# 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

福安直樹† 山本晋一郎† 阿草清滋†

CASEツールの開発には通常のリポジトリが扱わない細かい粒度の構成要素の管理を必要とする．我々はそのために，C言語のソースコードを12種類のクラスと29種類の関連としてモデル化した．また，このモデルに基づいたCASEツール・プラットフォーム **Sapid** を作成し，現在，複数の研究機関や開発現場でCASEツールの作成に用い評価をしている．**Sapid** の有効性を示すためにEmacs上で動作するソフトウェア操作エディタ，関数仕様書管理ツール，program slicingツール，依存解析ツールなどを作成した．これらの経験より，**Sapid** が提供するAPIを用いて各種の中流・下流CASEツールを見通し良く作成できることを確認した．

# CASE Tool Platform Sapid Based on a Fine Grained Repository

NAOKI FUKUYASU,† SHINICHIROU YAMAMOTO† and KIYOSHI AGUSA†

There is a need for developments of CASE tools to maintain fine grained components of softwares that are rarely treated by usual repositories. We have modeled for source codes of the C language by 12 classes and 29 relations. We have developmented the CASE tool platform, named Sapid, based on this model. We have made several CASE tools, such as an structured editor on Emacs, function specification maintenance tool, program slicing tool, dependency analysis tool, and so on, on Sapid in order to show its efficiency. These experiences let us confirm that Sapid APIs are effective to make CASE tools of middle or lower stream.

## 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われている[3),17)]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．たとえば，一般的なリポジトリ[2)]やプログラム理解支援ツール[20)]における管理の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，設計以降の局面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．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．

また，モジュールの内部を扱うリポジトリ[6)]も提案されているが，2.3節で示すように，ソフトウェアのライフサイクルのいろいろな局面で要求される多様な操作に十分対応しているとはいえない．

ソフトウェアの生産性を向上させるためには，特定の局面のみを支援するだけでは十分でないことは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局面が統一的に支援されなければ，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，設計以降の局面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．2000年問題に代表される大規模なレガシーシステムを効果的に保守する必要性の増大から，リストラクチャリング[4)]やリエンジニアリング[4)]技術が求められているため，ソフトウェアのライフサイクル全体の基礎となる細粒度リポジトリの確立は急務である．

各種のCASEツール，あるいはアイデアの検証を行うための実験的ツールの作成にあたって，対象ソフトウェアの字句解析，構文解析，意味解析を行うモジュールをツールごとに作成する必要がある．過去に字句解析器，構文解析器の作成支援に関する多くの研究が行

† 名古屋大学大学院工学研究科
School of Engineering, Nagoya University

われているが，LEX[7]やYACC[14]に代表される生成系は，字句解析器，構文解析器を記述するための枠組みであって，実際のCASEツールを作成する場合に，即座に利用できるモジュールは提供されていない．そのため，これらの成果を実世界のソフトウェアを対象としたCASEツールの作成に直接使うことはできない．しかも，これらのモジュールの大部分は，CASEツールに不可欠ではあるが本質的ではないことが多い．

本稿では，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**（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．**Sapid**の目標を以下に示す．

(1) 字句解析，構文解析などの，すでに技術的に確立されているがCASEツールにとって本質でない基礎的な機能を提供する．

(2) 波及解析や抽象実行に代表されるCASEツールに関する基盤技術の見通し良い実現を可能とする．また，基盤技術の実現をライブラリとして蓄積・流通・再利用するためのプラットフォームとなる．

(3) 新しいCASEツールを支える要素技術を確立するための実験基盤となる．

(4) 細粒度のソフトウェアモデルによるCASEツール間のデータ統合を実現する．

以下では，最初に2章で**Sapid**の概要を示し，3〜5章で**Sapid**を構成するサブシステムについて説明する．また，6章で，emacs上で動作するソフトウェア操作エディタ，仕様書管理ツール，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し，最後に評価と関連研究について述べる．

