ソフトウェア工学 98－1
（1994. 5. 25）

プログラムスライシングによるソフトウェア部品の作成

丸山 勝久　　高橋 直久
{maru, naohisa}@slab.ntt.jp

NTT ソフトウェア研究所
〒180 東京都武蔵野市緑町 3-9-11

部品を用いてソースコードを再利用する環境においては，部品の抽出や理解を容易にする手法が望まれている．これに対して，本稿では順方向及び逆方向区間限定スライスを定義し，既存のソースコードにプログラムスライシングを適用することで，再利用可能なソフトウェア部品を作成する手法を提案する．本手法では，スライシング技法をプログラムから部品本体を抽出するため，及び部品理解や変更を容易にする付加情報を提示するために用いる．付加情報は，部品を実行するために必要な実行前提条件例と部品がどのように利用されているかを示す利用条件例を合わせたもので，ソフトウェア部品として部品本体と同時に生成する．また，本手法によって作成された部品本体と付加情報はそれぞれ実行可能であり，必要最低限の記述コードからなる．さらに，本稿では本手法に基づき実現した部品作成システムを紹介し，その適用例を示す．

Extracting Reusable Software Components through Program Slicing

Katsuhisa MARUYAMA and Naohisa TAKAHASHI

NTT Software Laboratories
3-9-11 Midori-cho Musashino-shi Tokyo 180, Japan

This paper introduces two kinds of program slices, called *forward and backward bounded slices*, and presents a method for extracting reusable software components that are defined by these slices. The bounded slices are used to extract both the body of the software components and additional description from existing source code. This description consists of execution conditions and instances of utilization, which are the codes needed to execute a component and examples of how to use it, respectively. The codes facilitate understanding components. Any software component can be extracted from arbitrary bounds in a source code by means of this method, all extracted components are executable and do not include superfluous code. An experimental software reuse system implemented based on this method is presented.

1 はじめに

ソフトウェアの生産性や保守性を上げるために，従来からソフトウェアの再利用が考えられている．再利用の対象としては，プログラムのソースコード，設計，仕様，ドキュメント，プロセスなどがあり，それぞれの対象に応じて再利用の手法が提案されている．特に，プログラムのソースコードを再利用対象とする際には，まずプログラム部品を作成し，それらの部品をライブラリとして管理する方法がある．この方法では，あらかじめ用意されたプログラム部品を検索，変更，合成することで目的のプログラムを作成する[1]．このような部品化によるプログラム開発においては，再利用可能なソースコードの一部を重複して作成するという手間が省略され，開発コストを減少させることができる．また，部品を作成する際に，各部品に対する入出力を厳密に定義しておくことで，部品で構成されたプログラムは機能拡張や削除が部品単位で適用され，保守性が上がる．しかし，部品化による再利用に対しては，大きく分けて次に示すような2つの問題が存在する．

- 再利用性を上げるために，どのようなプログラム部品(汎用性，サイズ)を用意すればよいのかが確定していない．また，部品の再利用性は再利用を行なう領域に依存するため，部品作成時には未知であることが多い．(部品作成に関する問題)
- 目的のプログラムを作成するために，どの部品を再利用するのがよいのかを判断することが難しい．また，実際に部品を組み合わせる作業が複雑で，組み合わせてできたプログラムが正しいかどうかわからない．(部品検索，変更，合成に関する問題)

特に，部品化による再利用を推し進めていくと，再利用時の部品理解が必ず要求され，従来のプログラム部品だけでは再利用時の各過程を支援するのに不十分である．また，部品作成に関しても，現段階では明確な基準がなく，部品作成者の能力によって部品の質は左右される．さらに，部品の作成過程はその大部分を手動で行なっているのが現状である．

以上より，部品化再利用によるプログラム開発においては，部品をどのように作成するのかという部品抽出と，部品の検索，変更，合成の際に要求される部品理解を容易にする手法が望まれている．そこで，上で述べた部品化再利用の問題に対して部品抽出と理解に着目すると，以下に示す3つの問題が導出できる．

- 部品を抽出，作成する手法が確立していないため，その手続きが自動化されていない．また，部品がその機能や用途に明確な基準を持たないので，再利用時の部品選択及び修正が難しい．
- 部品が実行に必要な条件をすべて備えている保証はなく，実行により動作を確認したいという要求に応じられないことが多い．また，実行のための条件を部品ソースコードのみから捜し出すのは難しい．
- 部品の機能を理解する際，その部品がどのように利用できるのかを知りたいという要求がある．一般に部品とは再利用の対象そのものであり，再利用される環境や条件を含んでいない．よって，部品ソースコードだけからその利用方法を取得したり，推測したりすることが難しい．

