抽象構文木に基づくテストデータ生成支援ツールに関する研究
– OCL を利用した設計 –

2005MT069 水谷 誠　　2005MT095 根本 達也
指導教員　蜂巣 吉成

1　はじめに

コードインスペクション (以下 CDI) ツールに対するテストデータの作成には労力がかかることが知られている．その理由として，CDI ツールに対するテストデータは，大量のソースコードを作成する必要があることが挙げられる．テストデータごとの差異がわずかであり，１つのテストデータをもとに差異にあたる部分を変更し，新たなテストデータとなるソースコードを作成することができる．このようなテストデータ作成の作業を自動化できるツールを用いてテストデータ作成時の労力を削減する事が考えられる．テストデータを作成する場合，ソースコードを変更する際に構文規則に違反しないようにすることや，変更箇所として抽象構文木上の構文要素の位置関係等の特定の条件に合致した構文要素を指定することが求められる．これらを実現するツールを作成することで労力を削減することができるが，CDI ツールには様々なものがあり，利用するテストデータの言語の文法や，変更で行う操作等の要求が異なる．

本研究では，１つのテストデータをもとに変更を加え，複数のテストデータを生成するツールを提案し，CDI ツールによって異なる要求に柔軟に対応できるアーキテクチャの提案を行う．我々は提案するツールの中でも，ソースコードから変更箇所を特定し，変更を行う内部処理の設計を行う．抽象構文木に対して操作することでソースコードを構文規則に違反することなく変更する方法を用いる．変更箇所の特定に OCL を利用することで，テストデータの文法が変更された際に処理を変更する必要がなく対応しやすい設計を行う．本研究は，次のように進める．

1. テストデータの特徴の整理
2. テストデータ生成支援ツールのアーキテクチャの提案
3. アーキテクチャの拡張性の考察

2　背景技術

2.1　CDI ツール

CDI ツールとは，ソースコード中の欠陥が潜んでいる恐れがある箇所を発見するツールである．プログラムに対し静的解析することにより欠陥を検出する．

2.2　CDI ツールのテストデータ

CDI ツールの欠陥を検査する機能に対するブラックボックステストでは，テストデータとしてソースコードを用いる．

2.2.1　設計手法

CDI ツールに対するテストデータは，１つのテストデータの一部を変更することで新たなテストデータが作成できるという特徴がある．変更する箇所は，CDI ツールで検査する欠陥をもとに決定する．CDI ツールに対するテストデータの設計手法を，例をあげて説明する．

検査する内容として「&&演算子と||演算子を&演算子や|演算子と間違えていないかを検査する．ループ文の条件式内で&演算子または|演算子を使用している場合を検出する.」という仕様があった場合，ループの条件式内の中置演算子の種類が検査結果に影響することがわかる．欠陥の可能性があるとして CDI ツールが検出する場合は次の２通りになる．

- 条件式内で&演算子を使用しているループ文がある場合
- 条件式内で｜演算子を使用しているループ文がある場合

したがって，CDI ツールに対するテストデータは，条件式内で&演算子を使用しているループ文を含むソースコードである．また，演算子の種類によって検査結果が変わるので，演算子を変えたソースコードを作成する必要がある．また，ループ文には while 文，for 文，do 文の３種類があるので，演算子を&&，&，||，|に変えたテストデータそれぞれにループ文の種類を変えたテストデータを作成する必要がある．このように，１つの検査項目に対して，テストデータごとの差異がわずかであるが，12 個のソースコードを作成しなければならず，多くの労力を要する．

2.2.2　変更箇所

変更対象となる箇所は，抽象構文木中の構文要素の位置関係によって表現できる．次の情報を組み合わせて，変更箇所を表現する．

- 構文要素の種類
- 終端記号の種類
- 親子関係
- 子孫関係

2.2.1 節にあげた例で変更箇所はそれぞれ，「ループ文の構文要素の条件式の子孫の中置式の構文要素の終端記号演算子」，「ループ文の構文要素」である．

2.2.3 変更内容

CDI ツールで検出する欠陥は，構文要素の種類や有無によって左右される．種類や有無を変える操作として，構文要素の削除と置換の操作を行う．

2.2.1 節であげた例では次の操作になる．

- 中置式に対しては演算子を，&&, &, &, ||, | に置換
- ループ文に対してはループの種類を，while，for，do に置換

2.3 OCL

OCL[3] は，UML の一部であり，定義されたモデルに，図では表現できない制約を記述することができる．クラスや操作，属性，関連端等のモデル中の要素を用いて記述する．記述した条件は，作成するプログラム中に条件をチェックする操作を埋め込むことで，プログラムの動作を検証することに用いられる．