## 2. Sapidの概要

### 2.1 CASEツール・プラットフォーム

我々の考えるCASEツール・プラットフォームとCASEツールの関係を図**1**に示す．CASEツール・プラットフォームの中核は，ソフトウェアを構成するさまざまな粒度の要素を統一的に管理する細粒度リポジトリである．リポジトリはソフトウェアのライフサイクルで行われる多様な問合せや操作に対応できる必要がある．そのために，ソースコードに対する従来のリポジトリが用いるモデルよりも粒度の細かいモデルI-modelを用意した．図**2**にI-modelを示す．

モデルの粒度の設定には若干の恣意性が避けられず，定量的な議論は困難であるが，以下の点からI-modelは通常のプログラマにとって十分細かい粒度を持つといえる．I-modelに基づいた解析結果を用いてC言語の文を直接解釈実行する仮想機械を作成できることから，仮想機械の動作の系列をプログラムの意味とする操作的な意味論に基づいたプログラムの等価性を議論

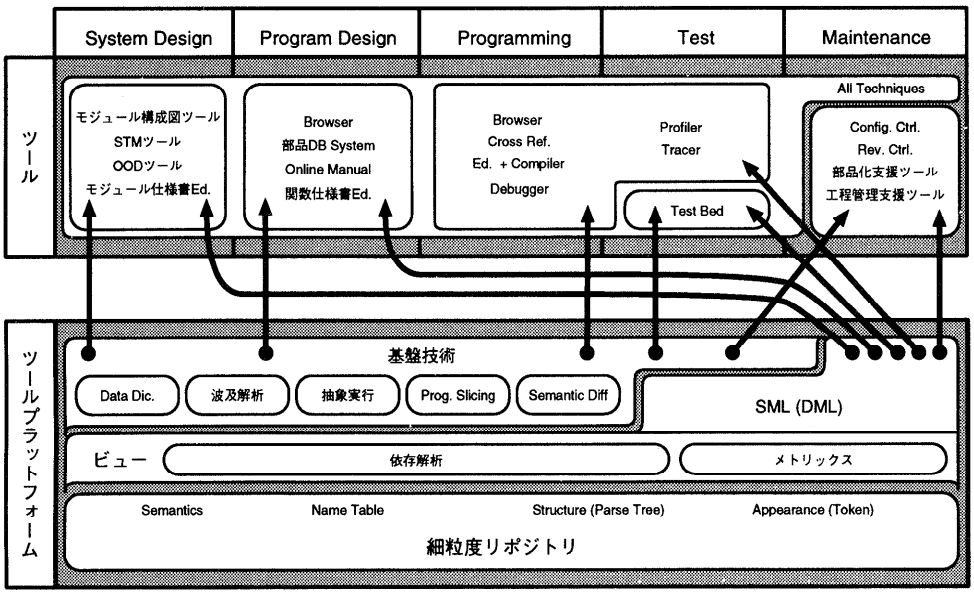



図1 ツール・プラットフォームの構成
Fig. 1 Composition of tool platform.