これらの問題に対して，プログラムスライシング[2]を用いたソフトウェア部品の作成手法を提案する．ソフトウェア部品とは，再利用ソースコードであるプログラム部品と部品理解に役立つ付加情報を合わせたものである．スライシング技法は作成中のプログラムから部品を抽出するため，及び付加情報を提示するために用いる．本稿では，このようにスライシング技法を用いることで，次に示す利点をもつ部品作成手法が実現可能であることを示す．

(1) 指定された変数の計算に対する必要最低限の記述コードを，ソースコードから部品として自動抽出できる．

(2) 動作確認可能な部品とその利用例を提供し，多様で柔軟なテストを可能にする．

(3) 部品の包含関係と等価関係によりライブラリの階層化とグループ化を実現し，部品変更や合成の際に必要な類似部品の検索を容易にする．

(4) 部品作成のための作業量が少なく，部品を利用しながら部品を作成するという再利用指向ソフトウェア開発環境の構築に適している．

本稿の構成は次の通りである．まず，2章ではスライシングによるソフトウェア部品の具体的な作成手法と部品抽出に関する利点(1)について述べる．次に，3, 4, 5章で，残りの3つの利点(2)~(4)に関して考察する．さらに，6章では本提案手法に基づいてワークステーション上に実現したソフトウェア部品作成システムを紹介する．

2 ソフトウェア部品の作成手法

2.1 プログラムスライシング

現在開発や変更中のプログラムからソフトウェア部品を作成するために，プログラムスライシングを用いる．スライシングとは，着目する変数に影響を与える一部のコードだけをもとのソースコードから抜き出すことであり，対象となるプログラムのデータ依存(data dependence)と制御依存(control dependence)解析[3]によって達成される．本手法においては，スライシングを部品作成の

手段として用いるため，静的で実行可能なスライスを作成する．また，プログラム制御の流れる向きを順方向，その逆向きを逆方向と呼び，部品に対する付加情報を提示するために，順方向と逆方向の 2 種類のスライスを定義する．各方向に対するスライスを以下に示す．

定義 1: 逆方向スライスと順方向スライス

$S_b(n,v)$: 逆方向スライス (文献 [4] の手法で作成)
(*Static Backward Executable Slice*)

$S_f(n,v)$: 順方向スライス
(*Static Forward Executable Slice*)

$$\stackrel{\text{def}}{=} \bigcup_{i \in Dom(n,v)} S_b(i, \{Def(i)\})$$

部品作成対象の手続き型プログラムは CFG (Control Flow Graph)[5] で表現する．n は CFG で着目する節点，v は着目する変数の集合である．また，$Dom(n,v)$ は節点 n で変数 v が定義されていると仮定して，データ依存と制御依存を順方向にたどるとき通過する節点の集合，$Def(i)$ は節点 i で定義される変数の集合である．これらのスライスは，それ自身単体で実行可能であり，すべての入力において節点 n の変数 v に関して実行結果がもとのプログラムと等価である．$S_b(n,v)$ と $S_f(n,v)$ を図 1 に示す．

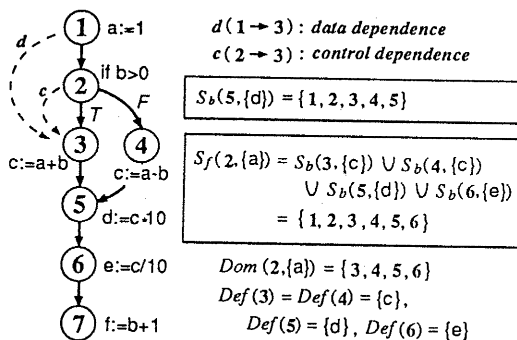

図 1: 逆方向スライスと順方向スライス

さらに，本手法ではプログラム中の任意の区間から部品を抽出できるようにするため，それに対応するスライスを用意する必要がある．そこで，CFG の任意の区間を上限節点 n_u と下限節点 n_l により $Bounds(n_u, n_l)$ と表現し，節点 n_u から節点 n_l に到達可能な経路上にある節点の集合を以下のように定義する．

