ソフトウェア工学 102-7
（1995．1.26）

細粒度リポジトリに基づいた ツール・プラットフォームとその応用

山本 晋一郎　　阿草 清滋

名古屋大学工学部

本稿では，細粒度のソフトウェア・リポジトリに基づいたCASEツール・プラットフォームSapidを提案し，細粒度ソフトウェア・リポジトリのモデル，およびリポジトリを構成するSDB, AR, SMLの概要について説明する．また，提案したリポジトリを用いた，ソフトウェア操作エディタ，仕様書雛型生成ツール，program slicingツールなどの応用について解説する．

CASE Tool Platform based on Fine Grained Software Repository and its Applications

Shinichirou YAMAMOTO　　Kiyoshi AGUSA

School of Engineering, Nagoya University
Nagoya, 464-01, JAPAN

In this paper, we propose CASE tool platform called Sapid which based on fine grained software repository. First, we clarify our software repository model and give brief explanation about Sapid's sub system SDB, AR and SML. Second, we give three application tools: software manipulating editor, functional specification generator and program slicing tool using Sapid to show effectiveness of our approach.

1 はじめに

ソフトウェアの生産性向上を目指して，リポジトリによるデータ統合，トースタモデルによる制御統合などの各種の標準化作業が行なわれている[2, 7]が，これらの統合化は比較的大きな粒度の成果物を対象としている．例えば，一般的なリポジトリにおける最小単位はモジュールであり，そのモジュールの内部構造や構成要素などについてのデータ管理は行なわれていない．しかし，統合化の対象とならなかった細粒度の構成要素もソフトウェア・システムにおいて重要な役割を果たす．特に，設計以降のフェーズではこのような細粒度の対象を扱う作業が大きな割合を占めるため，これを無視することはできない．

また，システムの大規模化に伴い，ファイルよりも細かい単位の構成管理，ソフトウェアの部品化・再利用などをリポジトリを用いて行ないたいという要求もあるが，従来のリポジトリではこれらの要求に答えることはできない．以下では，モジュールよりも粒度の細かい成果物を対象とするリポジトリを細粒度リポジトリと呼ぶ．

ソフトウェアの生産性を向上させるためには，特定のフェーズのみを支援するだけでは十分でないことも経験的に明らかである．すなわち，要求分析から保守・運用にいたるすべてのフェーズが統合されなければ，十分な効率化は達成できない．しかし，現状では個々のツールが独立して存在するのみで，設計以降のフェーズを支援する各種のツールが共有できる細粒度リポジトリは存在しない．保守・運用から上流に向かって遡るリストラクチャリングやリエンジニアリングを効果的に行なうことを考えると，細粒度リポジトリの重要性はさらに増大する．

一方，各種のCASEツールや研究のための試作システムにおいて，対象ソフトウェアの解析器を個別に持たなければならないため，解析器の作成に多くの労力が必要なことは大きな問題である．解析器の作成はCASEツールの作成に不可欠ではあるが，解析器の作成自体は本質的ではないことが多い．様々なCASEツールに共通して用いられる機能を提供するCASEツール・プラットフォームが存在すれば，解析器をCASEツールから分離することが可能となり，ツールの開発者はそのツールに本質的な機能の実現に専念できる．

本稿では，細粒度のソフトウェア・リポジトリに基づいたCASEツール・プラットフォーム Sapid (Sophisticated Application Programming Interface for software Development) を提案し，Sapidを用いることにより上述の問題点を解決できることを示す．以下では，最初にSapidの概要を示し，次にSapidを構成するサブシステムについて説明し，最後にemacs上で動作するソフトウェア操作エディタ，仕様書管理支援ツール，program slicingツールなどのSapidを用いて作成されたCASEツールについて解説する．

2 Sapidの概要

Sapidは，ソフトウェアデータベース (SDB: Software Database)[4]，アクセスルーチン (AR: Access Routines)，ソフトウェア操作言語 (SML: Software Manipulating Language)[9] から構成される．システムの構成を図1に示す．