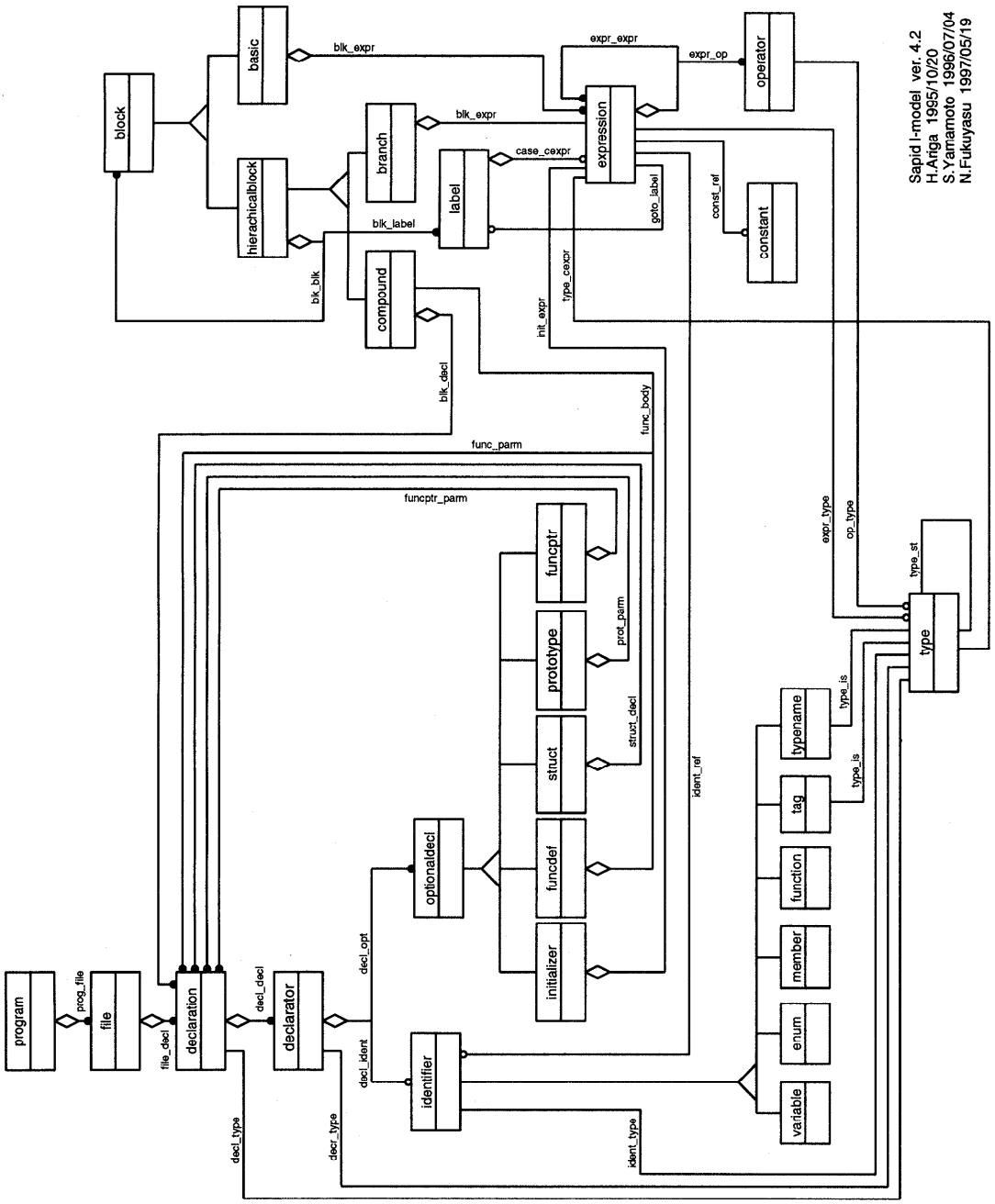



**図2** I-model
Fig. 2 I-model.

することが可能である．また，一般に細粒度のモデル上に適切なビューを設定して不要な情報を捨象することは容易であり，その逆を行うことはできないため，粒度の粗いモデルでは将来発生する可能性がある多様な要求に応えることができない．

一方，細粒度のリポジトリでは，管理するオブジェクトの数が増加し，性能が低下する．I-model の粒度は，上述のできるだけ細かい粒度という要請と，性能のトレードオフから設定した．I-model に関する説明は3章で行う．

特定分野の CASE ツールの立場から見ると，細粒度リポジトリは粒度が細かすぎることがあるので，適切なビューの役割は重要である．たとえば，program slicing [19] を行う CASE ツールは，ソースプログラム

をデータ依存グラフおよび制御依存グラフとしてとらえる．同様に，McCabeによるプログラム複雑度[9]を求めるCASEツールも，ソースプログラムを制御依存グラフとしてとらえる必要がある．このように，ビューはソフトウェアをある特定のCASEツールの見方でとらえるために用いられる．

**Sapid**には，I-model上にC言語で多用されるマクロに関するC-model，制御依存関係とデータ依存関係を表すビュー，C言語を直接解釈実行する仮想機械のビューなどが階層的に用意されているため，プログラムのライフサイクルにおける中・下流工程で行われる幅広い操作に対応できる．また，利用者が簡潔にビューを記述するための枠組みとしてソフトウェア操作言語[22]も用意されている．