定義 2: 到達可能経路 (RP : *Reachable Path*)

$$RP(n_u, n_l) \stackrel{\text{def}}{=} RP_f \cup RP_b$$

RP_f : 上限節点 n_u から下限節点 n_l に上限節点 n_u を中間節点として通らず順方向に経路をたどるとき通過する節点の集合

RP_b : 下限節点 n_l から上限節点 n_u に下限節点 n_l を中間節点として通らず逆方向に経路をたどるとき通過する節点の集合

このように RP を定義することにより，上限節点 n_u や下限節点 n_l が条件文や繰り返し文の内部にあるのか外部にあるのかを意識せずに指定することが可能で，この場合にも PR は一意に決定できる．逐次実行，条件分岐，繰り返しに対する RP を図 2 に示す．

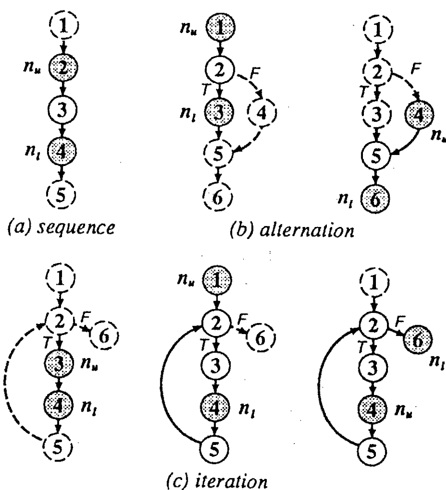

図 2: 到達可能経路 (RP) の例

定義 1 と定義 2 により，実際にソフトウェア部品を作成する際に利用するスライスを区間限定スライスと呼び，以下のように定義する．

定義 3: 区間限定スライス (*Bounded Slices*)

$\hat{S}_b(n_u, n_l, v)$: 逆方向区間限定スライス
(*Backward Bounded Slice*)

$$\stackrel{\text{def}}{=} S_b(n_l, v) \cap RP(n_u, n_l)$$

$\hat{S}_f(n_u, n_l, v)$: 順方向区間限定スライス
(*Forward Bounded Slice*)

$$\stackrel{\text{def}}{=} \bigcup_{i \in Dom(n_u,v) \cap RP(n_u,n_l)} S_b(i, \{Def(i)\})$$

CFG から $\hat{S}_b(n_u, n_l, v)$ と $\hat{S}_f(n_u, n_l, v)$ を作成する様子を図 3 に示す．

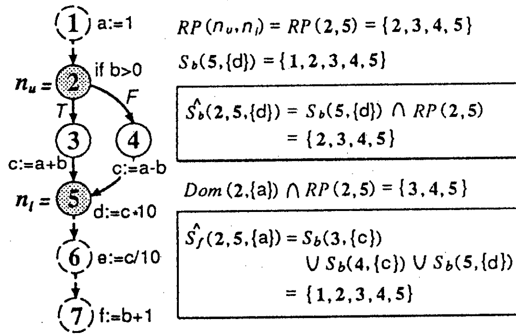

図 3: 区間限定スライス

2.2 スライシングによる部品作成

プログラム部品を再利用する場合，その部品の機能や使い方を理解したいという要求は多い．これらの要求に対して，再利用部品の本体コードだけを提供するのでは支援として不十分である．特に，部品を実行させてその動作を確認したい場合，部品単体では実行不可能なことが多く，またその実行結果がどのように利用できるのかを判断するのは困難である．そこで，本手法では再利用対象の部品本体を作成するのと同時に，その部品に対する付加情報も生成する．部品本体は，ソフトウェア開発者がプログラム作成時に区間 $Bounds\,(n_u, n_l)$ と着目する変数 v を指定し，スライシングによって作成する．また，付加情報は部品本体に対する一種の入出力例であり，これもスライシングによって生成可能である．本手法の概要を図 4 に示す．

