Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

デザインパターンを用いた CDI ツールのアーキテクチャとその実現

浦野彰彦† 沢田篤史††
野呂昌満†† 蜂巣吉成††

本稿では，コードインスペクションツール(CDI ツール)のアスペクト指向アーキテクチャを構築し，デザインパターンを用いた実現について議論する．CDI ツールは，ソースコードを静的に解析し，ソースコード中の欠陥の可能性があるコード片を発見するソフトウェアである．CDI ツールには，検査精度の向上や，空間効率の改善などの様々な要求が存在する． 我々はこれらの要求をアーキテクチャに横断するコンサーンとして捉えて，デザインパターンを用いて実現し，その妥当性について考察する．

Architectural Design and Implementation of a Code Inspection Tool using Design Patterns

Akihiko Urano† Atsushi Sawada††
Masami Noro†† and Yoshinari Hachisu††

We designed an aspect-oriented architecture for code inspection (CDI) tool and considered its implementation using design patterns. A CDI tool detects defects in source codes using static analysis. We recognize various requirements for CDI tool such as accurate defect detection and space efficiency. We regard them as concerns cross cutting the architecture and implemented them using design patterns.

1. はじめに

ソフトウェアの静的検証技術の一つにインスペクションがある．インスペクションとは，成果物に対して手順化された見直しを行ない，成果物に含まれる欠陥を発見し修正することで，成果物の品質の向上を図る技術である．とりわけソースコードに対するインスペクションであるコードインスペクションは，プログラムを実行せずにソースコード中の欠陥を発見できる点から，ソフトウェアの開発現場で積極的に取り入れられている．

我々は，先導的 IT スペシャリスト育成推進プログラムにおける OJL(On the Job Learning)プロジェクト [1]として，プログラミング言語 Java[2]を対象としたコードインスペクションツール(CDI ツール)の開発を行なってきた．CDI ツールは，ソースコードに対して意味解析や制御フロー解析，データフロー解析を行なうことで，ソースコード中の欠陥の可能性があるコード片を発見するソフトウェアである．CDI ツールで実現される検査には，プログラムの不正な処理を発見するものや，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．

CDI ツールの目的は，ソースコード中の欠陥の可能性がある箇所を指摘し，ソースコードの品質向上を手助けすることである．ソースコードに求められる品質は，プロダクトのドメインによって異なることから，CDI ツールの検査項目に対する要求は多様に変化する．このような要求の変化に対応するために，検査項目の追加や変更に対する柔軟性を担保することを目的として，共通部分と変動部分から成る CDI ツールのアーキテクチャを構築した．

CDIツールの検査項目に対する要求の中には，デッドコードやヌル参照の検出など，プログラムの実行時に決定する変数の値の情報が必要なものが存在する．このような検査項目に必要な情報を構築するための方法として，我々は定数伝播の導入を考えた．定数伝播は，ソースコード中の変数の値を定数値の伝播によって特定する技法である．

CDI ツールに対する非機能要求として，数十万行を超えるような大規模なソースコードに対する検査を可能にするための，CDI ツールの空間効率の改善が存在する．

本研究では，検査処理への定数伝播の導入と，CDI ツールの空間効率の改善を，CDI ツールのアーキテクチャに横断するコンサーンとして捉え，アスペクト指向アーキテクチャを構築した．構築したアーキテクチャに基づく，着目すべきコンサーンに対応したデザインパターン [3]による実現について議論する．

† 南山大学数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
†† 南山大学情報理工学部
Faculty of Information Sciences and Engineering, Nanzan University

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

2. CDI ツールとそのアーキテクチャ

2.1 CDI ツールに対する要求

CDI ツールは，ソースコードに対して静的に解析を行ない，欠陥の可能性があるコード片を指摘するソフトウェアである．

ソースコードに求められる品質は様々であり，それに応じて検査の視点も様々であることから，CDI ツールの検査項目に対する要求は多様に変化する．CDI ツールが行なう検査には，プログラムの不正な処理を発見するものや，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ことから，多くの検査項目を提供できることが望ましい．多くの検査項目を備えた CDI ツールを実現するためには，検査に共通の処理やコンポーネントの再利用が不可欠である．

