# 文脈に基づいたソースプログラムとドキュメント間の識別子対応付け手法

後藤 英斗　　大久保 弘崇　　粕谷 英人　　山本 晋一郎

愛知県立大学 情報科学部 情報システム学科

**要 旨**

ソースプログラムと XML で記述されているドキュメントに含まれる識別子を自動的に対応づける手法を提案し，実装した．ドキュメント中の識別子に関しては文脈から定義か参照かを判断する．対応付けの応用として，ソフトウェアの理解支援のためにソースプログラムとドキュメントの相互参照を提供するツールと，両者の整合性を検査するツールを実装し，その有用性を確認した．

# The Method Based on the Context for Making Relations between Source Programs and Documents

Hideto GOTO, Hirotaka OHKUBO, Hideto KASUYA, and Shinichiro YAMAMOTO

Department of Information Systems, Faculty of Information Science,Aichi Prefectual University

**Abstract**

The method making relations identifiers which are extracted from source programs and documents is proposed and implemented in this paper. Identifiers contained of documents was classified into “define” and “Refer” by checking the context which contains it. Two tools appling relations mede by the method was implemented. One is the tool which enable to refer source programs and documents mutually for understanding software. Another is the tool which examines coordination of identifiers.

## 1　はじめに

### 1.1　背景

ソフトウェアの開発や再利用にあたり，多くの場合，作業者はソースプログラムを理解する必要があるが，モジュールの呼び出し関係が複雑であったり，各種識別子の役割がわからないなどさまざまな理由により，その作業は困難である．そこで，ソースプログラムを解析してプログラマの理解を支援するためのツールが提案されている．例えばCASE ツールプラットフォーム Sapid [2, 1] では，ソフトウェアモデルに基づいた解析を行いソフトウェアデータベース SDB を生成する．この SDB をアクセスルーチン AR を用いて利用することで解析結果に基づいたソフトウェアの理解を支援する機能を提供する．また，Eclipse や Visual Studio のような統合型の開発ツールでは，識別子にマ

ウスカーソルを重ねると関数のプロトタイプ宣言やオーバーロードのリストを表示できる．また，JavaDoc や Doxygen のようなソースプログラム中にドキュメントを簡潔に記述するための枠組も提案されている．

しかし，仕様書やライブラリのマニュアルなどは別ファイルに保存されているのが一般的である．また，開発者間でやりとりされるメールなどのように，ソースプログラムに埋め込むことは難しい．このように，同一のソフトウェアを構成するソースプログラムと関連ドキュメントが別々のファイルにそれぞれ記述されているため，相互に連携して利用することは現状では困難である．また，ソースプログラムを対象としたツールに比較して，ドキュメントを対象として解析や参照を支援するツールも少ない．

### 1.2 目的

我々は,「ソフトウェア＝ソースプログラム＋ドキュメント」という観点で両者の密接な連携を支援するための研究を行なっている．ソースプログラムとドキュメントは同じ１つのソフトウェアの構成要素であるが，別々のファイルに存在しているためにそれぞれ別の用途のためのものと考えられがちである．そこで，本稿では別々のファイルとして記述されるソースプログラムとドキュメントを対応付けることによって連携して活用する方法を提案することを目的とする．

ソースプログラムを閲覧しているとき，例えば現在注目している関数の仕様が知りたいという状況が考えられる．この場合，改めてドキュメントのファイルを開いて，検索機能や索引を用いて該当箇所を探さなくてはならない．その作業は決して難しいことではないが回数が増えると煩雑になり，作業効率にも影響を及ぼす．また逆に，ドキュメントからソースプログラム上の関連箇所を参照するという動作もそれ以上に煩雑である．この煩雑な参照作業は予め両者間を対応付けておくことで解決できる．対応付けによってリンクを作成しておけば，検索機能などを用いることなく参照が行える．

### 1.3 概要

本稿ではソースプログラムとドキュメントのそれぞれに含まれる識別子に注目し，これらを対応付けて利用する手法を提案する．手法を次の２つに分けて提案する．

- ソースプログラムとドキュメントの識別子の対応付け
- 対応付けを応用したツール

また，ドキュメントに含まれる識別子に関して，文脈の考慮を提案する．ソースプログラム中の識別子の出現を宣言と参照に分けられるように，これを「定義」と「参照」に区別して扱う方法である．