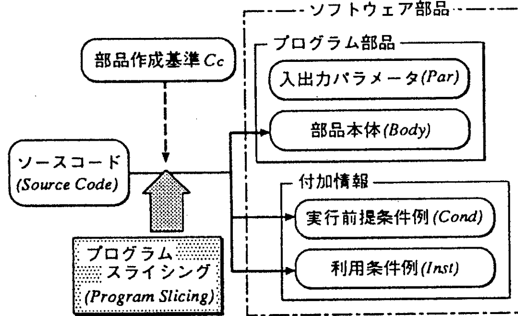

図 4: 本手法の概要

次に，部品本体に対する入出力パラメータを決定するために，入力データ依存と出力データ依存を考える．入力データ依存は部品本体外部の節点から内部の節点へのデータ依存で，依存元節点の定義側変数集合 V_{entry} が入力パラメータ In である．出力データ依存は部品本体内部の節点から外部の節点へのデータ依存であり，依存元節点の定義側変数集合を V_{exit} とする．出力パラメータ Out としては，部品作成時に着目する変数 v，部品本体内部で定義される変数集合 V_d，同一区間でスライスを作成したとき値が一意に定まる変数集合 V_s，部品外部で利用される変数集合 V_{exit} を提示し，部品登録時に作成者が選択する．変数集合 V_d は部品内で値が定まることだけしか保証されていないので，もとのプログラムと部品とで変数の意味が変わることがある．これに対して，変数集合 V_s はスライス作成前後で値が一意であることが保証されるので，変数の意味が変わることはない．また，変数集合 V_{exit} は部品外部で利用されているため，一時変数である可能性は低く，出力パラメータに変数を残しておくかどうかの指針となる．

以上より，部品作成基準 $C_C = <n_u, n_l, v>$ におけるソフトウェア部品を以下のように定義する．

定義 4: ソフトウェア部品 (C_S)

$\hat{S}_b(n_u, n_l, v)$ ： 本体 ($Body$)

$(In, Out) = (V_{entry}, v\,;\,V_d\,;\,V_s\,;\,V_{exit})$

$V_s = \{w \mid w \in V_d \wedge \hat{S}_b(n_u, n_l, w) \subseteq Body\}$

： 入出力パラメータ (Par)

$\hat{S}_b(first, n_l, v) - Body$ ： 実行前提条件例 ($Cond$)

$\hat{S}_f(n_s, n_r, v)$ ： 利用条件例 ($Inst$)

$n_s \in succ(Body),\ n_r \in Ref(n_s, v)$

ただし，$succ(Body)$ は CFG において $Body$ 内の節点から他の節点を通らずに到達可能な節点の集合であり，節点 n_s と最終節点の間で変数 v を参照している節点の集合を $Ref(n_s, v)$ と表す．また，$A\,;B\,;C$ は A, B, C のうちどれか一つを選択することを意味する．ここで，$Cond$ と $Inst$ を合わせて付加情報と呼ぶ．図 5 に示すプログラムに対して，ソフトウェア部品を作成すると図 6 のようになる．本手法におけるソフトウェア部品本体は，部品作成者が指定した範囲内に必ず含まれる．つまり，$Body \subseteq RP(n_u, n_l)$ である．また，スライスの作成に関しては有限時間で計算可能であるため，区間と変数の指定後は自動で部品作成が可能である．

このようにプログラムスライシングを用いて部品を作成する手法は，スライスの性質により以下の特徴をもつ．

(a) プログラム作成者の意志でプログラム中の任意の区間から部品が抽出可能である．つまり，部品抽出の際に抽出対象のプログラム構造から制約を受けない．

(b) 変更により構造が煩雑になったプログラムからでも指定した変数に対応する部品が抽出可能で，それらに対して不必要な記述が削除される．

(c) 部品作成基準変数の意味を把握していれば，抽出対象のプログラムの内部コードにまで関わらなくても部品が作成できる．また，特定の変数に着目して部品を作成するので，その変数の意味を記述するだけ