2.2 共通部分と変動部分の定義

CDI ツールのアーキテクチャを設計する際に，我々は CDI ツールの共通部分，及び変動部分の分析，定義を行なった．

CDI ツールの処理の概略を図 1 に示す．CDI ツールは，入力されたソースコードに対して字句解析，及び構文解析を行ない，抽象構文木を構築する．さらに，抽象構文木を基にフロー解析を行なうことで，制御フローグラフとデータフローグラフを構築する．CDI ツールの検査処理は，構築された抽象構文木，制御フローグラフ，データフローグラフを走査することで，欠陥の可能性があるコード片を検出する．

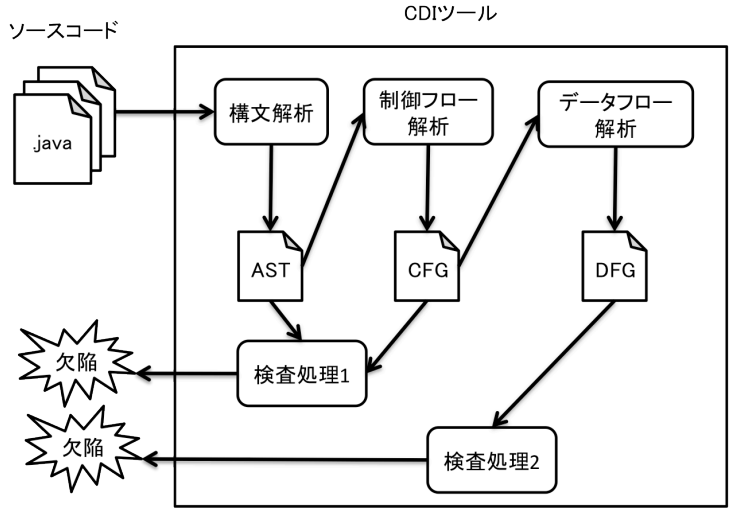

図 1 CDI ツールの処理

字句解析処理，構文解析処理，フロー解析処理や，これらの処理から出力される成果物は，検査の対象となるプログラミング言語の言語仕様や文法によって決定することから，変更の頻度が低い．一方で検査処理は，CDI ツールに対する検査要求によって多様に変化するので，追加や変更の頻度が高い．

このような方針で CDI ツールを分析した結果，共通部分と変動部分を以下のように定義した．

- 共通部分
  - ●構文解析処理
  - ●抽象構文木
  - ●フロー解析処理
  - ●フローグラフ
- 変動部分
  - ●検査処理

2.3 アーキテクチャの定義

前節で，抽象構文木をはじめとする検査対象データを CDI ツールの共通部分として定義し，検査処理を CDI ツールの変動部分として定義した．我々は，CDI ツールの共通部分と変動部分を適切にモジュール化するために，CDI ツールのアーキテクチャをアスペクト指向に基づき設計した．検査対象データを CDI ツールのコアコンサーンとして捉え，検査処理をコアコンサーンに横断する手続きコンサーンとして分離した．

定義した共通部分と変動部分に基づいた CDI ツールのアーキテクチャの静的構造を図 2 に示す．

検査対象データである抽象構文木を，Composite パターンと Interpreter パターンを組み合わせることで実現した．Composite パターンを適用することで抽象構文木の構造を定義し，Interpreter パターンを適用することで抽象構文木に対する走査順序を定義した．また，検査対象データに対する検査処理を手続きとして分離し，Visitor パターンを適用することで実現した．検査対象データから検査処理を分離することで，検査対象データに変更を加えずに，検査処理の追加や変更ができる．

2.4 CDI ツールの実現

我々は設計したアーキテクチャに基づき，CDI ツールを実現した．CDI ツールは，統合開発環境ソフトウェア Eclipse[4]のプラグインとして実現した．この CDI ツールでは 36 項目の検査項目が実現されており．その検査結果は検査レポートとして出力される．また，ソースコード中の欠陥の可能性がある箇所にマーカーを表示し，検査レポートとの対応関係がわかるようにユーザインタフェースを実現した．CDI ツールの画面イメージを図 3 に示す．