対応付けの目的は利用者により様々であると考えられるので，対応付けとその応用を別々の手法として提案する．利用者はまずソースプログラムとドキュメントの対応付けを行い，その結果を目的に合った応用ツールで利用する．

3 章で対応付けの手法について説明をし，4 章でその対応付けを応用するツールを２つ紹介する．

## 2 ソースプログラムとドキュメントの対応に関する研究

### 2.1 ADIOS

ADIOS [5, 6] はソフトウェアに関係する多種のドキュメントをアスペクト指向を用いて整理する手法を提案している．多くのドキュメントの中から対応するものを人手で見つけることは煩雑である．そこでソースコード中のキーワードからドキュメントへの参照を「関連」と呼び，その関連をソースコード中に一定の形式で埋め込む手法を用いている．埋め込まれた関連はこの手法を実装したツールである ADIOS によって収集され，これをアスペクト指向により横断的に整理する．関連には種類やキーワードが設定してあり，これらをもとに整理を行い同類の関連をまとめることで，関連先を推移的に辿ることができる．

ソースプログラム中には関連のみを埋め込む手法であるため，後述の Javadoc や Doxygen と

は異なりソースプログラムの可読性に影響を与える可能性は少ない．関連を手動で埋め込む必要はあるが，後から収集して整理することができるので利用性は高い．

### 2.2 机上デバッグ支援ツール Prio

Prio [7] に関する研究では仕様書に記述されている情報とプログラムに記述されている情報の間の不整合を検査することによってソフトウェアの不具合を発見することを目的としている．このツールを用いた机上チェックを行うことにより，仕様書とプログラムの不整合ミスを効率的に発見できる．また，後工程で評価作業時間を短縮することもできる．

この研究はプログラムの不具合を仕様書の不整合から見つけようという試みである．対応関係を示した XML 文書を基にソースプログラムと仕様書の間での矛盾を検出する．ソースプログラムのみでは見つけにくい不具合でも，仕様書と食い違っていることが発見できれば成果物の品質も向上させられる．

### 2.3 文芸的プログラミング

文芸的プログラミング（Literate Programming）[8] とは，Donald Knuth が提唱したプログラムの作成とその文書化を同時に行なう手法である．「コンピュータプログラムは計算機が実行可能でなくてはならないが，それは主な目的ではない．有用なコンピュータプログラムは人間が読むことができなくてはならない」という考えに基づいている．

プログラムとその文書をひとつのファイルに混在させながら追加したり修正したりして同時に開発していくことにより、完全に整合性のとれた文書とプログラムの開発を目指している．プログラムをまず作ってからその解説文書を作成する (またはその逆) という一般的な手法では、後で修正などを加える場合などは注意しないと両者の整合性がとれなくなってしまうことがよくあるが、文芸的プログラミングの手法では両者がひとつのファイルになっているのでそのようなことが起こりにくくなる．

### 2.4 Javadoc, Doxygen, Perldoc

ソースプログラムのコメント中に規定の形式でドキュメントを埋め込む方法を持つプログラミング言語がある．Java では Javadoc，C++では Doxygen，Perl では perldoc がこのドキュメントを抽出するツールである．

**Javadoc** クラス，メソッド，フィールドの直前にコメントとしてそれぞれの説明を記述する．例えば “`@param` 引数名 コメント” という記法でメソッドの引数の説明を記述する．`javadoc` コマンドでこれらの記述を HTML ファイル群にして取り出せる．記述の中に HTML 断片を埋め込むこともできる．

**Doxygen** 記法は Javadoc と似ている．`doxygen` コマンドによって HTML ファイルをはじめ，TEX や RTF のファイルも生成できる．

**Perldoc** perl プログラムのファイルには POD という形式で説明を記述できる．この説明記述は `perldoc` コマンドによって取り出し，`pod` コマンド群によって man 形式や HTML 形式などに変換して参照できる．

