ソフトウェア構築ログ解析器を用いたソースプログラム解析

永井 郁哉[1]　大久保 弘崇[2]　粕谷 英人[2]　山本 晋一郎[2]

概要：我々はソフトウェアの構築過程を表現するモデルを提案している．これは大規模・複雑化したソフトウェアの構築過程におけるファイルの依存関係を統一的に調査・分析することを可能にする．本論文では，このモデルの応用として Sapid のソースプログラム解析過程を置き換える，新たな手法を提案する．モデルを用いることでファイルの依存関係を正しく把握することができるため，従来の方式では解析が不可能であったソフトウェアに対しても正確な解析が可能となる．現状で Sapid のソース解析に不必要なコマンドとオプションが原因で SDB 作成に失敗する問題に対して，ログを用いて SDB を正しく構築できることを確認した．

キーワード：ソフトウェア構築ログ，ソフトウェアリポジトリ

1. はじめに

細粒度リポジトリに基づいた CASE ツール・プラットフォーム Sapid[1] は，C 言語に対する CASE ツールを構築するためのフレームワークである．その主要な部分は，C ソースプログラムを解析し，その結果を独自のリポジトリ (SDB と呼ぶ）に格納するコマンド sdb4 である．sdb4 コマンドは，cc や ld といったソフトウェア構築に用いるコマンドと差し替えて実行することで，ソフトウェアの構築の代わりにソースの解析を実行するように設計されている．そのため，例えばソフトウェアの自動構築ツールの一つである make コマンドで，CC=cc のようにコンパイラが変数により指定されている Makefile により構築規則が作られている一般的な場合では，make CC=sdb4 として起動することにより，cc によるソフトウェア構築の代わりにそのソフトウェア全体を sdb4 により解析することができる．しかし，make が再帰的に実行されるソフトウェアの構築においてこの方法で SDB を作成するとき，Sapid はファイルの依存関係を把握することができず，make が呼び出された各ディレクトリに SDB を作成しソフトウェア全体の SDB を正確に作成することができない．

また，コンパイラを sdb4 に置き換えるだけでは make によって Sapid のソース解析に不必要なコマンドが起動される．これが原因で SDB の作成に失敗する場合があり，ユーザは構築ルール記述ファイルを修正する必要がある．

ソフトウェアの構築のための手順や命令が複雑化し，ファイルに与える影響やファイルの依存関係も複雑化している．一般にこの工程は make コマンドを代表とする自動構築ツールによって管理される．自動構築ツールは構築に必要なコマンドに多くのオプションを指定して実行するため，構築ルール記述ファイルも大規模化しこのファイルから依存関係を把握するための静的解析が困難となっている．我々はこの問題を解決するために構築ログを解析し，モデル化 [2] することでファイルの依存関係を読み解くことに費やす時間を短縮した．

本論文では提案したモデルを応用し，Sapid に利用することを考える．モデルが持つファイルの依存関係を利用することでソフトウェアの SDB を正確に作成する．また，ソフトウェア構築ログをモデル化する解析器を利用することで Sapid のソース解析に不必要なコマンドとオプションをフィルタリングし，ユーザが構築ルール記述ファイルを理解し，SDB を構築できるよう修正する行為を取り除く．

2. ソフトウェア構築ログのモデル

ソフトウェア構築ログを表すモデルの UML 図を**図 1** に示す．ソフトウェア構築ログは**コマンド**，**オプション**，**引数**の 3 項組から成り立つと考える．コンパイルされるソースファイルとそのコマンド実行による生成物はファイルで

[1] 愛知県立大学大学院 情報科学研究科
Graduate School of Information Science and Technology, Aichi Prefectural University
[2] 愛知県立大学 情報科学部
School of Information Science and Technology, Aichi Prefectural University

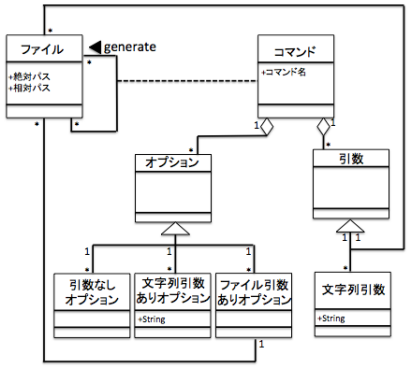

図 1　ソフトウェア構築ログの UML クラス図

```
CC = gcc
software: array.c
          CC -MD -MP -MF .deps/array.Tpo -c \
          -o array.o array.c
          mv -f .deps/array.Tpo .deps/array.Po
```