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

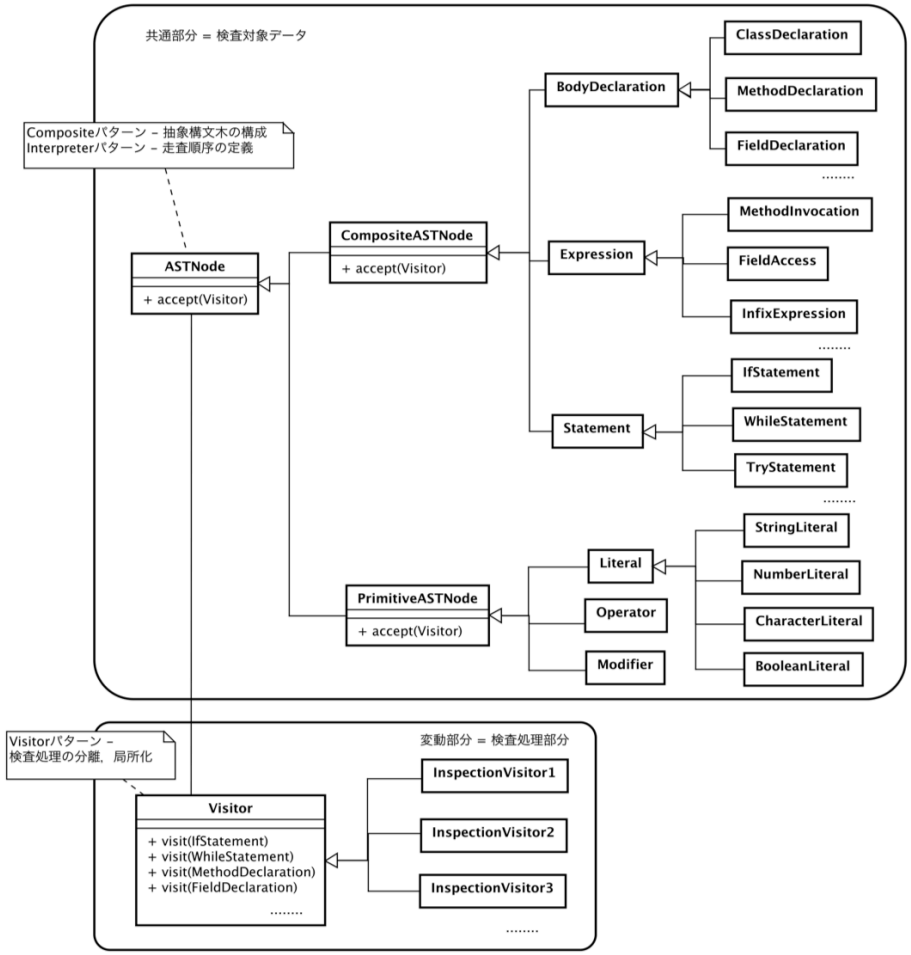

図 2 CDI ツールのアーキテクチャ

3. 定数伝播

定数伝播はコンパイラの最適化技法の一つであり，ソースコード中の変数の値を既知の定数値に置き換えることで，プログラムの最適化を図る方法である．定数伝播によって変数の値を計算し，その計算結果を利用することで，高度な検査処理の実現が可能になる．