CASEツール・プラットフォームの最上位層は，波及解析や抽象実行などの基盤技術ライブラリである．CASEツールはこれらのライブラリを適切に組み合わせて作成される．

なお，図1における各要素は必ずしも網羅的なものではない．また，本論文ではC言語を対象としているが，**Sapid**の基本的なアイデアは他の言語にも適用可能である．

### 2.2 Sapid

**Sapid**は，ソフトウェアデータベース（SDB: Software Database）[8),21)]，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）を主要なサブシステムとする．システムの構成を**図3**に示す．

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結果を格納するデータベースからなる．

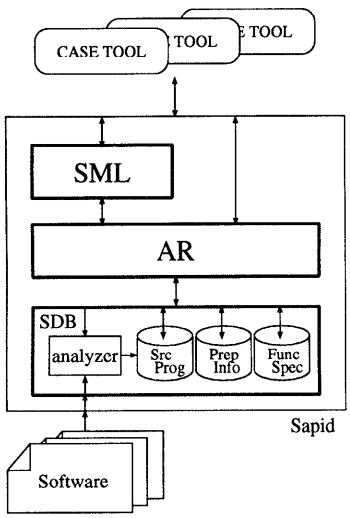



**図3** Sapidの構成
Fig. 3 Composition of Sapid.

ARはSDBに対するアクセス機能を提供するとともに，C言語を用いたプログラミングで多用される前処理に対応するためのPIDB（Preprocess Information Database）[10)]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分かりやすく記述するための言語である．SMLの詳細は文献22)で報告した．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．

**Sapid**は全体で約77,000行のC言語プログラムであり，そのうちSDBは約12,600行，ARは約6,400行，PIDBは約12,400行（ただし，約6,500行はgccから流用），SDAは約5,900行である．SMLはオブジェクト指向言語Sather[12)]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2,300行であり，SMLのためのSatherのクラスライブラリは約3,500行である．**Sapid**は現在，HP，Sun，SGIなどの計算機で稼働している．また，anonymous FTPやWWW[15)]上で一般に公開し，複数の開発現場においてCASEツールの作成に用いられている．

### 2.3 関連研究

以下の2つのシステムと**Sapid**との明確な違いは，両者ともリポジトリに基づいてソースプログラムを変更することを考えていない点にある．すなわち，これらのシステムでは，マクロやコメントを残した部品の切出しや，変数名やメンバ名の書換え，複数の変数を構造体に統合するような変更操作が困難である．一方，**Sapid**ではソフトウェアの操作に特化したソフトウェア操作言語SMLを提供して，ソフトウェアの開発・保守を行う際に，ソフトウェアに対して行われる様々な情報取得操作や変更操作をリポジトリに基づいて行うことができる．I-modelはそのために必要な情報を管理できるように設定した．

#### 2.3.1 COBOLアナライザ

COBOLアナライザ[6)]は，本研究と類似した細粒度の構成物管理を目指していて，商用システムの一部として用いることができる高い信頼性と性能を持っている．

**Sapid**は細粒度リポジトリにより，一般のコンパイラにおける構文木と記号表の情報を管理している．細粒度リポジトリに基づいてコンパイラのコード生成部を実現できることから，ターゲットマシンの動作の系列をプログラムの意味とする操作的意味論の立場に立てば，**Sapid**は十分に細かいソフトウェアのモデルを

提供している．文献6)では「構文木」としか触れられていないが，COBOLアナライザの基となるモデルの粒度もI-modelと同程度であると考えられる．また，COBOLアナライザもSDAに相当する制御依存関係とデータ依存関係を解析するAPIを提供している．

COBOLアナライザの主な目的は，対象プログラムの解析であり，変更必要箇所の発見を支援することに重点が置かれている．すなわち，ソフトウェア操作言語が目指す，リポジトリに基づいた変更操作は主要な目的ではない．

#### 2.3.2 Refine

Resoning Systems社のRefine Language Tools[13]（以下，Refine）は，GUIと利用者がカスタマイズ可能な強力な構文解析器を備えているリエンジニアリングのためのワークベンチとして有名である．モデルの粒度はRefineよりも**Sapid**の方が細かい．いうまでもなく，細粒度のモデルに基づく解析結果を抽象化することは可能であるが，その逆は不可能である．ただし，解析時間や解析結果の容量などの観点からの比較は行われていない．また，RefineにはSDAのような依存解析のための高度なビューは存在しない．