図 2　Makefile

ある．ファイルを引数に取るコマンドを実行することで新たなファイルが生成される関係を *generate* とする．

ファイルの依存関係を明確化するためにそれらの**絶対パス**，**相対パス**の情報が必要である．パスを取得するために我々は make によって出力される実行詳細に着目した．make を実行する際にオプションとして-w を渡すことでコンパイラやファイル操作コマンドの実行パスを知ることができる．

3. ソフトウェア構築ログによる SDB 作成

SDB は Makefile を利用して作成される．多くの場合，Makefile 内に存在するコンパイラのマクロ定義を sdb4 に置き換える．しかし，sdb4 はコンパイラに付随する全てのオプションを忠実に実行することができず，SDB の作成に失敗することがある．

一例として**図 2** を考える．gcc はコンパイルと同時に翻訳リソースファイルを生成し，その後，翻訳リソースファイルを mv で操作する．

単純に gcc を sdb4 に置き換えて make を実行すると本来 gcc の起動によって生成される翻訳リソースファイルが構築されず，mv の引数として使用されるファイルが存在しないことで SDB の作成に失敗する．

この場合，-MD，-MP，-MF とその引数，さらに mv コマンドの一行が起動されないように Makefile を修正する必要がある．しかし，Makefile のような構築ルール記述ファイルは大規模化しており，これを静的解析し，修正することは困難である．また，構築ルール記述ファイルの内容を変更することは知識も必要となる．

そこで我々はソフトウェアの構築ログから SDB を作成することを考える．ソフトウェア構築ログの解析から SDB を得るには実行されたコマンドがコンパイラであるか，そうでないかを判断し，コンパイラであれば sdb4 に置き換えて実行する必要がある．ソフトウェア構築ログを解析後，起動されたコマンドを判断する知識を解析器に与えることでこれを実現する．解析器がコンパイラを sdb4 に自動で置き換えることでユーザがコンパイラのマクロ定義を書き換える代わりをする．

さらにソフトウェア構築ログの解析と同時に解析器が SDB を作成することに必要なコマンドとオプションを取捨選択する．この機能を実現することでコンパイル時に生成される翻訳リソースファイルを無視することができ，翻訳リソースファイルを操作するコマンドも無視することが可能となる．これにより解析に不必要なコマンドとオプションの実行で SDB の作成に失敗する問題を解決することができると同時に，Sapid ユーザが Makefile を静的解析し，構築ルール記述ファイルを変更する必要がなくなる．

4. おわりに

本論文では Sapid の SDB 作成工程に着目し，その問題を指摘し，解決策を提案した．Sapid のソース解析に不必要なコマンドとオプションが原因で SDB 作成に失敗する問題に対し，ログを用いて SDB を正しく構築できることを確認した．

現在，以下の評価方法を検討している．オープンソースのソフトウェアを用いて，現状の SDB 作成方式より本手法の SDB 作成方式の方が多くのソフトウェアに対し SDB を構築することが可能であることを確認する．また，ファイルの依存関係を基に SDB の作成を実現した上でオープンソースを用いて正確な SDB が作成できているかを確認する．

今後の課題を以下に挙げる．第一に引数やファイルが持つ絶対パスと相対パスを用いた SDB の作成を実装することである．現状ではソフトウェア構築ログに出現する引数やファイルの絶対パスと相対パスを取得することは実現できている．しかし，この値を利用した SDB の作成を実現するまでには至らなかった．第二に本手法によって SDB 作成が失敗した場合のエラー報告である．SDB の作成に失敗した原因とその対処方法をユーザに提示することでさらに SDB 作成にかかる手間を省略することができる．SDB の作成に失敗する原因を分類し，エラーに応じて対処方法を提示することを実現することで解決を図る．

参考文献

[1] 福安直樹，山本晋一郎，阿草清滋：細粒度リポジトリに基づいた CASE ツール・プラットフォーム Sapid，情報処理学会論文誌，Vol. 39, No. 6, pp. 1990–1998 (1998).
[2] 永井郁哉，大久保弘崇，粕谷英人，山本晋一郎：実行履歴に基づくソフトウェア構築モデルの提案，ソフトウェアエンジニアリングシンポジウム 2013 論文集，Vol. 2013, pp. 1–2 (2013).