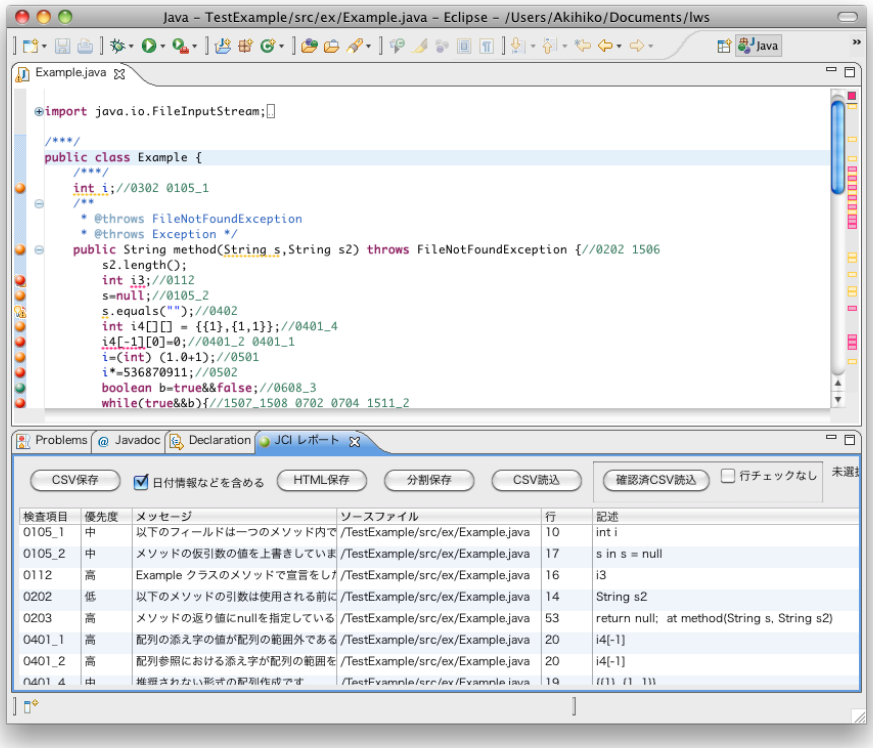

図 3 CDI ツールの画面イメージ

3.1 定数伝播の導入による効果

CDI ツールの検査処理において定数伝播の計算結果が利用可能になることで，以下のような欠陥が発見可能になると我々は考えた．

- デッドコード
- ヌルオブジェクトに対する参照
- 配列の範囲外参照

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

欠陥の例として，ヌルオブジェクトに対する参照について説明する．図 4 のコード例では，メソッド m1 の中で，メソッド m2 にヌルオブジェクトを引数として渡している．引数がヌルオブジェクトである場合，メソッド m2 の戻り値はヌルである．メソッド m2 の戻り値を代入した obj2 に対する参照はヌル参照となる．各文に到達した時の変数の値を，抽象構文木やデータフローを基に計算することで，ヌルオブジェクトに対する参照を検出することが可能である．

```
void m1() {
  Object obj1 = null;
  Object obj2 = m2(m1);
  System.out.println(obj2.toString);
}

Object m2(Object obj) {
  Object anObj = obj;
  if(anObj != null)
    anObj = new Object();
  return anObj;
}
```

図 4 ヌルオブジェクトに対する参照

3.2 計算処理の導入

定数伝播による計算処理は，複数の検査処理で利用されることから，複数の検査処理に横断する手続きコンサーンとして捉えることができる．

この手続きは，制御フローグラフを走査しながら，抽象構文木やデータフローグラフを基に，変数が持つ値を計算することで実現できる．抽象構文木やフローグラフに対する走査は，Visitor パターンによって既に実現されているので，検査処理に新たに Visitor を追加することで定数伝播の計算処理を実現する．

3.3 計算結果データの導入

定数伝播の導入にあたり，検査対象データに対する定数伝播の計算結果の導入方法について比較，検討を行なった．

3.3.1 抽象構文木のデータ構造の変更による導入

定数伝播の計算結果を，抽象構文木のデータ構造を変更することで導入した場合の静的構造を図 5 に示す．

定数伝播の計算結果を，抽象構文木のデータ構造を変更することで導入する方法として，計算結果のデータを表すフィールドを文要素に追加する方法がある．この方法では，構文要素間の依存関係を変更することなく定数伝播処理の計算結果を導入できるので，既存の抽象構文木に対する走査順序を利用できる．ただし，Visitor や構文要素の振る舞いが，抽象構文木のデータ構造を変更する前と後で変わらないことを検証しなければならない．

