コーディング規約自動検査器のための DSL の提案と実装

伊藤 達也　　指導教員：山本 晋一郎

1 はじめに

ソフトウェアが高機能になるにつれて，ソースコードの量が増加する．例えば，Black Duck Software の調査 (`https://www.openhub.net/`) によると，Web ブラウザの Firefox は約 1,200 万行，MySQL は約 230 万行となっている．これらのソフトウェアは現在も開発が進んでおり，機能の追加に伴いソースコード量はさらに増加することが予想される．

また，ソフトウェアの品質を維持するために，ソースコードを適切に管理する必要があるが手動管理には限界がある．理由として，ソースコード量の増加に伴う管理者の負担増大，管理者ごとに判断基準に差があり，管理基準が一意でないなどがあげられる．この問題を解決するために品質管理の自動化が有効である．

ソースコードの品質管理の項目の例として，コードの可読性や保守・テストの容易性，コンパイル時の警告数があげられる．これらの項目は明確な規則やガイドライン，すなわちコーディング規約を遵守してソースコードを記述することで品質を維持することができる．組み込みシステム向けの MISRA-C 2012 Guidelines [1] や一般的なソフトウェアのセキュリティ確保を目的とした CERT C Secure Coding Standard [2] などの公的なコーディングが存在することから，コーディング規約の導入が有効であると言える．

コーディング規約自動検査器の 1 つに CX-Checker があり，検査ルールを独自に追加できるので，社内規約などの独自ルールに対応しやすい．しかし，現在のルール作成言語にはいくつか問題点があり，ルール作成における負担が大きい．

本研究ではルール作成用 DSL の提案と実装を行い，ルール作成にかかる負担軽減を実現する．現在のルール作成言語の短所を補うことでルール作成にかかる負担を軽減し，生産性の向上を図る．

2 CX-Checker

2.1 CX-Checker とは

CX-Checker [3] とは本学と名古屋大学で共同開発されている C 言語向けのコーディングチェッカーで，Sapid プロジェクトで開発されている．構文・意味解析情報を用いてモデル化することでソースコードを CX-Model 形式の XML に変換し，それを対象にチェックルールを適用することでコーディングチェックを行っている．

2.2 ルール作成

ルール作成言語には XPath と Java の 2 種類がある．

2.2.1 XPath

XPath [4] とは，W3C で開発された XML 文書の特定の部分を指定する言語構文である．ルールの検査対象の位置を簡潔に指定することができる．しかし，指定した複数のノードを比較するといったルール作成に必要な機能が乏しく，作成できるルールに制限がある．また，ルールを作成するために CX-Model の構造を理解する必要がある．

2.2.2 Java

CX-Model でチェック可能なルールは理論上、全て作成することができる．例えば，Misra-C 2012 Guidelines の規約文書内で検査可能とされている 119 ルールのうち 81 ルールが作成対象となる．残りの 37 ルールは CX-Model が未対応なルールである．しかし，ルールを作成するためには Java における DOM 操作や CX-Model の構造に関する知識が必要であり，1 ルールあたりのコード量が多いといった問題もある．

2.2.3 問題点

2 つの言語についてまとめると表 1 のようになる．

表 1　ルール作成言語まとめ

	XPath	Java
ルール作成のしやすさ	○	×
ルール作成に利用できる機能数	少	多
ルールあたりの記述量	20 行前後	100 行以上
作成可能なルール量	少	多
ルール作成に必要な知識	XML, CX-Model の構造	

どちらの言語を利用しても何かしらの問題があり，ルール作成にかかる負担が大きいことがわかる．

3 DSL

ドメイン固有言語 (Domain Specific Language, DSL) [5] とは，特定の作業の遂行や問題の解決に特化して設計されたプログラミング言語のことである．対象領域にそった抽象化を行うことで簡潔に記述できるようになり，品質や生産性の向上が期待できる．一方で，DSL 自体の設計や保守が難しかったり，応用範囲が狭いと言った欠点がある．

4 ルール作成用 DSL

4.1 概要

既存のルール作成言語の短所を補えるような，CX-Checker ルール作成用 DSL の提案と実装を行う．実装方法は CX-Checker と連携のしやすく既存のライブラリを活用できる内部 DSL を選択した．ベース言語は Java との連携が容易で，DSL の実装を行う上で有用な機能を持つ Scala を選択した．また，設計方針として内部データである CX-Model を扱いやすい形に変換しユーザに提供する形式とした．これによって，内部データやその構造をなるべく意識せずにルール作成を行うことができる．

4.2 機能

本 DSL では検査対象のプログラムを**要素**という単位でとらえ，要素を扱うために本 DSL 大きく分けて 3 種類の**機能**を提供する．

4.2.1 検査対象の取得

ルール検査に必要な要素を取得する．

例えば，**goto 文**を取得したい場合，コード 1 のように記述する．

ソースコード 1　DSL による検査対象の取得

```
1 Target := content("goto") in kw
```

`kw` はキーワード要素を，`content()` は指定した文字列と一致する要素を取得する機能である．これらを`in`結合子を使って組み合わせることで、検索対象の要素を取得する．

4.2.2 型情報の取得・加工

型情報とは，識別子が持つ型に関する情報をまとめたものである．この情報を利用することで，利用しない場合に比べて型や変数参照に関するルールを容易に記述できる．しかし，型情報は型情報タグの属性として保持されている．利用するには複雑な条件式を用いる必要があり，容易に記述できるとは言い難い．