オブジェクトに対して制約を評価する処理を行い，プログラムの動作を検証することをサポートする既存のツールとして，EclipseMDT の OCL プラグイン [2] 等がある．

3 OCL を用いたテストデータ生成支援ツールの提案

テストデータ作成時の労力を削減する方法として，本研究では，ソースコードに対して抽象構文木上で変更箇所を特定し，変更を加えることでテストデータを生成するツールを提案する．

本研究では，提案するツールの内部処理のアーキテクチャを設計する．抽象構文木は，ソースコード中の構文要素を構文規則に従って木構造にしたもので，抽象構文木上で変更を行うことで構文規則に違反することなくソースコードを変更することが可能である．変更箇所と構文要素の照合には OCL を利用することで，文法の変更に容易に対応できる設計を行う．

3.1 ツールの概略

提案するツールを実現する上で, 必要な情報をまとめる．

3.1.1 処理の流れ

提案するツールの処理の全体図を図 1 に示す．

提案するツールの処理の流れは，次のようになる．

1. 入力されたソースコードを構文解析し，抽象構文木を構築する．
2. 入力から渡されたテストデータの仕様の中間形から，変更箇所を OCL 式に変換する．
3. 構築した抽象構文木を深さ優先探索で走査し，ノードと OCL 式を照合する．

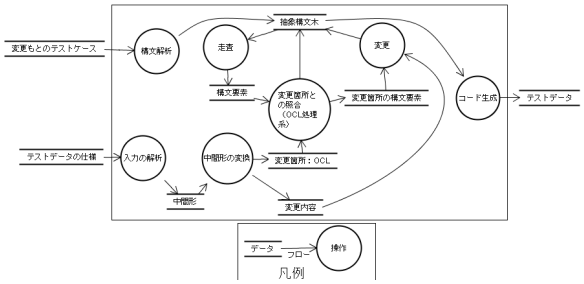

図 1　ツールの処理の全体図

4. 照合した結果，ノードが変更箇所であった場合，抽象構文木に変更を加える．
5. 変更後の抽象構文木からコードを生成する．

本研究ではユーザインターフェイスの方式については扱わない．入力を解析した結果は共通の中間形に変換されるとし，中間形以降の処理の設計を行う．

3.2 入出力

提案するツールの入力として変更対象となる Java ソースコードと，ソースコードへの変更情報としてテストデータの仕様を入力する．テストデータの仕様は，2.2 節であげた変更箇所と変更内容の組によって構成される．これらの入力をもとに，テストデータの仕様に基づいて変更を加えた Java のソースコードを，テストデータとして出力する．

１つのテストデータの仕様で複数の変更箇所と変更内容の組を複数入力した場合は，変更を加えたものそれぞれに更に次の変更を加える．

3.3 非機能要求

CDI ツールには様々なものがあるので，CDI ツールごとに違う要求に柔軟に対応できる設計を行いたい．CDI ツールごとの要求の違いとして，Java のバージョンの違い等による文法の違いと，変更で行う操作の違いがあげられる．これらのことを考慮して設計する．

3.4 内部処理の設計

処理の流れを基に，3.3 節であげた非機能要求を考慮しデザインパターン [1] を利用した設計を行う．

3.5 テストデータの仕様の中間形

中間形のデータ構造は，テストデータの仕様で入力される変更箇所と変更内容を入力に依存せずに表現することが求められるので，それぞれの要素は文字列で保持する．変更内容の情報を保持する変更対象クラスと，対象との関係を表す関係要素クラスを作成する．図 2 に中間形のクラス図を示す．

「変更対象」クラスがソースコードにおいて変更される構文要素を表す．「変更内容」の値は置換，または削除である．置換の場合は「置換対象」で，置換後の値を表す．「終端記号」は，構文要素に加えて構文要素が持つ終

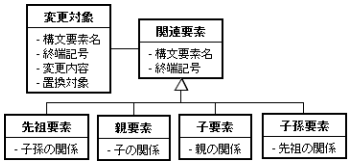

図 2　中間形のクラス図

端記号を指定する場合に，終端記号名を表す．「関連要素」クラスのサブクラスは変更対象の構文要素に親子関係や子孫関係の制約を加える場合に用いる．例として，2.2.2 節にあげた変更箇所を中間形のデータ構造を図 3 に示す．