GUIについては，**Sapid**においてもGUIを持ったブラウザSAT（SDB Access Tool）の研究，実装を進めているが，Refineに及ばない．**Sapid**がカスタマイズ可能な構文解析器を提供する予定はないが，Javaへの対応は計画中である．

## 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program, file, declaration, declarator, identifier, optionaldecl, block, expression, label, constant, type, operatorの12のクラス，および構成関係や定義参照関係などを表す29の関連からなるモデルI-modelとしてとらえた．I-model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クラスや関連を加えたり，逆に重要ではないものは削除することで，モデルを単純で直観的なものにした．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，ソースプログラムに関係しないクラスと関連は省略している．図2にI-modelを示す．

関数，変数，型，定数，構造体のメンバはクラス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．たとえば，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

ARはSDBにアクセスするためのAPI関数群である．ARで用意されている関数は，CASEツール作成者にとって必要かつ基本的なものである．API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関数，オブジェクトや関連の属性値取得関数，オブジェクト取得関数，関連取得関数などが用意されている．メタデータとは，SDB内のデータを構成するクラス名，関連名，およびクラスと関連の持つ属性名，属性の型を表す．

**Sapid**を用いるCASEツールは，これらの関数を組み合わせて，より複雑な機能を作成する．ARを用いると，

- 構造体のメンバ`foo`に代入している箇所とその値を参照している箇所をあげよ
- 関数`printf()`の呼び出しで，第1引数が文字列定数`"Hello"`であるものをあげよ

などの操作を容易に実現することができる．これらの操作はいわゆるクロスリファレンサが提供する程度のサービスであるが，現在市販されているCASEツールで，上述の操作を真に実現しているツールは存在しない．

また，ARには，C言語を用いたソフトウェアの開発で多く用いられる前処理に対処するために，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象となる記述はその言語の構文規則に従わない場合もあり，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持たせると，I-modelが複雑になるため，SDBは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．

## 5. SDA

### 5.1 制御とデータの依存関係

SDAは，ソースプログラムの制御依存関係とデー

```
 1 void function(void)
 2 {
 3     int a;
 4
 5     a = 1;
 6     {
 7         int a;
 8         a = 22;
 9         printf("1: a = %d \n", a);
10     }
11     printf("2: a = %d \n", a);
12     if (condition()) {
13         printf("3: a = %d \n", a);
14         a = 333;
15     }
16     else {
17         a = 4444;
18     }
19     printf("4: a = %d \n", a);
20 }
```

図4 被解析プログラム
Fig. 4 Target program.

タ依存関係を扱うCASEツールを見通し良く実現するためのAPI関数群である．SDAを用いることにより，I-modelに高い抽象度のビューを設定することができる．

制御依存関係は，制御流れグラフ（CFG: Control Flow Graph）で表され，プログラムを構成するステートメントを節点とし，また2節点間の制御依存関係を，2つの節点を結ぶ辺とする．

データ依存関係は，データ流れグラフ（DFG: Data Flow Graph）で表され，変数の出現を節点とし，データ依存関係を辺とする．データ依存関係の定義はその目的によってさまざまに変わりうるが，定義-使用の関係など一般的な関係は，SDAの関数としてあらかじめ提供される．また，CASEツール作成者が独自の依存関係を手続き的に与えられる仕組みも用意されている．

また，CFGの節点とDFGの節点の対応をとる関数や，到達可能性の判定などグラフに対する一般的な操作も用意されている．現在のSDAは関数を単位とした解析を行う．関数に跨った解析とalias解析の機能の充実は今後の課題である．

### 5.2 CFG, DFGの例

図4に示したプログラムに対するCFG，DFGの例を図5に示す．図5の左側がCFGで，右側がDFGである．ただし，DFGの辺は定義-使用関係を表している．また，DFGの節点に対応する変数の出現は，左側のCFGの節点における変数の出現に対応している．