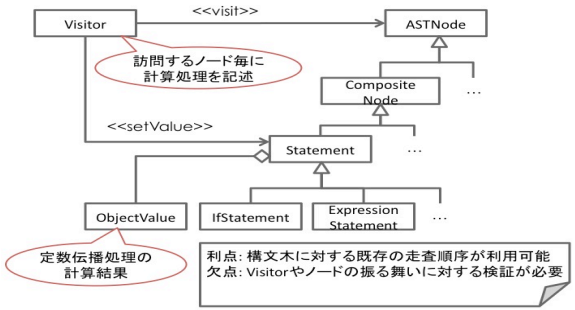

図 5 抽象構文木のデータ構造の変更

3.3.2 検査対象データのアーキテクチャの変更による導入

定数伝播の計算結果を，検査対象データのアーキテクチャを変更することで導入した場合の静的構造を図 6 に示す．

定数伝播の計算結果を，検査対象データのアーキテクチャを変更することで導入する方法として，Decorator パターンを適用する方法がある．この方法では，計算結果のデータを持つ構文要素を，抽象構文木のデータ構造に対して取り外しが可能な形で導入できる．しかし，追加した構文要素に対する走査順序を新しく定義しなければならない．

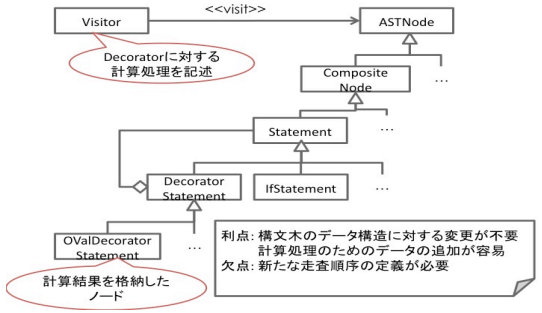

図 6 検査対象データのアーキテクチャの変更

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

3.3.1 導入方法の比較

検査処理のためのデータの追加に対する拡張性や，追加したデータに対する保守性に着目した場合，アーキテクチャを変更する方法を選択するのが妥当である．

定数伝播の計算結果を，抽象構文木やフローグラフと同様に，CDI ツールのコアコンサーンとして捉えた場合，抽象構文木のデータ構造を変更する方法を選択するのが妥当である．

今回は，定数伝播処理の計算結果を CDI ツールのコアコンサーンとして捉え，抽象構文木のデータ構造を変更する方法を選択した．

4. CDI ツールの空間効率の改善

CDI ツールの入力となるものはソースコードであり，その規模も多様である．大規模なソースコードであれば，その規模は数十万行から数百万行にものぼる．このような大規模なソースコードに対する検査処理を可能にするためには，CDI ツールの空間効率の改善は不可欠である．

4.1 空間効率を改善すべき箇所の特定

検査対象データから使用されるメモリが多い箇所を特定し，修正を加えることで CDI ツールの空間効率を改善する．使用されるメモリが多い箇所を調査した結果，以下の二箇所でメモリを大量に使用していた．

- 抽象構文木の末端要素の生成
- 定数伝播の計算結果データの生成

例として，抽象構文木の末端要素の生成について説明する．図 7 のコード例には，public 修飾子，static 修飾子が複数存在する．修飾子を表す構文要素は，情報としてその修飾子のキーワードを持つが，図 7 のようなソースコードを抽象構文木に変換した場合，同じキーワードを持つ修飾子を表す構文要素が複数生成され，メモリを余分に使用することになる．

```
public class ClassA {

  public static int FIELD_A = 0;

  public static int FIELD_B = 1;

   . . .
}
```

図 7 修飾子を表す構文要素

4.2 アーキテクチャの変更による空間効率の改善

検査対象のソースコードが大規模なものである場合，同様のデータが何度も生成されることがわかった．また，CDI ツールの空間効率の改善は，一部の検査対象データを横断する非機能コンサーンとして捉えることができる．一度生成したデータを二度生成しないように，データの生成を制御する必要があるので，Flyweight パターンを適用し，CDI ツールのアーキテクチャを変更することで CDI ツールの空間効率を改善する．検査対象データの一部に Flyweight パターンを適用した場合の静的構造を図 8 に示す．

5. 考察

5.1 要求実現後の CDI ツールのアーキテクチャ

定数伝播の導入と，空間効率の改善を行なった CDI ツールのアーキテクチャの静的構造を図 9 に示す

定数伝播の計算処理は，検査処理から手続きとして分離し，新たに Visitor を追加することで実現した．定数伝播の計算結果のデータは，抽象構文木のデータ構造に変更を加えることで実現した．また，CDI ツールの空間効率を改善するために，抽象構文木の末端要素と，定数伝播の計算結果データに対して Flyweight パターンを適用した．

