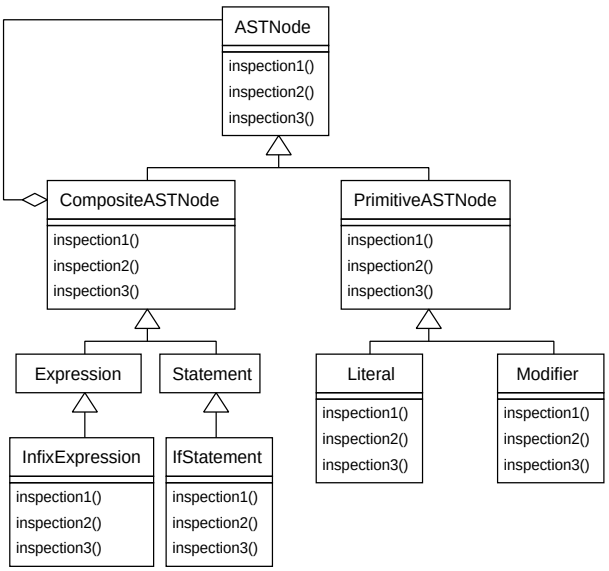

図 4　Interpreter パターン，Composite パターンの組合せ

Interpreter，Composite パターンの組合せでは，検査の追加変更を行うには構文要素クラスの全てに対して，検査の処理を行うメソッドを定義する必要がある．

Interpreter パターン，Composite パターン，Visitor パターンを組合せて適用した場合の構造を図 5 に示す．
Interpreter パターン，Composite パターン，Visitor パターンの組合せでは，Visitor のサブクラスに検査処理を定義する．Visitor クラスのサブクラスを追加変更することで検査処理を追加変更が可能である．

検査処理の追加変更に対する柔軟性を確保するために，

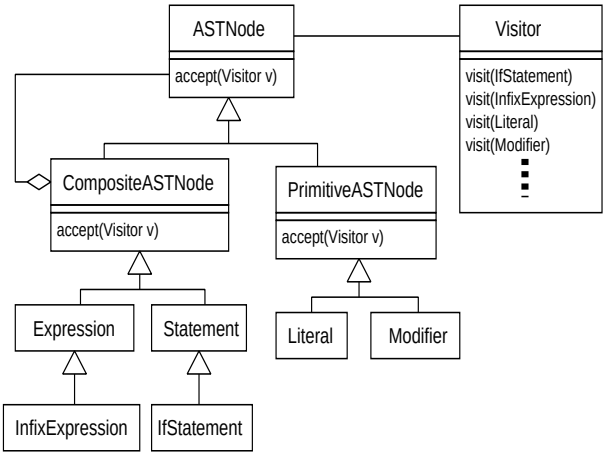

図 5 Interpreter，Composite，Visitor パターンの組合せ

Interpreter パターン，Composite パターン，Visitor パターンを組み合わせて適用する．

6 考察

構築したアーキテクチャの妥当性を示すために以下の考察を行った．

- 共通部分と変動部分の考察
- 部品と要求の対応関係の考察

6.1 共通部分と変動部分の考察

構築したソフトウェアアーキテクチャが共通部分と変動部分の部品化が可能か考察する．CDI ツールの変動部分は検査処理である．CDI ツールの各検査処理は Visitor クラスのサブクラスとして定義する．検査処理の追加，変更は定義した Visitor クラスのサブクラスを追加，変更することで可能である．このことから Visitor のサブクラスを変動部分の再利用部品として構築可能である．

CDI ツールの共通部分は抽象構文木とその走査の機能である．検査処理の追加，変更の際にデータ構造に対して変更を加える必要がないことから，抽象構文木は共通のデータ構造として再利用可能である．以上から構築したアーキテクチャによって共通部分と変動部分の部品化が可能である．アーキテクチャの共通部分と変動部分は図 6 のようになる．

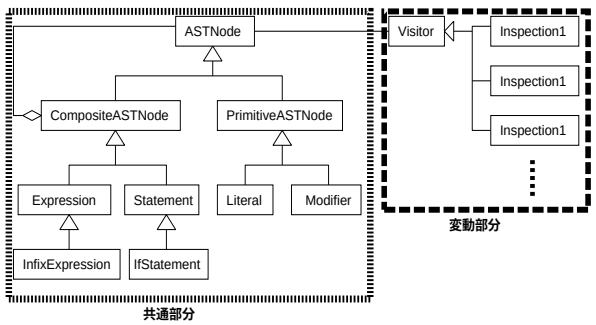

図 6 共通部分と変動部分

6.2 部品と要求との対応関係の考察

PLSE では，製品の要求から再利用部品の選択を行い，アーキテクチャに基づいて再利用部品を統合し製品を開発を行う．製品開発を行う上で適切な再利用部品を選択可能にするためには要求とアーキテクチャと再利用部品の間で対応づけられている必要がある．変動再利用部品である検査モジュールは仕様書と実現したソースコードの組合わせで構成される．検査モジュールの仕様書は PLSE の核資産である，再利用可能な要求を文書化したものである．本 OJL では製品開発において，製品の要求と，再利用可能な要求である検査モジュールの仕様書を照合することで，製品に必要となる再利用部品を選択することができた．選択した検査モジュールの統合はアーキテクチャによって定義されていることから，要求とアーキテクチャと再利用部品の間で対応関係が十分に明らかであったことを確認した．

7 まとめ

本研究では，オブジェクト指向に基づいて定義されたデータ構造に対してデザインパターンを用いてデータ構造の走査を実現した．Interpreter パターン，Composite パターン，Visitor パターンを適用した事で処理の追加・変更に柔軟に対応可能なアーキテクチャを構築出来た．さらに構築したアーキテクチャの共通部分はデータ構造であり，変動部分は検査処理であることを明らかにし，検査処理に対する柔軟性を確認した．さらに構築したソフトウェアアーキテクチャに基づいて再利用部品を構築した．それらの再利用部品を組合せ，PLSE に基づいて製品開発を行い，構築したアーキテクチャが有用であることを確認した．

今後の課題は他言語への応用と，コードインスペクション以外の静的解析ツールへの応用を考察することである．

参考文献

[1] L. M. Northrop,"SEI's Software Product Line Tenets",*IEEE Software,Vol.19 No.4*,pp.32-40,2002

[2] M.Shaw, and D.Garlan, *Software Architecture - Perspective on an Emerging Discipline*,Prentice Hall,1996

[3] E. Gamma,J. Vlissides,R. Helm, and R. Johnson, *Design Pattern Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] 中田育男，コンパイラの構成と最適化, 朝倉書店，1999

[5] JAVA, http://java.sun.com/ .

[6] B. Joy, G. Steele, G. Bracha, and J. Gosling, *The Java Language Specification*, Addison-Wesley, 2005.

[7] Eclipse Java development tools, htttp://www.eclipse.org/jdt/ .