5行目の`a = 1`では，変数`a`が定義されている．この定義が使用されるのは，11，13行目の`printf(...)`文である．また，スコープの異なる`a`が8行目で定義され，9行目で使用される．さらに，14，17行目で定義された変数`a`が19行目で使用される．SDAを用いると，sdaMakeCFG()関数とsdaMakeDefUseDFG()関数の呼び出しにより簡単にこれらのグラフを作成することができる．

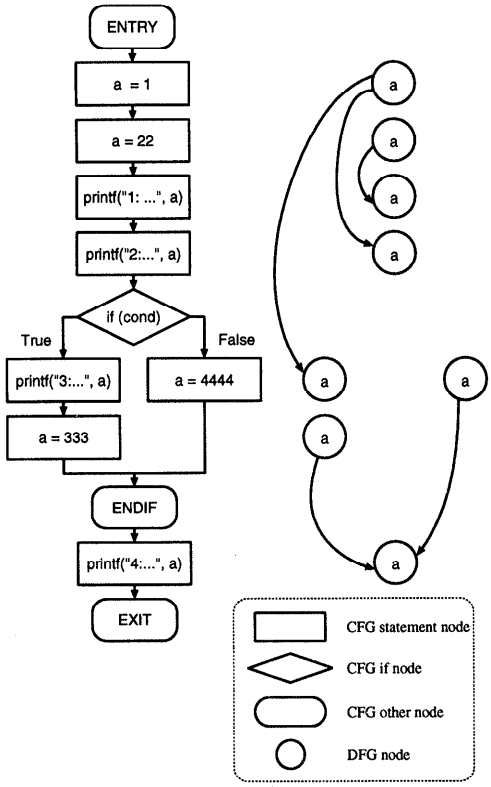



図5 CFG, DFGの例
Fig. 5 Example of CFG and DFG.

## 6. 応用と評価

**Sapid**を利用したCASEツールの例として，ソフトウェア操作エディタ，仕様書管理ツール，program slicingツールを示す．この他にも，我々の研究グループによる依存関係が定義可能なテストベッド[18]，依存関係に基づく差分抽出ツール[23]や，京都大学の小田ら[11]によるC言語を対象としたリバースエンジニアリング・ツールなどが**Sapid**を用いて作成されている．

### 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か行えないため，あまり普及していない．我々は，自由度を制限せずに，必要なときにのみ構文を理解して置換を行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の情報を基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．

### 6.2 仕様書管理ツール

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相互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．仕様管理ツールは，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通常のエディタを用いることができる．

なお，このツールでは仕様書が存在しない既存のソースプログラムから仕様書の雛型を作成することもできる．この生成された雛型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2,000行のC言語プログラムからなる．

### 6.3 Program slicing ツール

Weiserによって提案されたprogram slicing[19]は，変数の定義参照の依存関係を利用して，プログラム中の指定された文に影響を与えるすべての文を抽出する技術である．program slicingの応用として，プログラムのデバッグ，メインテナンス，プログラムの理解支援などが考えられている．

我々はprogram slicing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，制限されたC言語（ポインタを含む）を対象にして，program slicingツールを試作した[5]．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2,000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文を持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．
- 高次ポインタの解析，および関数間に跨ったポインタ型引数の解析は行わない．

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った．この関数は数値計算部と計算された数値の表示部からなるが，試作したprogram slicingツールによって数値計算部のみを取り出す試行を行ったところ，人間が手動で抽出したものとほぼ同じ結果が得られた．

### 6.4 評　　価

**時間的・空間的効率**

X11R6.3で配布されている端末エミュレータxtermは15個のC言語プログラムファイルからなり，合計行数は約21,000行である．このプログラムをI-modelに基づいて解析すると，103,000個のオブジェクトと187,000個の関連が生成される．そのための時間はDELL社のDIMENSION XPS D300，FreeBSD-2.2.2で121秒（ユーザタイムとシステムタイムの合計）である．ちなみに，このプログラムに前処理を施すと全体で約253,000行になり，I-modelはこの前処理後のソースプログラムに対して生成される．また，xtermを最適化しないでコンパイルするのにかかる時間は9秒である．

xtermを構成する344個の関数に対して，6.2節の仕様書管理ツールで関数仕様書の雛型を作成するのに68秒かかった．出力は約7,000行のLaTeXソースである．