### 2.5 関連研究の考察

ADIOS や Prio は別個にファイルに記述されたソースプログラムとドキュメントを連携して利用できる．このツールを用いることでソースプログラムとドキュメントの対応関係を集約してソフトウェア開発に利用できる．しかし，対応を人手で指定しなくてはならない点が短所である．ソースプログラムの作成とドキュメントの作成のほかに，ソースプログラムとドキュメントの対応づけをするという作業が必要となる．

文芸的プログラミングは，実際のコードは自然言語で記述された文書の部分が圧倒的に多く，それに対して実行コードに変換される部分は僅かしかない．Knuth も著書 [8] のなかで「プログラムを文学と捉える」,「プログラムは人間に読めなくてはならない」と述べているので，そのようになるのは当然である．文芸的にプログラムするということは先に自然言語の文書を作成する必要がある．しかしプログラ

ミングにおいてはプログラム作成中に仕様や方針が変わるということは多いので，このような場合に文書も変更する必要があり不便である．

Javadoc などのソースプログラムのコメントにドキュメントを埋め込む手法ではソースプログラム中の説明を行う要素の近くに説明記述が存在するため，同時に修正を施すことができ，ソースプログラムとドキュメントの間の整合性を保つことに長けている．しかし，説明記述が増えるとソースプログラムのファイルが巨大になり，またファイル中にプログラムのコードよりも文章のほうが多くを占めるようになるのでソースプログラムの可読性に影響する．

ADIOS，Prio は対応関係を作成者が手作業で設定させる手間が短所である．この作業を自動で行うことができれば利用しやすくなる．一方，文芸的プログラミングやソースプログラムにドキュメントを埋め込む手法はソースプログラムのファイルが巨大化，複雑化することが短所である．この点も ADIOS や Prio の短所と同様にソースプログラムとドキュメントの間の対応箇所を自動で抽出できるようにすることで解決できる．１つのファイルの中にソースプログラムとドキュメンを混在させなくても，抽出した対応関係を用いて相互に参照できればソースプログラムとその説明記述が近くにあるのと同様の効果が得られる．

# 3　識別子の対応付け

本稿では，ソースプログラムとドキュメントの両方に出現する同じ種類で同じ名前を持つ識別子の出現を見つけて組にする作業を「対応付け」と呼び，この組を「対応」と呼ぶ．ソースプログラム，ドキュメントそれぞれで同じ識別子は複数回出現する．従って，ソースプログラム中の１つの識別子の出現に対してドキュメント中の複数の出現箇所が対応付けられる．また，逆にドキュメント中の１つの識別子の出現に対してもソースプログラム中の複数の出現箇所が対応付けられる．

入力ファイルは識別子の抽出のためにソースプログラム，ドキュメントともに XML でマークアップしたファイルを対象とする．ソースプログラムに対する意味解析は予め行われているものとして，その結果にしたがって XHTML でマークアップされたものを用いる．このマークアップは SPIE によって行う．ドキュメントは DocBook と呼ばれる形式で記述されたものを対象とする．

## 3.1　ソースプログラムのマークアップ

ソースプログラムは SPIE [4] によってマークアップされたものを扱う．SPIE は CASE ツールプラットフォーム Sapid [1, 2] に含まれるツールであり，Sapid によるソースプログラムの解析に基づいてソースプログラム内のクロスリファレンス機能を提供する．ソースプログラムを XHTML でマークアップし，また各種識別子についての情報をまとめた情報テーブルという XML を出力することでハイパーリンクを利用したクロスリファレンスを Web ブラウザを用いて実現する．

表 1: SPIE が出力するソースプログラムの class 属性の意味

| 属性値 | 意味 |
|---|---|
| Func | 関数名 |
| Var | 大域変数名 |
| L_Var | 局所変数名 |
| Const | 定数値 |
| Arg | 引数名 |
| Tag | 構造体タグ名 |
| Member | 構造体メンバ名 |
| Macro | マクロ名 |