5.1 設計判断に対する考察

定数伝播の導入や，CDI ツールの空間効率の改善にあたり，検査対象データのデータ構造の一部に対し追加や変更を行なった．データ構造に対する追加や変更を行なう際に，構文要素間の依存関係や，Visitor の抽象構文木に対する走査順序を変更してはならないという制約があった．アスペクト指向による横断的コンサーンの分離と，デザインパターンの適用によって，前述した制約の中で要求を実現することができた．

定数伝播の計算結果データを，検査対象データに導入したことで，定数伝播を用いた，高度な検査処理の開発が可能になった．また，定数伝播の計算処理を Visitor に集約したことで，定数伝播の計算モジュールに対する保守性を確保できたと考えられる．

CDI ツールの空間効率を改善するために，抽象構文木の末端要素と，定数伝播の計算結果データに対して Flyweight パターンを適用し，CDI ツールのアーキテクチャを変更した．このアーキテクチャの変更により，CDI ツール全体で使用されるメモリの約 11 パーセントの削減に成功し，約百万行のソースコードに対する検査が可能になった．

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

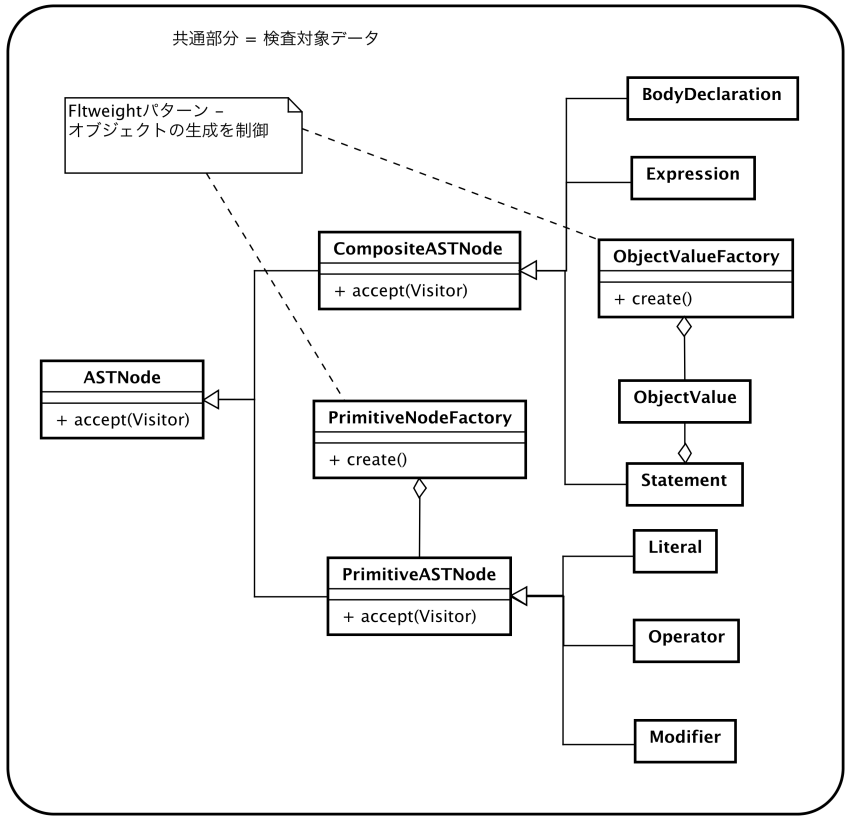

図 8 Flyweight パターンを適用した検査対象データ

6. まとめ

本研究では，CDI ツールの検査処理に対する定数伝播の導入と，CDI ツールの空間効率の改善を行なうために，それぞれの要求をアーキテクチャに横断するコンサーンとして捉え，デザインパターンを適用することで要求を実現した．また，アーキテクチャの変更による要求の実現が妥当であったことを確認した．