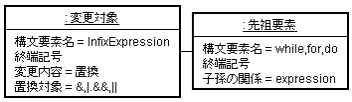

図 3　中間形データ構造

変更対象のオブジェクトは中置式の演算子を&&，&，||，|に置換することを表す．この例の場合，ループ文の条件式の中置式の演算子を置換するので，ループ文の条件式であるという祖先要素を変更対象と関連づける．

3.6　抽象構文木の表現

抽象構文木は，変更の操作を行いやすいことが求められるので，構文要素 1 つ 1 つをクラスとして表現することで，構文要素ごとに操作を定義できる Interpreter パターンを用いる．

3.7　OCL を利用した変更箇所の特定

変更箇所を特定する処理では文法の変更に対応しやすいことが求められる．変更箇所と構文要素の照合に OCL を利用する．中間形を OCL 式に変換し，抽象構文木のノードに対して評価した結果の真偽値によって，変更対象であるかを判断する．変更箇所の特定に OCL 式を利用することで，文法を変更した際に処理を変更する必要がなく，文法の変更に対応しやすい．

OCL 式への変換

テストデータの仕様で与えられる変更箇所の親子関係、子孫関係を OCL 式で表現する．テストデータの文法に依存しない OCL 式の雛型を用意し，変更対象ノードや関係するノードが持つ情報を雛型の OCL 式に当てはめて OCL 式を生成する．以下に OCL の雛型を示す．

ocl 式の雛型

self.oclIsTypeOf(＜変更対象の構文要素名＞)
＜関連＞.oclIsTypeOf(＜関連要素の構文要素名＞)
self. ＜親要素との関係＞ = self
self. 終端記号 = ＜終端記号名＞

＜＞で囲んだ部分に中間形の要素を代入することで OCL 式を生成する．これらの式を組み合わせ，and や or を利用して，複雑な条件を指定する．例として，図 3 の中間形を変換した OCL 式を次に示す．

```
self.oclIsTypeOf(InfixExpression)
```

図 4　変更の対象の OCL 式

```
(parent.oclIsTypeOf(WhileStatement) or
 parent.oclIsTypeOf(ForStatement) or
 parent.oclIsTypeOf(DoStatement) ) and
parent.expression = self
```

図 5　先祖要素の OCL 式

図 4 の OCL 式は，変更対象の中置式を指定する OCL 式であり，図 5 は対象の先祖要素を指定する OCL 式である．

3.8　構文要素の変更

変更の操作では，新たな操作を追加しやすいことが求められる．また，1 つの操作でも構文要素ごとに違う処理を作成する必要があるので，構文要素ごとに処理の違う同じ操作を 1 つのクラスにまとめることができる Visitor パターンを用いる．

3.9　内部処理のアーキテクチャ

各処理の設計をまとめた，内部処理のアーキテクチャのクラス図を図 6 に示す．

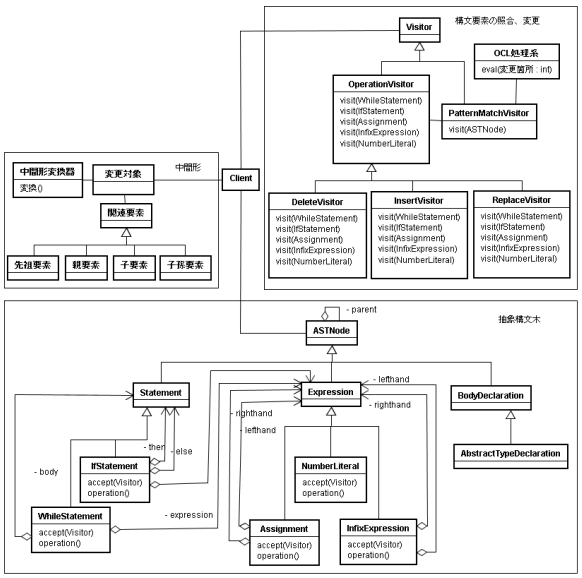

図 6　内部処理のクラス図

制御は，入力からテストデータの仕様を受け取り，照合処理に OCL 式を渡し，抽象構文木に Visitor を渡す操作を行う．

4　考察

4.1　アーキテクチャの考察

操作の追加や文法の変更を行う際に必要な作業を考察し，非機能要求が達成できているかを考察する．

操作の追加を行う場合

抽象構文木を変更する操作を定義する方法として，本研究では Visitor パターンを用いた．Visitor パターンを利用しない方法として，抽象構文木のそれぞれのノード