SPIE が出力する XHTML 化したソースプログラムでは識別子は<a>でマークアップされ，その class 属性に指定される表 1 に挙げる値によって識別子の種類が示されている．また name 属性にはソースプログラム中でユニークな値が指定されているのでこれによってファイル中の要素を特定することができる．本手法では，表 1 に挙げた値を class 属性に持つ要素を識別子として対応付けの対象とした．

### 3.2 ドキュメントの記述形式

ドキュメントはDocBook [3] と呼ばれるXMLの形式のものを扱う．DocBookとはソフトウェアの関連文書向けのXMLタグセットである．元々UNIXの文書交換のために作成され，現在もFDP [1]やLDP [2]で用いられている．またオープンソースソフトウェアのドキュメンテーションプロジェクトであるOSWG [3]でも採用されており，多くのソフトウェア関連文書の記述に用いられるXML形式であることから本研究でも対象として採用した．

DocBookにはプログラムに含まれる要素を示すためのタグが用意されているので，これを用いて識別子を得る．本研究において，対応付けの対象となる要素を表すタグを表2に示す．

表 2: DocBookのプログラムに関する意味をもつタグ

| タグ | 意味 |
|---|---|
| function | 関数名 |
| symbol | マクロ |
| literal | リテラル値 |
| structname | 構造体名 |
| structfield | 構造体のメンバ |
| type | 型 |
| varname | 変数名 |

### 3.3 文脈情報の利用

ソースプログラム中の識別子の出現には「宣言」であるものと「参照」であるものの２種類がある．このことをドキュメントにも反映し，ドキュメント中の識別子の出現を「定義」と「参照」に区別する．DocBookの文脈を利用し，識別子の親要素に表3に挙げる要素を持つ場合に「定義」であると判断する．この表に挙げたタグは本質的には識別子の定義であることを示すタグではない．しかし，意味やその使用法を考慮して識別子の定義であることを示していると思われるものを選択した．

表 3: 識別子を定義と判断するタグ

| タグ | 意味 |
|---|---|
| title | タイトル |
| grossterm | 定義リストの項目名 |
| funcprototype | 関数のプロトタイプ |

識別子の出現を「定義」と「参照」に区別することは，対応付けの応用において利用できる．例えば，ある関数名についてその意味を調べるためにドキュメントを参照する場合，「定義」の記述に意味が記述されている可能性が高い．また，ソースプログラムとドキュメントの整合性を検査する場合にもドキュメント中に「定義」の無い識別子のリストアップなどの詳細な検証を行うことができる．

### 3.4 対応付け

ソースプログラムとドキュメントから抽出した識別子を表4に示すタグの関係に従って対応付ける．

対応付けを応用するためにはファイル中の識別子の出現を特定する手段が必要である．ソースプログラム中の識別子についてはSPIEが付与している要素のname属性の値を用いる．この値は全体のプログラムでそれぞれがユニークである．一方，ドキュメント中の識別子は要素のid属性を用いる．これはファイル中でユニークであることが定められているが，必須

表 4: タグによる対応関係

| 対応の種類 | DocBook | SPIE |
|---|---|---|
| 関数 | function | Func |
| 変数 | varname | Var, L_Var |
| 定数 | literal | Const |
| 引数 | parameter | Arg |
| 構造体タグ | structname | Tag |
| 構造体メンバ | structfield | Member |
| マクロ | symbol | Macro |

[1] FreeBSD Documentation Project
[2] Linux Document Project
[3] Open Source Writers Group

の値ではないので筆者が付与していない場合が多い．そのため，対応付けの前処理としてこの id 属性を付与することとする．

対応付けの処理の流れを図 1 に示す．ソースプログラムを SPIE によりマークアップし，識別子を抽出する．ドキュメントは id 属性を補完して識別子を抽出する．それぞれから抽出した識別子に対して対応付けを行い，対応表を出力する．対応表は XML ファイルで保存し，応用ツールが利用できるようにする．