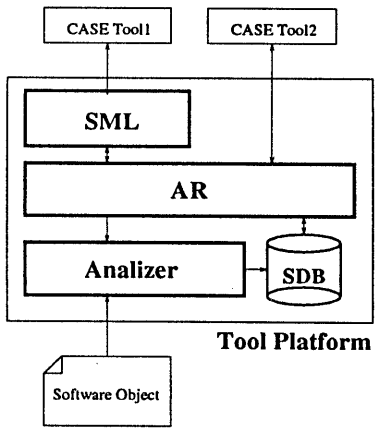

図 1: ツールプラットフォームの構成

SDBはSapidの基盤であり，要求されたソフトウェアをソフトウェアモデルに基づいて解析する解析器と解析の結果得られた実体・関連情報を格納するデータベースからなる．

ARはSDBに対するアクセス機能を提供す

るとともに，ソフトウェア開発において多用される前処理に対応するするためのPIDB(PIDB: Preprocess Information Database)[5]を含んでいる．SMLはソフトウェアに対する各種の操作を簡潔で直観的にわかりやすく記述するための言語である．クロスリファレンサなどの比較的単純なCASEツールはARを用いて実現し，複雑なCASEツールはSMLを用いて実現する．

SDBとARはC言語によって実現されていて，SDBは約7300行，ARは約2300行，PIDBは約18000行(ただし，約10000行はgccから流用)である．SMLはオブジェクト指向言語Sather[6]へのトランスレータとして実現されている．トランスレータはC言語で約2300行であり，SMLのためのSatherのクラスライブラリは約3500行である．Sapidは現在，Sun, MIPSなどの計算機で稼働している．また，複数の開発現場においてβテストを行なっている．

類似したシステムとしては，CIA[8]があげられる．CIAはCソースプログラムから，ファイル，関数，外部変数，型，マクロに関する情報を抽出して関係データベースに蓄積し，ユーザは問い合わせ言語を用いてソースプログラムの内容に関する照会を行なう．CIAはクロスリファレンサに代表されるソースプログラムからの情報抽出のみを行ない，ソースプログラムの変更を行なう機能はないため，我々の目指すツール・プラットフォームとしては不十分である．

3 SDB

SDBはソフトウェアを解析し，その構造や様々な関連を格納する．我々はソフトウェアをproject, program, file, function, declaration, block, label, expression, funccall (関数呼び出し), const (定数), member (構造体のメンバ), variable, type の13の実体，および構成関係や定義参照関係などの34の関連からなるERモデルとしてとらえた．このモデルは構文規則に基づいて作られたが，ソフトウェア開発者の立場から必要となる実体や関連を加えたり，逆に重要ではないものは削除することで，実体と関連を絞り込み，モデルを簡潔で扱い易いものにした．なお，ソフトウェアの構成要素には，仕様書，構成管理書類などの各種ドキュメントがあるが，現段階ではソースプログラムに重点を置いているため，ソースプログラム以外の実体と関連は省略している．また，本稿ではC言語を対象としているが，SDBの基本的なアイデアは他の言語にも適用可能である．

関数，変数，型，定数，構造体のメンバは実体としてモデル化されているため，名前の置き換えや参照情報の抽出は容易である．例えば，有効範囲や名前空間に基づいて，大域変数，局所変数，構造体のメンバなどを整合性を保ちつつ置換することができる．

4 AR

SDBのデータにアクセスするための機構としてアクセスルーチンがC言語のライブラリとして用意されている．その中で定義されているルーチンはユーザにとって必要かつ基本的なものである．基本アクセスメソッドとして，データベーススキーマを得るためのメタデータ取得メソッド，データ属性値取得メソッド，関連取得メソッド，関連オブジェクト取得メソッドがある．メタデータは本稿ではデータベース内のデータを構成する実体名，関連名および実体，関連の持つ属性名，属性の型を表す．