今後の課題として，今回実現した要求以外の機能要求，非機能要求に対する，本研究の適用可能性の検証が挙げられる．

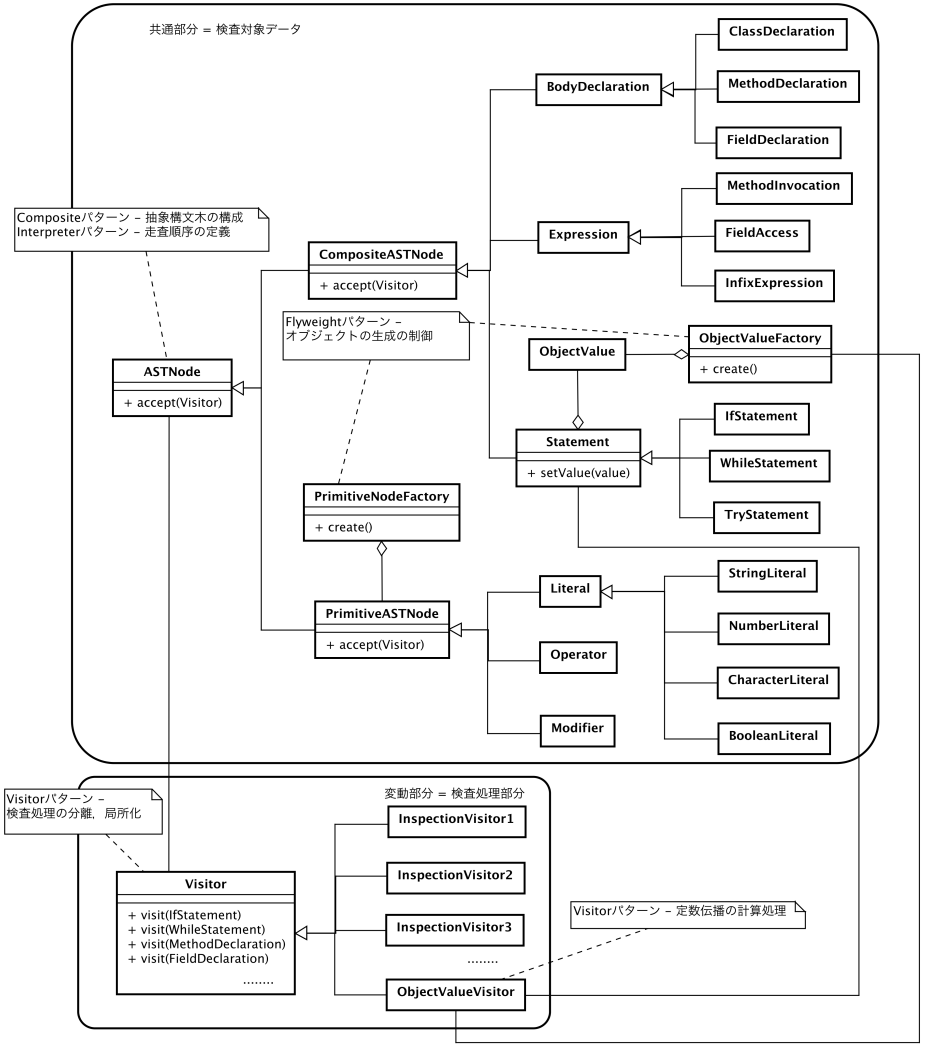

図 9 要求実現後の CDI ツールのアーキテクチャ

Vol.2010-SE-168 No.11
Vol.2010-EMB-17 No.11
2010/6/2

参考文献

1) 小林隆志，沢田篤史，山本晋一郎，野呂昌満，阿草清滋：*On the Job Learning*: 産学連携による新しいソフトウェア工学教育手法，情報システム学会誌，Vol.5，No.2，pp.32-45，(2010).
2) B. Joy, G. Steele, G. Bracha, and J. Gosling: *The Java(TM) Language Specification*, Addison-Wesley, (2005).
3) E. Gamma, R. Helm, R. Johnson, and J. Vlissides: *Design Patterns, Elements of Reusable Object-Oriented Software,* Addison-Wesley, (1995).
4) Eclipse, http://www.eclipse.org