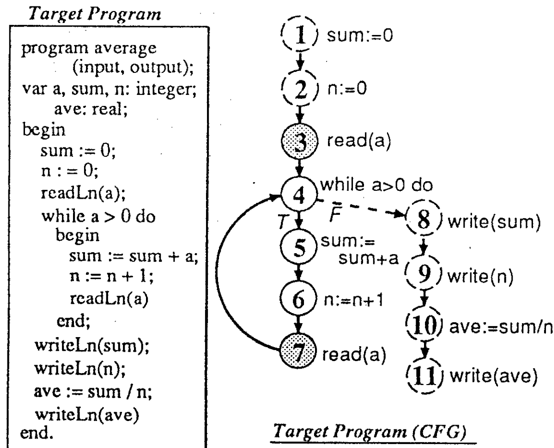

図 5: 部品作成の対象プログラム

Reusable Software Component

$Body : \hat{S_b}(3,7,\{sum\}) = \{3,4,5,7\}$

$Par : (\{sum\}, \{sum\}; \{sum, a\}; \{sum, a\}; \{sum\})$

$Cond : \hat{S_b}(1,7,\{sum\}) - Body = \{1\}$

$Inst : \hat{S_f}(8,8,\{sum\}) = \{1,3,4,5,7,8\}$

$\hat{S_f}(8,10,\{sum\}) = \{1,2,3,4,5,6,7,8,10\}$

$(n_u = 3, n_l = 7, v = \{sum\})$

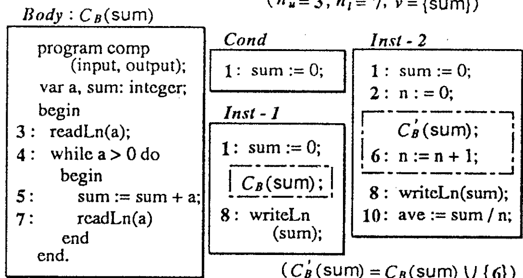

図 6: ソフトウェア部品の例

で部品の機能に対するドキュメントを作成可能である.

(d) 部品の実行や理解を支援する必要最低限のコードだけを，付加情報として提示可能である.

(e) スライス間の包含関係と等価関係を利用して，部品の階層化とグループ化が可能である．これにより，部品検索過程でより多くの関連部品を提示することができ，部品の再利用性が高くなる.

3 ソフトウェア部品における付加情報

本章では，定義4の実行前提条件例と利用条件例について考察を加える.

3.1 実行前提条件例

部品を再利用してプログラムを作成する場合，ライブラリ中の部品を実行させ，その動作を確認したいという要求がある．このような要求に応じるために，本手法では部品本体に実行前提条件例を付加する．実行前提条件例とは，もとのプログラムにおいて部品を実行する前に設定されていなければならな変数，つまり部品にとっての入力変数の値を定義するコードである．このコードは部品を実行するのに必要な条件の一例であり，部品と組み合わせることで部品は必ず実行可能となる．一般に部品を実行するためには前提条件が必要で，その条件は必ず部品に附属しているとは限らない．これに対して，この実行前提条件を附属することは，部品について少なくともひとつの実行パターンを保証しているといえる.

実行前提条件例により，部品の動作確認が可能になると，検索で見つかった類似部品から適切なものを選択したり，部品を変更する際にその部品を理解するのに有利である．例えば，目的の機能に対して部品が利用可能かどうかを調べる場合，機能を満たす入出力を揃えて実行結果と比較することが考えられる．また，どの入力に対して機能を満たしていないのかを調べ，その時の実行経路をたどることで部品変更や修正の範囲を特定することもできる.

さらに，変更を行なって新しく作成した部品に対して，変更前と同じ実行前提条件例を組み合わせて実行を試みることで，以前の機能に対しても正しく動作するのか，また以前の機能に対して変更が反映されているのかを検査することができる．特に，この実行前提条件例はそれ自身ソースコードであり，入出力の組み合わせだけのテストケースと比較して，より多様で柔軟なテストが適用可能である．例えば，テストに合わせて，実行前提条件例のコードについても追加，削除，変更が可能である.

このように，実行前提条件例は再利用時の部品検索，理解，変更を支援する情報となる．また，この実行前提条件例もスライシングを用いて生成しているため，その部品を実行するのに不必要な記述は削除されている.

3.2 利用条件例

利用条件例とは，作成した部品がもとのプログラムでどのように利用されていたのかを示したもので，実際に部品を合成する際の扱い方の一例となっている．これは，部品作成対象のプログラムから部品が影響を与えているコードを抜きだすことで得られる.

この利用条件例は，部品がどの機能を担っていたのか，また部品と部品外部との相関を理解する指針となる．部品の機能を把握しようとする場合，部品内部のコードを

解析し，その出力を明確にする方法と，利用している部品の外部から部品の出力を推測する方法がある．特に，作成中のコードに部品を埋め込もうとする際には，部品の前後のコードについて機能が明確であることが多く，後者の方法が有効である．利用条件例を提示することで，部品内部だけでなく部品外部からの理解も支援しているといえる．また，複数の部品に対してその利用条件例を比較することで，部品間の関係も明確になる．

さらに，この利用条件例もソースコードであり，部品本体と組み合わせることで必ず実行可能となる．よって，部品を利用しているコードまで含めて再利用対象として扱うことができ，検索，変更，合成が可能である．また，部品の機能を把握するためだけでなく，実際にコードとしてどのように記述すればよいのかも示している．

このように，利用条件例は再利用時の部品理解，変更，合成を支援する情報となる．また，この利用条件例もスライシングを用いて生成しているため，実際の利用に関して不必要な記述は削除されている．