そこで本 DSL では予め型情報をルール作成に利用しやすい

データ構造に変換し，検索用機能を用意した．

例えば，**可変長配列**という要素を取得したい場合，Java ではコード 2 の様な，複雑な条件式を用いて型情報を取得するが，ルール作成用 DSL ではコード 3 の様に記述することで取得できる．

ソースコード 2　Java による型情報取得

```
1 if ((fstChild.getNodeName()
2   .equals("TypeInfo"))
3   && (((Element) fstChild).getAttribute(
4     "sort").equals("array"))
5   && !(((Element) fstChild).getAttribute(
6     "array_size").matches("^[0-9]+")))
```

ソースコード 3　DSL による型情報取得

```
1 Target := pTiArraySize("\\D")
2   and tiSort("array")
```

4.2.3　取得した検査対象への処理

取得した要素に対して処理を行う．処理内容は作成するルールによって変化するので，利用者が独自に定義した機能を本 DSL に組み込むことができる．作成には Scala を利用する．

5　評価

本 DSL の記述性と実行性能の評価を行うために既存言語との比較を行った．記述性の評価の為に作成可能なルール数の比較を，実行性能の評価の為にルール検査時のメモリ使用量と実行時間の比較を行った．

5.1　記述性の評価

公的な規約文書 MISRA-C 2012 Guidelines にどれだけ対応しているかについて評価を行う．

規約文書が検査可能としている 116 ガイドラインから CX-Checker が対応していない 35 ガイドラインを除いた，81 ガイドラインを対象にルール作成可能かどうかの確認を行う．結果は表 2 の通り．DSL の記述力は Java と同等であるといえる．

表 2　作成可能ルール数

	XPath	Java	DSL
ルール作成可能数	25	68	67

5.2　実行性能の評価

5.2.1　条件

共通の条件を以下に示す．

検査対象　bison-1.25, dhrystone-2.1, gawk-3.0.1, gnugo-1.2, gzip-1.2.4, patch-2.5 のソースプログラム総計 91 ファイル
ファイルあたりのコード行数　最小：6 行 / 最大：5612 行
総行数　50517 行

検査ルールは 2 つ作成し，XPath と Java はそれぞれ 1 つずつ DSL は両方作成した．XPath で作成したルールは 91 ファイル中 11 ファイル，Java で作成したルールは 8 ファイル検出した．検出したファイルとそれ以外で挙動が変化したので，分けて比較を行った．

5.2.2　検出箇所あり

XPath　概ね行数に比例して増加する．
Java　メモリ使用量は一定だが，実行時間は行数に比例して増加する
DSL　概ねモデルサイズに比例して増加するが，サイズ以外の要因が大きい．

5.2.3　検出箇所なし

XPath　行数にかかわらずほぼ一定
Java　行数にかかわらずほぼ一定
DSL　行数等との相関がみられない．関係性を見つけるには内部要素のさらなる調査が必要．

5.2.4　言語間の比較

ファイルの行数に対する，検査時のメモリ使用量及び実行時間 (5 回計測した平均値) について比較を行った．グラフは図 1 で，縦軸はメモリ使用量 (Byte) と時間 (秒)，横軸はファイルの行数である．

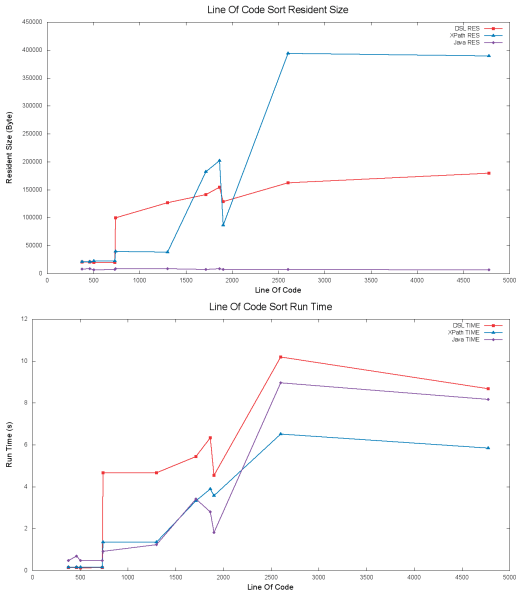

図 1　言語間の比較

メモリ使用量は概ね XPath と Java の中間あたりと言える．実行時間は XPath や Java と比べて時間がかかっているが，平均 1.5 秒の増加にとどまっている．以上の結果より，本 DSL は実用に耐えるものだといえる．

6　まとめと今後の課題

CX-Checker のためのルール作成用 DSL の提案と実装を行った．内部データを扱いやすい形式に変換しユーザに提供することで，ルール作成にかかる負担の軽減と生産性の向上を実現した．本研究は SIG-SE にて発表予定である．

今後の課題として，検査対象に対する処理を行う機能を追加することで作成可能なルールを増やすこと，また，本 DSL を内部データを意識することなくルール作成を行える形式に発展させるの 2 点が挙げられる．

参考文献

[1] Misra, *MISRA C 2012 : Guidelines for the Use of the C Language in Critical Systems*, Motor Industry Research Association, 2013, 226pp
[2] JPCERT, CERT C コーディングスタンダード, `https://www.jpcert.or.jp/sc-rules/`
[3] CX-Checker, `http://cxc.sapid.org/`
[4] W3C, XML Path Language (XPath) Version 1.0, `http://www.w3.org/TR/xpath/`
[5] Debasish Ghosh, 『実践プログラミング DSL ドメイン特化言語の設計と実装のノウハウ』佐藤 竜一 (監修, 翻訳), 翔泳社, 2012, 400pp