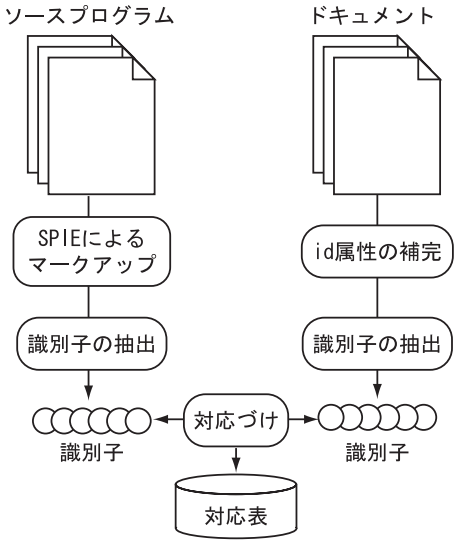


図 1: 対応付けの流れ

### 3.5 評価

対応付けシステムを実装し，愛知県立大学情報科学部情報システム学科のプログラミング実習で作成するコンパイラとその 2002 年度版指導書を対象に対応付けを行い，検証した．

結果を表 5 に示す．出力された対応を観察すると局所変数や定数の対応付けには誤りが多かった．局所変数の対応付けに誤りが多いのは後述のスコープルールの問題によるものである．定数については特に意味のあるものはマクロ定数で表現され，定数をそのまま記述しているものは特に意味をもたないものばかりであった．そのような定数はドキュメントにも記述されず，正しく対応付けができなかった．局所変数と定数についてはドキュメントに記述する必要のあるものが少ないので，利用者がオプションによって指定した場合のみ対応付けを行うようなシステムにする必要があると判断した．ソースプログラムではスコープ内で重複しなければ，異なる実体を示す同名の識別子を定義できる．しかしドキュメントから識別子を取り出す際にはこのスコープを考慮することができないので，同名の識別子は全ておなじ実体を示すものとして扱われてしまう．特に局所変数や引数についてはこの問題が重要であり，より正確な対応付けのためには解決しなくてはならない．

表 5: 対応付けの結果

| | |
|---|---|
| ドキュメントのファイル数 | 8 |
| 行数 | 2643 行 |
| ファイルサイズ | 104KB |
| ソースプログラムのファイル数 | 5 |
| 行数 | 4667 行 |
| ファイルサイズ | 115KB |
| 対応の数 | 1803 |
| 誤った対応 | 31 |

## 4 対応付けの応用

3 章で行った対応付けを応用するツールとして次の 2 つを紹介する．

- クロスリファレンスツール
- 整合性検査ツール

クロスリファレンスはソフトウェアを再利用するなどの目的でその理解のために利用できるツールである．ソースプログラムとドキュメントの相互参照機能を提供し，ソフトウェアの理解を支援する．

整合性検査ツールはドキュメントの妥当性を検証して校正作業をしたり，ソースプログラムを仕様書と比較して作成したプログラムが仕様に則っているのかを検証したりすることに利用できる．

### 4.1 クロスリファレンスツール

そこで対応付けに基づいたソースプログラムとドキュメントの間での相互参照機能を提供するツールを提案する．XML ファイル群を出力し，Web ブラウザ上でその機能を実現する．図 2 に示すように SPIE がソースプログラム内のクロスリファレンスを提供するために出力するファイルに，ドキュメンとの相互参照をリンクとして埋め込む．SPIE はソースプログラムを XHTML でマークアップすると共に，各種識別子についての情報をまとめた情報テーブルを XML ファイルとして出力する．この情報テーブルに，それぞれの識別子が出現するドキュメント上の箇所へのリンクを列挙することでソースプログラムからドキュメントへの参照を可能にする．またドキュメント上の識別子は，対応付けられた識別子が該当する情報テーブルへのリンクを付加する．このように，情報テーブルを介してソースプログラムとドキュメントの相互参照を実現する．

識別子のドキュメント上の出現箇所を列挙する際，3.3 節で示した識別子の「定義」と「参照」の区別によって表示方法を変化させる．特に優先的に参照したいであろう「定義」の記述へのリンクは強調表示をする．このようにすることで利用者はリストの中から定義へのリンクを選択することができる．