これらの経験より，**Sapid**が実用的な時間的，空間的効率で動作することが分かる．

**CASEツールのプログラムサイズ**

ここでは，6.1節のソフトウェア操作エディタの置換コマンドと6.3節のprogram slicingツールを対象にして，**Sapid**を用いる場合と用いない場合のそれぞれについて，CASEツールのプログラムサイズを考察する．ただし，実際に**Sapid**を用いないでこれらのCASEツールを作成するのは非現実的であるので，これらのCASEツールが**Sapid**のどの部分を用いているかを分析し，その効果を以下のように推定する．

**表1**より，**Sapid**を用いない場合は，**Sapid**を用いる場合に比べて，2.5倍から3倍程度のプログラム

表1 CASEツールのプログラムサイズ（単位：行）
Table 1 Program size of CASE tools (lines).

| | Sapid使用 | Sapid未使用 |
|---|---|---|
| 置換コマンド | 800 | 2,400 |
| program slicingツール | 2,000 | 5,000 |

を作成する必要があることが分かる．これらのCASEツールは**Sapid**の比較的低レベルのAPIのみを使っているので，より高機能なAPIを用いたCASEツールでは，その差はさらに大きくなると考えられる．

また，2.3.2項のRefineを利用した場合の評価に関して，Refineを用いない場合は，Refineを用いる場合の約2.7倍程度のプログラムの記述が必要であることが文献16)において報告されている．

**評価のまとめ**

1章であげた**Sapid**の目標（1）「CASEツールの基礎的な機能の提供」は達成することができた．

目標（2）「CASEツール・プラットフォームの実現」は部分的に達成することができた．今後は，このプラットフォーム上にライフサイクルのより広い範囲を支援する高機能なCASEツールを作成し，枠組みを洗練するとともにその実用性を検証することが望まれる．

目標（3）「CASEツールの新しい要素技術」については，文献11)などの若干の成果が生み出されつつあるが，**Sapid**を用いた革新的な技術の誕生が待たれる．

目標（4）「**Sapid**を用いたデータ統合」は今後の課題として残されている．

## 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うツールを容易に構築するCASE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われる以前のソースファイルについて操作を行うことができる．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．

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，関数仕様書管理ツール，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良く記述することができた．これらの経験より，**Sapid**が提供するAPIを用いて各種の中流・下流CASEツールを見通し良く作成できることを，定性的かつ定量的に確認した．

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．現在の**Sapid**はソースプログラムと関数仕様書を対象としているが，各種仕様書や図，構成管理書類，テストケースに関する書類など，ソフトウェアを構成するより多くの書類を統一的に管理することが望まれている．また，現在の**Sapid**は解析結果を単なるUNIXのファイルとして管理している．そのため，排他制御，障害回復など一般的なDBMSが持つべき機能が実現されていない．PCTEなどの解放型のデータ管理機構を用いて，ソフトウェアデータベースのスキーマを記述することや，一般的なDBMSを用いることは今後の課題である．

## 参考文献

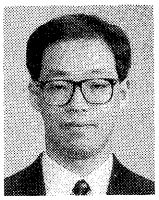

1) 有賀寛朗，山本晋一郎，阿草清滋：ソフトウェア構造解析情報に基づくツールプラットホームシステム，電子情報通信学会技術研究報告，Vol.SS94, No.15, pp.25–32 (1994).
2) Buxon, J.N.: *DoD Requirements for Ada Programming Support Environments*, United States Department of Defense (1980).
3) Chen, M. and Norman, R.: A Framework for Integrated CASE, *IEEE Software*, Vol.9, No.2, pp.18–22 (1992).
4) Karlsson, E. (Ed.): *SOFTWARE REUSE – A Holistic Approach*, John Wiley & Sons (1995).
5) 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，Vol.SS94, No.10, pp.73–80 (1994).
6) 川辺敬子，上原三八，金谷延幸：COBOLアナライザ：リバースエンジニアリング・プラットフォームの開発，電子情報通信学会知能ソフトウェア研究会，Vol.KBSE94, No.32, pp.1–8 (1994).
7) Lesk, M.E.: LEX – Lexical Analyzer Generator, Technical Report 39, Bell Telephone Laboratories (1975).
8) 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，第43回情報処理学会全国大会論文集（5），pp.1J–1 (1991).
9) McCabe, T.J.: A Complexity Measure, *IEEE Trans. Software Engineering*, Vol.2, No.4, pp.308–320 (1976).
10) 三村俊彦，山本晋一郎，阿草清滋：前処理情報データベース，電気関係学会東海支部連合大会講演論文集，p.584 (1993).
11) 小田章夫，鯵坂恒夫：Cプログラムに対するカプセル発見手法とその支援ツール，電子情報通