4 ソフトウェア部品ライブラリ

本手法で作成したソフトウェア部品は作成者の登録要求によって，部品ライブラリに (Software Components Libraly) に蓄積される．このとき，部品の出力変数集合に対するベキ集合 (power set) を生成し，それをスライシング基準変数とするスライスを作成する．これらのスライスの包含関係は，スライス基準変数，つまり出力変数のベキ集合の各要素に対する包含関係と等しい．また，スライシングによって作成された部品は基準変数の値を決定する機能を持つといえるので，スライスの包含関係は機能に対する包含関係とも等しい．よって，部品の出力変数から部品の機能に対する包含関係を求めることが可能で，機能に従ってソフトウェア部品を階層化することができる．機能を包含するとは，包含されている機能に対しても動作を保証していることを指す．

いま，各ソフトウェア部品を節点とし，包含する部品から包含される部品に矢印をつけてできる有向グラフを部品グラフと呼ぶ．このグラフにおいて，接続先をもたない部品を特に単機能部品とする．作成したソフトウェア部品を部品ライブラリに登録する際には，お互いの単機能部品をそれぞれ比較し，スライス基準変数が等しいもの，つまり機能が等価なものを合併する．この操作により，ライブラリは階層 (hierarchy) と類似 (similarity) をもつ複数の部品グループで構成されることになる．部品グラフで表現された部品ライブラリの各要素は包含関係を表す矢印のみで関連づけられているので，部品階層がループになることは決して起こらない．ソフトウェア部品ライブラリを図7に示す．

このように階層化されたライブラリにおいては，階層が上位に向かうほど多機能で大きな部品となり，下位に向かうほど単機能で小さな部品となる．一般に部品を再利用する際には，部品のサイズが大きくなるほど再利用時の効果は高くなる．しかし，部品が大きくなるにつれ特化するので適用範囲が狭くなり，修正が必要となるため再利用コストが増大する．逆に部品が小さいほど再利用の適用範囲は広くなり，部品検索に成功する可能性は高くなるが，再利用時に部品の合成操作が多くなる．以上より，ライブラリ内で色々なサイズの部品が階層と類似関係で対応づけされていると，部品検索や合成に有利であるといえる．例えば，多機能な部品を必要とする場合，各機能に対して検索を行ない，それぞれ検索によって得られた部品から階層を上位にたどることが考えられる．これにより，機能の一部のみが満たさないために検索に失敗する部品に対しても，本ライブラリでは類似部品として提示可能である．また，検索によって見つかった部品から階層を上位にたどることで，その部品が他の部品とどのように合成されているかが得られる．逆に，階層を下位にたどることで，その部品がどのような部品を合成したものであるのかも取得できる．

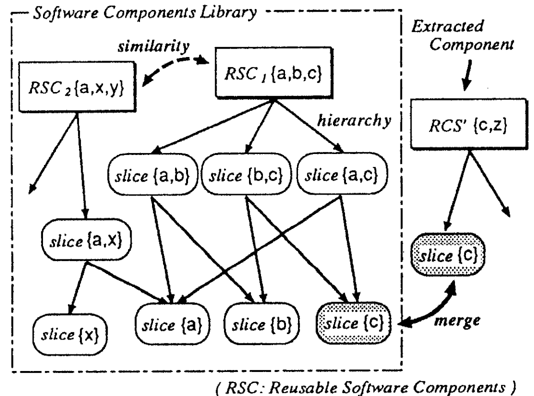

図 7: ソフトウェア部品ライブラリ

5 再利用指向ソフトウェア開発環境

ソフトウェアの再利用性を上げるためには，作成したソフトウェアが再利用されることを常に意識しながら開発を行なうべきである．特に，ソースコードにおいては形式性が高く，そのままの形で再利用されることが多いため，開発時に記述するコードが部品抽出や理解に大きく影響を与える．ソフトウェア開発時に再利用を意識しながらプログラムを作成するためには，作成者が現在記述しているプログラムからどのような部品が抽出できるのか，その部品がどのように再利用されるのかを随時及び容易に取得できる環境が望ましい．

本手法によるソフトウェア部品作成は，ソフトウェア開発におけるプログラム作成過程の任意の時点で任意のコードを対象とすることができ，部品作成基準である区

間と変数に制約はない．よって，プログラム作成中であっても部品の抽出が可能で，作成中のプログラムからその部品に対する付加情報が取得できる．また，作成者は区間と変数を指定するだけでよく，ソフトウェア部品の作成は自動である．以上より，本部品作成手法は上記のような開発環境を提供するのに適しているといえる．さらに，本手法を用いてプログラム作成中にソフトウェア部品を作成してみることで，作成者は再利用を意識するようになるだけでなく，作成中のプログラムに対する構造を見直す機会も得られる．