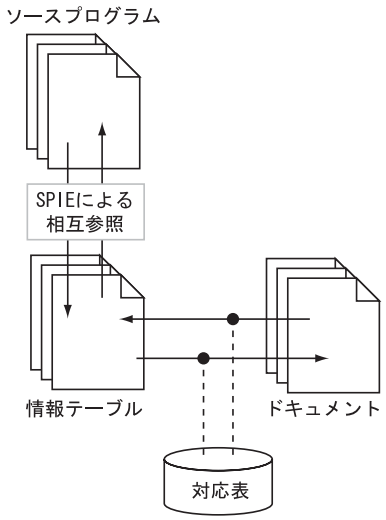


図 2: クロスリファレンスの実現

実際の Web ブラウザでの画面を図 3 に掲載する．図中左下のウィンドウには情報テーブルが表示され，右上のウィンドウにドキュメントが表示されている．右上のウィンドウはソースプログラムとドキュメントの兼用で，情報テーブルで指定されたものが表示される．


図 3: クロスリファレンス画面

### 4.2 評価

3.5 節と同様にプログラミング実習の教材を利用して機能を検証した．実際にクロスリファレンス機能を提供するファイル群を生成し，その講義を受講したことのある学生 5 人に使用した感想を聞き，以下のような回答を得た．

- 実際の指導書は内容が前後している部分があり，必要な箇所を探すのに手間取ることが多かったが，このツールを使えばすぐに見つけられた．
- 前回の内容を参照しなくてはならないときに，忘れてしまっていてもすぐに指導書から見つけられた．

### 4.3 整合性検査ツール

１つのソフトウェアの構成要素であるソースプログラムとドキュメントの間では，整合性が保たれている必要がある．もし，整合性が崩れているのならばどちらかの内容が誤っている可能性があり，修正しなくてはならない．そこで，対応付けに基づいてこの整合性についての検査を行うツールを提案する．ソースプログラムとドキュメントの間での識別子の存在に注目した整合性検査を行う．ソースプログラムの

みに含まれる識別子とドキュメントのみに含まれる識別子をリストアップする．前者は主にドキュメント上での書き漏れが原因である．ソースプログラムの修正が先行し，ドキュメントの修正が行われていない場合に起こりやすい．後者は例えば仕様書で決められた識別子名が実際には用いられていない場合などに検出される．仕様にそっていないため，ソースプログラムの不具合の発見にも役立つと考えられる．

このツールの主な用途はドキュメントの記述と校正作業を想定している．しかし，不整合の検出から不具合などを検出できる可能性もあるのでテストやデバッグの作業にも利用できる．

また，3.3 節で示した「定義」と「参照」の区別により，ドキュメント上に「定義」のない識別子をリストアップすることも可能である．

### 4.4 評価

3.5 節同様，プログラミング実習の教材を用いてツールの検証を行った．検証では重要ではない局所変数と定数の検査を行っていない．図 6 のような結果を得た．

表 6: 整合性検査の結果

| | |
|---|---|
| ソースプログラム中の識別子数 | 534 |
| ドキュメント中の識別子数 | 99 |
| 対応の数 | 1804 |
| ソースプログラム中のみの識別子 | 451 |
| ドキュメント中のみの識別子 | 41 |

ソースプログラムでは 84%の識別子が，ドキュメントでは 41%の識別子が不整合として検出された．不整合とされた割合が高い理由を考察し，次のようにまとめた．

- 用いたデータが教材であり，ドキュメントでプログラムの全ての詳細を説明する必要がないため．
- ドキュメントでは作成したプログラムには関係の無い関数名なども含まれており，これらも検出してしまうため．

前者は整合性検査に用いるドキュメントとしては仕様書などの詳細を記したものを対象とする必要があることを示している．後者からは 1 つのドキュメントには 1 つのソフトウェアに関する記述のみであるという制限を設ける必要があると考えられる．

リストアップされた識別子を観察した結果，プログラミング実習の指導書として記載されているべき識別子が記載されていなかった例を関数名で 3 件，マクロで 1 件発見できた．受講していた学生に聞いたところ，実際に実習で意味が調べられずに困ったもの，またはその存在を知っていれば作業を減らすことができたのもであった．よって，これらはドキュメントの記載漏れと判断でき，このツールは有効である．