アプリケーションに提供するアクセスメソッドは基本アクセスメソッドを組み合わせてより複雑なアクセスを可能にする．アクセスメソッドを用いると，

- 関数`foo()`内で使われている大域変数を挙げよ
- 関数`printf()`の呼び出しで文字列定数`"Hello"`を引数に持つものを挙げよ

といった質問が可能となる．

また，ARには，ソフトウェアの開発で多く用いられる前処理(Preprocess)に対処するために，前処理情報データベースPIDBが組

み込まれている．前処理系が持つマクロの展開や条件付コンパイルなどの機能は，ソフトウェアの柔軟な開発・保守・管理に有用である．しかし，ソースプログラムに含まれる前処理の対処となる記述はその言語の構文規則にそわない場合もあり，解析の障害になることが多い．SDB に前処理に関する情報をもたせると，ソフトウェアモデルが複雑になるため，SDB には前処理に関する情報を入れず，前処理後のソースプログラムに関する情報のみを格納している．その代わり，前処理を施す前と後のソースプログラムの間の対応を PIDB に格納し，ユーザからのアクセス要求は PIDB に基づいて前処理後のソースプログラムに対するアクセス要求に変換される．また，SDB を検索して得られた情報は PIDB により前処理前の情報に変換されユーザに返される．

AR により，ユーザは基本的な変更操作や情報取得を行なうことができる．しかし，最小限の機能であるため，大規模な操作や複雑な操作には向かない．そのような操作は次章で述べるソフトウェア操作言語を用いることで容易に行なうことができる．

5 SML

ソフトウェアの操作を行なうためには，操作の記述が容易にできるような言語が必要である．特に，操作が大規模あるいは複雑で，さらにその操作を繰り返し行なうためには欠くことができない．そこで，操作を簡潔で直観的にわかりやすく記述できる言語としてソフトウェア操作言語 SML を提案し，その処理系を作成した[9]．SML ではソフトウェアを書式を持ったドキュメントの複合体として捉える．ドキュメントは書式に従って生成・変更する．この書式とドキュメントの関係はオブジェクト指向方法論におけるクラスとオブジェクトの関係に酷似しており，書式とドキュメントをそれぞれクラスとオブジェクトとして扱う．

例として，関数呼び出しを定数に置き換える操作を示す．ここでは，引数が文字列定数であるような C 言語のライブラリ関数 strlen() を，その文字列定数の具体的な長さに置き換える．図 2に置換前のソースプログラムを，図 3に置換後のソースプログラムを示す．ソースプログラム中に複数の strlen() の呼び出しが存在するが，引数が文字列定数"Hello "である strlen() のみが文字列の長さ 6 に変換されている．この置換操作のソースプログラムを図 4に示す．

```
 1 /*
 2  *  hello.c: sample program for changing strlen("Hello") to 6.
 3  */
 4
 5 #include <stdio.h>
 6
 7 void    main(int argc, char **argv)
 8 {
 9     char        *buf;
10
11     if (argc < 2) {
12         fputs("Usage: hello <name>\n", stderr);
13         exit(1);
14     }
15
16     /* Following strlen("Hello ") will be changed... */
17     buf = (char *)malloc(strlen("Hello ") + strlen(*++argv) + 1);
18     strcpy(buf, "Hello ");
19     strcat(buf, *argv);
20
21     fputs(buf, stdout);
22 }
```

図 2: 置換前

```
 1 /*
 2  *  hello.c: sample program for changing strlen("Hello") to 6.
 3  */
 4
 5 #include <stdio.h>
 6
 7 void    main(int argc, char **argv)
 8 {
 9     char        *buf;
10
11     if (argc < 2) {
12         fputs("Usage: hello <name>\n", stderr);
13         exit(1);
14     }
15
16     /* Following strlen("Hello ") will be changed... */
17     buf = (char *)malloc(6 + strlen(*++argv) + 1);
18     strcpy(buf, "Hello ");
19     strcat(buf, *argv);
20
21     fputs(buf, stdout);
22 }
23
```