いま，このような環境においてプログラム作成を行なうと，プログラムを記述しながら再利用部品を作成することが考えられる．また，既存の部品を再利用するのと同時に，部品を変更，合成して得られたプログラムから新しい再利用部品を抽出することも可能である．つまり，部品を利用しながら部品を作成するというプログラミングスタイルを提供する．このようにソフトウェア開発を行なう場合は，部品検索，修正，変更，合成，新規コードの記述，部品作成の過程を繰り返すことで目的のプログラムを得ることになる．この繰り返し過程は，スパイラルモデル[6]や再利用指向ライフサイクルモデル[7]と相性が良く，各モデルのコード記述過程に埋め込むことが可能である．

6 ソフトウェア部品作成システム

本手法に基づき実現したソフトウェア部品作成システムは図8に示すCFG作成部，依存解析部，部品抽出部，部品登録部で構成されている．本システムでは，部品化対象プログラム記述言語として手続き型言語Pascalを用いる．部品作成基準の上限，下限節点 ($Bounds(n_u, n_l)$) はCFG作成部で作成されたCFG上で指定し，基準変数については，指定された区間内で定義，参照される全変数が選択の候補となる．ソフトウェア部品の構成要素である部品本体，実行前提条件例，利用条件例はそれぞれCFGとPascalのコードで出力される．また，入出力パラメータは変数を要素とする集合形式で表示される．これらの部品は再利用部品として，ユーザの意志でソフトウェア部品ライブラリに登録可能である．本システムにおける各構成要素の処理概要を以下に述べる．

(1) CFG作成部

Pascalプログラムを読み込み，CFGを作成し，表示する．また，ソフトウェア部品としてプログラムコードを提示するため，CFGの節点とソースコードとの対応をとる．

(2) 依存解析部

データ依存と制御依存を解析し，PDG(Program Dependence Graph)[3]を作成する．本システムでは，解析結果である参照，定義変数集合と各依存関係を表示する機能を備えている．

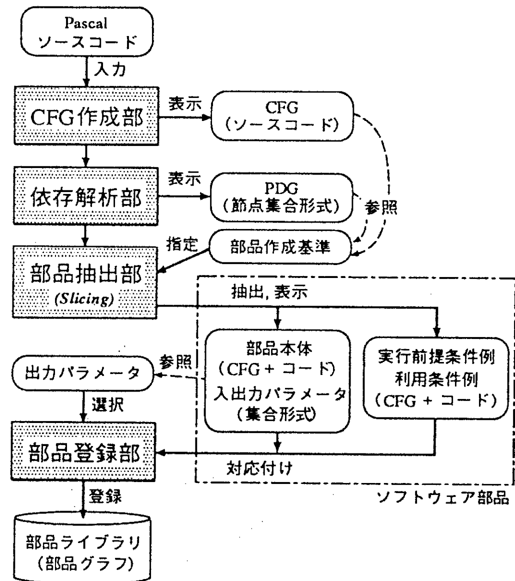

図 8: ソフトウェア部品作成支援システム

(3) 部品抽出部

与えられた部品作成基準をスライス基準に変換し，スライシングによってソフトウェア部品の各構成要素を抽出する．

(4) 部品登録部

作成されたソフトウェア部品をライブラリの適切な位置に登録する．登録の際には，出力パラメータの中からユーザがどれか1つの出力変数集合を選択する．

これらの処理を通して，本システムではソースコードからソフトウェア部品を作成する．ソフトウェア部品を作成する様子を図9に示す．

7 おわりに

ソースコードを再利用の対象にするとき，プログラムスライシングによってソフトウェア部品を作成する手法を提案した．本手法は，部品作成に対して統一的な抽出手続きを与えており，部品作成を容易にしている．また，実行前提条件例と利用条件例を提示して，部品の実行動作と利用方法の確認を可能とすることで，検索，変更，合成時の部品理解を支援している．さらに，本稿では本手法に基づき部品作成を行なった際に構築される部品ライブラリとソフトウェア開発環境について利点を述べ，実際に実現したソフトウェア部品作成システムを紹介した．今後の課題として，本手法によって作成するソフトウェア部品のライブラリ構築手法と部品検索手法の実現がある．また，作成したソフトウェア部品の有効性の評