現在は単純に対応付けられているかどうかの検査のみであるので，誤記などを検出することは不可能である．近くに記述されている識別子や文脈などの情報を利用して，より高度な整合性の検査を実現すればより有用なツールになる．

## 5 おわりに

### 5.1 まとめ

本稿ではソースプログラムとドキュメントのそれぞれに含まれる識別子を対応付け，その結果を応用する手法を提案した．

対応付けについては，ドキュメントから識別子を抽出する際に，その文脈から「定義」と「参照」を区別する方法を用いた．このことによって識別子名の一致のみによる対応付けよりも有用なものになった．

対応付けを行うプロセスとその結果を応用するプロセスを分けたことでそれぞれにまだ改良を施すことができる．対応付けをある特定の目的のために行わないので，本稿で紹介した応用ツールのほかにもさまざまな用途に利用できる．

### 5.2 今後の課題

現段階では応用ツールを利用するためには対応付けを行う必要がある．従って，それぞれの編集に対してインタラクティブな機能を提供することはできない．ソースプログラムを作成中にドキュメントを素早く参照する機能や，

ドキュメントを誤りを直しながら校正する作業のためには，こうした変更に対して毎回の対応づけを必要としない方法が必要である．

本稿で提案した対応付けシステムではドキュメント上の識別子に関して，プログラムのスコープルールを考慮していない．ソースプログラムには識別子の有効範囲であるスコープがあり，そのスコープ内で重複しなければソースプログラム内で同一の識別子名を複数の場所で別のものとして使うことができる．ドキュメント上の識別子についても同様のルールが適用でき，異なるのもを示す同名の識別子が存在しうる．しかし，本手法では異なるものを示す識別子も同一のものとして扱ってしまう．ソースプログラムとドキュメントからそれぞれ取り出した識別子の単純な比較だけではこのスコープルールの問題が解決できない．そこで，局所変数については近くにある関数名などを調べることによってどの関数内で用いられる変数なのかが調べられると考えられる．文脈や近くにある要素などによって意味を考慮した対応付けを行い，より多くの意味を持った対応付けを行う必要がある．

## 謝辞

ご指導，ご議論頂いた愛知県立大学情報科学部 稲垣康善教授，名古屋大学大学院情報科学研究科 阿草清滋教授，および山本研究室の皆様に感謝します．本研究は文部科学省科学研究費補助金基盤研究 (A)『「ソフトウェア=プログラム+ドキュメント」の視点に基づく多言語対応大規模コーパス』(課題番号 16200001) を受けて行われた．

## 参考文献

[1] Sapid, http://www.sapid.org

[2] 福安 直樹, 山本 晋一郎, 阿草 清滋, “細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid”, 情報処理学会論文誌, Vol.39, No.6, pp.1990–1998, 1998

[3] OASIS, http://www.oasis-open.org

[4] 大橋 洋貴，山本 晋一郎，阿草 清滋，“ハイパーテキストに基づいたソースプログラム・レビュー支援ツール”，電子情報通信学会技術研究報告，Vol.98, No.28，pp.15–22，1998

[5] 大場勝，権藤克彦，“アスペクト指向を用いたドキュメント整理法の提案”，日本ソフトウェア科学会 第 7 回プログラミングおよび応用のシステムに関するワークショップ，2004, http://spa.jssst.or.jp/2004/pub/papers/04027.pdf

[6] 大場勝，“OS 入門用の教材を事例としたアスペクト指向によるソースコードとドキュメントの関連づけ”，北陸先端科学技術大学院大学情報科学研究科 修士論文，2004, http://www.jaist.ac.jp/library/thesis/is-master-2004/paper/m-

[7] 山田信幸, 鈴木幹雄, 坂守,“XML 形式の仕様書作成によるソフトウェア机上チェックの効率化－机上デバッグ支援ツール Prio（プライオ）－”，ソフトウェアテストシンポジウム 2004，ソフトウェアテストシンポジウム 2004 予稿集，pp.92–99, http://www.swtest.jp/, 2004

[8] D.E.Knuth, 有澤誠 訳, “文芸的プログラミング”, アスキー出版局, 1994