信学会論文誌，Vol.J79-D-I, No.10, pp.745–758 (1996).
12) Omohundro, S.M.: The Sather Language, Technical Report, Internal Computer Science Institute (1991).
13) Reasoning Systems: Refine User's Guide, Palo Alto, CA (1989).
14) Johnson, S.C.: YACC – Yet Another Compiler Compiler, Technical Report 32, Bell Telephone Laboratories (1975).
15) Sapid Home Page: http://www.sapid.org/.
16) 高田智規，佐藤慎一，飯田　元，井上克郎：ソースコード解析ツール開発支援システムの試用，電子情報通信学会論文誌，Vol.J80-D-I, No.3, pp.317–318 (1997).
17) Thomas, I. and Nejmeh, B.A.: Definition of Tool Integration for Environments, *IEEE Software*, Vol.9, No.2, pp.29–35 (1992).
18) 内山晃司，山本晋一郎，阿草清滋：依存関係が定義可能なテストベッド，情報処理学会ソフトウェア工学研究会，Vol.106, No.6, pp.41–47 (1995).
19) Weiser, M.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol.10, No.4, pp.352–357 (1984).
20) Chen, Y., Nishimoto, M.Y. and Ramamoorthy, C.V.: The C Information Abstraction System, *IEEE Trans. Softw. Eng.*, Vol.16, No.3, pp.325–334 (1990).
21) 山本晋一郎，阿草清滋：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，情報処理学会ソフトウェア工学研究会，Vol.102, No.7, pp.37–42 (1995).
22) 吉田　敦，山本晋一郎，阿草清滋：CASE ツール開発のためのソフトウェア操作言語，情報処理学会論文誌，Vol.36, No.10, pp.2433–2441 (1995).
23) 吉田　敦，山本晋一郎，阿草清滋：意味を考慮した差分抽出ツール，情報処理学会論文誌，Vol.38, No.6, pp.1163–1171 (1997).

(平成 9 年 10 月 20 日受付)
(平成 10 年 3 月 6 日採録)

**福安　直樹**

1973 年生．1996 年名古屋大学工学部情報工学科卒業．現在，同大学大学院工学研究科情報工学専攻博士前期課程に在学中．ソフトウェア開発環境に関する研究に興味を持つ．ソフトウェア科学会会員．



**山本晋一郎**（正会員）

1962 年生．1987 年名古屋大学工学部卒業後，同大学大学院に進学，1991 年同大学助手，1996 年講師．プログラミング言語処理系，ソフトウェアの形式的開発手法，ソフトウェア開発環境に関する研究に従事．最近は，細粒度のソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームに関する研究を進めている．電子情報通信学会，ソフトウェア科学会各会員．

**阿草　清滋**（正会員）

1947 年生．1970 年京都大学工学部電気工学第二学科卒業．1972 年同大学大学院工学研究科電気工学第二専攻修士課程修了．同博士課程へ進学．1974 年より同情報工学科助手．同講師，助教授を経て 1989 年より名古屋大学教授．現在，同大学大学院工学研究科情報工学専攻知的インタフェース学講座担当．工学博士．専門分野はソフトウェア工学．ソフトウェア開発方法論，知的開発環境，ソフトウェアデータベース，仕様化技法，再利用技法，マンマシンインタフェース．電子情報通信学会，ソフトウェア科学会，IEEE，ACM 各会員．