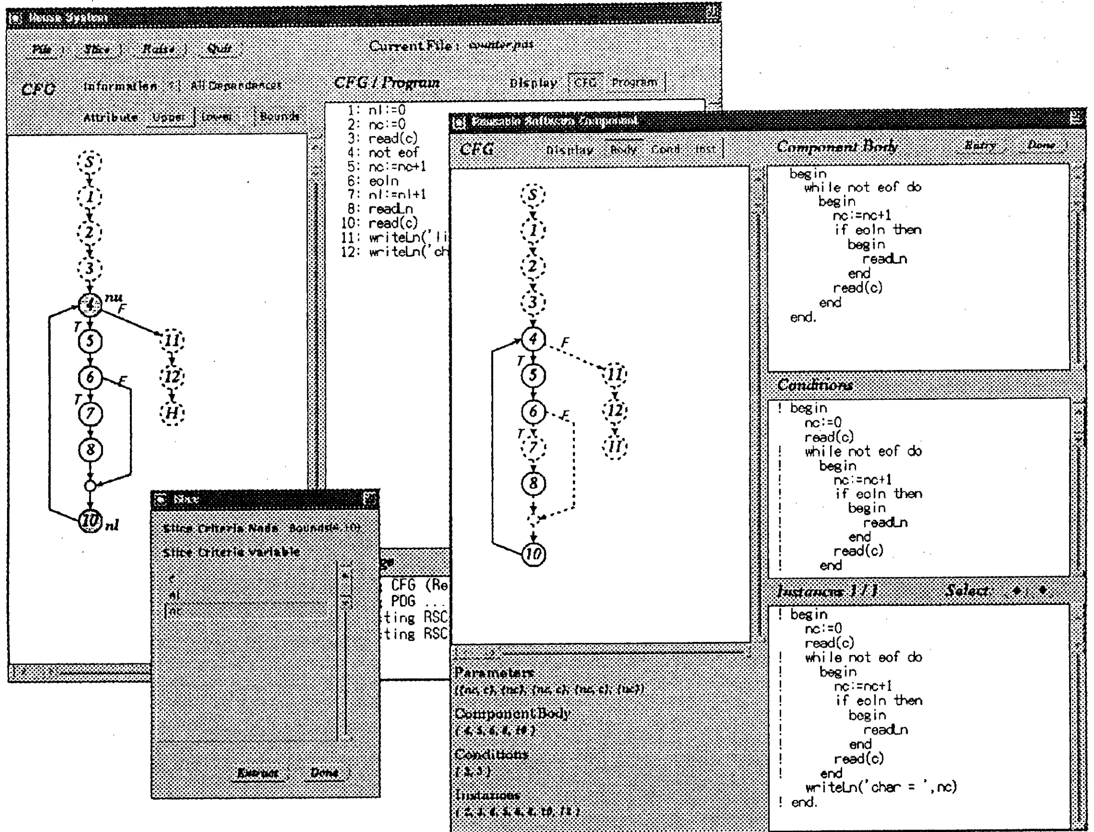

図 9: ソフトウェア部品作成システム ($C_C = <4, 10, \{nc\}>$)

価や再利用指向ソフトウェア開発環境に対する検討があげられる.

謝辞　日頃御指導御討論いただく伊藤正樹リーダーはじめ，グループの皆様に深く感謝します.

参 考 文 献

1) Biggerstaff,T. and Richter,C., "Reusability Framework, Assessment, and Directions", IEEE Software, Vol.4, No.2, pp.41-49, Mar. 1987

2) Weiser,M., "Program Slicing", IEEE Trans. Software Eng., Vol.10, No.4, pp.352-357, Jul. 1984

3) Ferrante,J., Ottenstein,K.J. and Warren,J.D., "The Program Dependence Graph and Its Use in Optimization", ACM Trans. Programming Languages and Systems, Vol.9, No.3, pp.319-349, Jul. 1987

4) Ottenstein,K.J. and Ottenstein,L.M., "The Program Dependence Graph in a Software Development Environment", Proc. ACM SIGPLAN / SIGSOFT Symp. Practical Programming Development Environments, ACM SIGPLAN Notices, Vol.19, No.5, pp.177-184, May. 1984

5) Aho,A.V., Sethi,R. and Ullman,J.D., "Compilers : Principles, Techniques, and Tools", Addison-Wesley, 1986

6) Boehm,B.W., "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol.21, No.5, pp.61-72, May. 1988

7) Gall,H. and Klösch,R., "Reuse Engineering: Software Construction from Reusable Components", IEEE COMPSAC'92, pp.79-86, Sep. 1992