に操作を定義する方法が考えられる．それぞれの方法に対して操作の追加を行う際に，変更を行う箇所の違いを図 7.8 によって示す．抽象構文木は一部のみ示す．

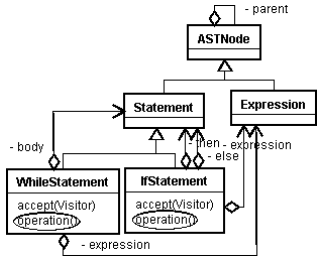

図 7　Visitor パターンを使わない場合の変更箇所

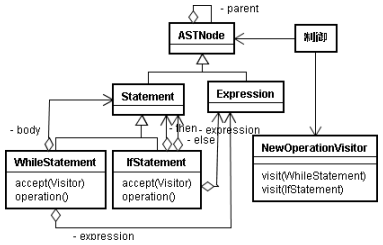

図 8　Visitor パターンを使った場合の変更箇所

楕円で囲んだ箇所が操作を定義する場所である．抽象構文木のそれぞれのノードに操作を定義する場合，全てのノードクラスに操作を定義する必要がある．Visitor パターンを利用する場合は，操作を定義する箇所が 1 つの Visitor クラスに局所化され，操作の追加が容易になる．

文法の変更に対応する場合

Java の文法が変更した際には，抽象構文木の構造が変更される．抽象構文木の変更に伴って変更する必要がある箇所は，構文要素ごとの操作である．構文要素ごとの操作の変更は，Interpreter パターンを利用して構文要素 1 つ 1 つをクラスとして表現したことにより変更を行う箇所が明確になり，変更を容易に行うことができる．本研究では変更箇所と抽象構文木のノードの照合処理に OCL を利用している．OCL を利用した場合と，利用しない場合の比較を行う．

OCL 式を評価する処理を利用せずに構文要素の関係の照合を行う場合，構文要素によって関係する要素が異なるので，それに応じた処理を作成する．新たな構文要素を追加した場合，新たに関係を調べる処理を作成する必要がある．

OCL を評価する処理を利用する場合，構文要素の関係をクラス間の関係として OCL 式で表し，評価を行う．OCL を利用することで，構文要素の関係を簡単に表現する事ができる．OCL を利用して関係を照合する場合は，抽象構文木が変更された場合でも，OCL 式で表現することができれば，処理をそのまま利用できる．テストデータの仕様の中間形は，構文要素の関係をクラスで表し，構文要素名や関係等の具体的な要素を文字列として保持することで，文法に依存しない構造になっている．OCL 式は，構文要素の関係を表す OCL 式の雛型を用意し，中間形から具体的な要素を当てはめることで生成する．OCL 式への変換は文法に依存しないので，抽象構文木が変更された場合でも，OCL 式で表現することができる．OCL を利用することで照合処理を変更する必要がなく，文法の変更へ容易に対応できる．

4.2　関連研究

ソースコードを変更する方法として，抽象構文木を XML 文書で表現し，既存の XML 処理系を利用する方法 [4] も考えられる．

抽象構文木を XML 文書で表現する場合，XML 文書は元のソースコードよりサイズが大きくなり，更に処理する際には DOM 木を生成して行う．本研究のアプローチでは，ソースコードから構築した抽象構文木を直接操作するので，XML 文書を経由する場合と比べ，操作に必要な手順が少ない．

XML を利用する場合，実装や他の機能を追加する際に，既存の様々な関連技術を利用でき，ソースコードの中間形として汎用性が高い．XML を利用してソースコードを変更する場合，既存の XML 技術を使用すればよいとされているが，具体的な方法については示されていない．本研究のアーキテクチャの構文要素の特定処理を XPath に置き換え，抽象構文木の変更処理は Visitor パターンを XML 文書の DOM 木に対する処理として利用する事で，XML を利用した方法にも適用できる．

5　おわりに

本研究では，OCL を用いてソースコード上から変更箇所を特定し，変更を加えることで CDI ツールに対するテストデータを作成するツールを提案し設計を行った．今後の課題として設計に基づいて実装を行うことが挙げられる．

参考文献

[1] E. Gamma, R. Helm, R. Johnso, and j. Vlissides, *Design Patterns*：*Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[2] The Eclipse Foundationt, *Eclipse Modeling MDT*, 2009；www.eclipse.org/modeling/mdt/.

[3] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, Addison-Wesley, 2003.

[4] 吉田一, 山本晋一郎, 阿草清滋,“XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装,” 情報処理学会論文誌, vol. 44, no. 6, 2003, pp. 1509-1516.