図 3: 置換後

6 応用

Sapid を利用した応用プログラムの例として，ソフトウェア操作エディタ，仕様書管理支援システム，program slicing ツールを示す．

6.1 ソフトウェア操作エディタ

ソースプログラム内の識別子を置き換える場合に，単なる文字列の置き換えでは不十分なことは明らかである．一方，エディタに構文に関する知識を持たせた構文エディタが提案されているが，構文規則に拘束された操作し

```
class STRLEN is
  main(args: ARRAY{STR}) is
    if (args.asize < 2) then
      ERR::s("no target*").nl;
      return;
    end;
    filename: STR := args[1];
    if (args[1].is_equal("-e")) then
      SDB::analyze := true;
      filename := args[2];
    end;

    file: SOURCE_FILE := SOURCE_FILE::target(filename);

    foreach f: FUNCTION_CALL in {f: FUNCTION_CALL in file | is_a_target(f)} do
      arg: EXPRESSION := f.get_arguments.get_an_element;  -- an argument
      const: CONSTANT := arg.get_a_CONSTANT;
      f.set_text("".i(const.get_text.length - 2));  -- changing text
    end;  -- foreach

    OUT::s(file.get_text).nl;
  end;  -- main

  is_a_target(f: FUNCTION_CALL): BOOL is
    -- f must be "strlen()".
    if (not f.get_function.get_name.str.is_equal("strlen")) then return; end;

    -- Number of argument must be one.
    args: AGGREGATION{EXPRESSION} := f.get_arguments;
    if (args.size /= 1) then return; end;

    -- The argument must be a constant.
    arg: EXPRESSION := args.get_an_element;
    if (not arg.is_a_CONSTANT) then return; end;

    -- The constant must be a string.
    const: CONSTANT := arg.get_a_CONSTANT;
    res := const.is_string;
  end;  -- is_a_target
end;  -- STRLEN
```

図 4: 置換ソースプログラム例

か行なえないため，あまり普及していない．我々は，自由度を制限せずに，必要な時にのみ構文を理解して置換を行なうことを目的としてEmacs上にSDBとARを用いた識別子置換コマンドを作成した[1]．

このコマンドでは，Emacs上で任意の識別子の名前を指定し，新しい名前を入力すると，SDBの情報を元にして置換が必要な識別子をすべて置き換える．このとき，有効範囲および名前空間が異なる識別子は置き換えず，また置き換え後に名前が重複する場合にはユーザに確認を行なうなど整合性を保つように動作する．

プログラムは，マクロの処理に関する部分も含めてemacs lispで約600行である．また，ARのみを用いて実現されている．

6.2 仕様書管理支援システム

一般に，仕様書やソースプログラムなどの各種ドキュメントは常に整合性が保たれなければならない．しかし，整合性を保つ作業には非常に大きな労力がかかるため，これの自動化，あるいは整合性チェックの自動化が望まれている．仕様管理支援システムは，あらかじめ整合性の保たれた関数仕様書(以下，単に仕様書)とソースプログラムがSDBに格納されているときに，仕様書が変更されるとソースプログラムの該当する箇所を変更する．また，逆にソースプログラムが変更されると仕様書の該当する欄を変更する．仕様書とソースプログラムの変更にはどのようなツールを用いても良い．

なお，このシステムでは仕様書がない既存のソースプログラムから整合性のとれた仕様書の雛型を作成することもできる．この生成された仕様書に必要な情報を書き込むことで，仕様書を完成させることができる．

関数仕様書は，関数名とその型，関数の引数名とその型，関数内で用いられている変数名とその型，呼び出し関数名とその型，作成者氏名，作成年月日，最終更新年月日の7項目からなる．また，関数仕様書の記述には我々が作成したLaTeXのスタイルファイルを用いる．このシステムは，約200行のSMLプログラムと60行のC言語プログラムからなる．

6.3 Program slicing ツール

ソフトウェアの再利用においては，実行効率や可読性の向上のため，既存のプログラムから必要な部分のみを抽出する作業が頻繁に行なわれるが，作業の成否は作業者個人の能力に依存するところが大きい．Weiserによって提案されたprogram slicingは，変数の定義参照の依存関係を利用して，プログラム中のある文に関係があるすべての文を抽出する技術であり，これを用いて上述の作業を支援することが望まれている．program slicingは，プログラムのデバッグやメインテナンスに応用されているが，文の依存関係が得られることからプログラムのカスタマイズに用いることも検討されている[3]．

我々はスライシングがプログラムのカスタマイズにどの程度利用できるかを確認するために，制限したC言語(ポインタを含む)を対象にして，スライシングツールを試作した[3]．試作したツールは，制限したC言語(ポインタを含む)のソースプログラムのある文s中の変数vを指定して，ソースプログラム中のvの値に影響を与えるすべての文を抽出するツールである．このツールは約2000行のC言語の

プログラムである．その他の仕様を以下に示す．

- if-then-else 文，および while 文をもつプログラムから実行可能な必要部分を抽出する．switch-case 文, goto 文, for 文は扱わない．
- 高次ポインタ，関数内部におけるポインタ型引数の解析は行なわない．

Xwindow 上で動作するアプリケーションの一部をなす 100 行程度の関数を対象に評価を行なった．この関数は数値計算部分と計算された数値を表示する部分からなるが，試作したスライシングツールによって数値計算に関する部分のみを取り出す試行を行なったところ，人間が手動で抽出したものとほぼ同じ結果が得られた．現在，定量的な評価を行なっている．

7 おわりに

本稿では，ソフトウェアに対する変更や情報取得などの操作を行なうツールを容易に構築するツールプラットフォーム Sapid を提案した．Sapid は主に SDB, AR, SML から構成されており，SDB はソフトウェアに関する情報を保持するデータベースであり，AR は SDB にアクセスするためのルーチンである．AR には前処理に関する情報を保持した PIDB が組み込まれており，ユーザは前処理が行なわれる以前のソースファイルについて操作を行なうことができる．SML はソフトウェアに対する操作を柔軟にわかりやすく書くための言語である．本稿ではこれらのサブシステムについて説明した．

また，ツールプラットフォームの応用例としてソフトウェア操作エディタ，仕様書管理支援システム，program slicing ツールを示した．Sapid を用いることで CASE ツールのプログラムをアルゴリズムに忠実に見通しよく記述することができた．このことから，ツールプラットフォームとしての Sapid の有効性を確認することができた．ただし，定量的な評価は今後の課題として残されている．

今後の課題として，ソフトウェアモデルの拡張があげられる．現在はソースプログラムに重点を置いているが，仕様書や構成管理書類，テストケースに関する書類などソフトウェアに含まれる各種書類に対応できるようにすることが望まれている．構成管理ツールとして普及している Make に対するモデルを作成し，これをソフトウェアモデルと統合する試みを行なっている．

また，ソフトウェアモデルの表現に ER モデルを用いたが，必ずしもわかりやすいものではない．現在，オブジェクトモデルを用いた，より簡潔なモデルに関する研究を進めている．

参考文献

[1] 有賀寛朗, 山本晋一郎, 阿草清滋. ソフトウェア構造解析情報に基づくツールプラットホームシステム. 電子情報通信学会 ソフトウェアサイエンス研究会, Vol. SS94, No. 15, pp. 25–32, 1994.

[2] M. Chen and R.J. Norman. A Framework for Integrated CASE. *IEEE Software*, Vol. 9, No. 2, pp. 18–22, 1992.

[3] 橋本靖, 山本晋一郎, 阿草清滋. Program slicing を利用したプログラムカスタマイザ. 電子情報通信学会技術研究報告 SS94-10, pp. 73–80, 1994.

[4] 馬渕謙, 山本晋一郎, 酒井正彦, 阿草清滋. ソフトウェア保守におけるプログラム理解支援システム. 情報処理学会第 43 回全国大会講演論文集 (5), pp. 1J–1, 1991.

[5] 三村俊彦, 山本晋一郎, 阿草清滋. 前処理情報データベース. 平成 5 年度 電気関係学科東海支部連合大会講演論文集, pp. 584, 1993.

[6] Stephen M. Omohundro. The Sather Language. Technical report, Internal Computer Science Institute, 1991.

[7] I. Thomas and B. A. Nejmeh. Definition of Tool Integration for Environments. *IEEE Software*, Vol. 9, No. 2, pp. 29–35, 1992.

[8] Yih-Farn Chen, Michael Y. Nishimoto, and C. V. Ramamoorthy. The C Information Abstraction System. *IEEE Trans. Software Eng.*, Vol. 16, No. 3, pp. 325–334, 1990.

[9] 吉田敦, 山本晋一郎, 阿草清滋. ソフトウェア再利用のためのソフトウェア操作言語の提案. ソフトウェアシンポジウム'93 論文集, pp. 84–89, 1993.