2005－SE－149（8）
2005／7／29

XML 記述によるソフトウェアリポジトリを用いたコード検索

渥美 紀寿 †　　山本 晋一郎 ‡　　阿草 清滋 §

† 南山大学 数理情報学部 情報通信学科
‡ 愛知県立大学 情報科学部 情報システム学科
§ 名古屋大学 情報科学研究科 情報システム学専攻

natsumi@nanzan-u.ac.jp
yamamoto@ist.aichi-pu.ac.jp
agusa@is.nagoya-u.ac.jp

概要

本稿では，XML 記述によるソフトウェアリポジトリを構築し，これを利用する事でプログラム要素間の関連性を考慮した検索を容易に実現できる事を示す．プログラム要素間の関連性とは，例えば，変数には型が付けられるので，変数と型との関連を表わす．ソフトウェアリポジトリは，ソースコードに構文情報を付加した CX-model と各プログラム要素に対するクロスリファレンス情報で構成される．これらの XML 記述の各要素間の対応付けを行い，ソースプログラムブラウザを実装した．プログラム要素間の関連性を考慮した検索機能をこのソースプログラムブラウザに組み込む事によって，検索結果からソースコードへの参照を容易に行えるようにした．

キーワード:　コード検索　正規表現　XML

Code Search by XML Software Repository

Noritoshi Atsumi†, Shinichiro Yamamoto‡ and Kiyoshi Agusa§

†Faculty of Mathematical Sciences and Information Engineering, Nanzan University
‡Faculty of Information Science and Technology, Aichi Prefectural University
§Graduate School of Information Science, Nagoya University
†natsumi@nanzan-u.ac.jp　‡yamamoto@ist.aichi-pu.ac.jp　§agusa@is.nagoya-u.ac.jp

abstract

This paper shows that it becomes easy to implement search function by XML Software Repository, which takes account of the relationships between each program elements. For example, a variable and the type is one of the relationship between each program elements. The software repository consists of cross-reference information and a CX-model, which contains source code with additional syntax information. The repository maps relationships between each elements and we implement Source Program Browser. Incorporating a code search function in the Source Program Browser makes it possible to reference source code from a search result.

Keywords:　Code Search　Regular Expression　XML

1 はじめに

プログラムを理解する際には，ソースコードの中から特定の情報を検索し，その周辺のコードを参照する事を繰り返し行う．そのため，コード検索はプログラム開発において非常に重要である．

ソースコードの検索に Unix のコマンド `grep` に代表される正規表現による文字列検索ツールが非常によく利用される．しかし，これらの検索ツールはソースコードを文字列として検索を行うため，ソースコード中の各要素の意味を無視した検索となる．例えば，関数 A の呼び出し箇所を検索しようとすると，関数の A だけではなく，変数や型，コメントなどマッチする文字列全てが結果として得られる．このため，得られた結果から，1 つずつ関数かどうかを調べ，必要な結果を取り出すという作業に手間や労力をかけなければならない．

Semantic Grep[3] や gonzui[8] ，統合開発環境として知られる Eclipse の Java の検索機能は，プログラムの意味を考慮した検索を行う事ができる．これらはプログラムを構文解析し，ソースコード中の各要素の意味を調べる事によって，個々の要素を区別した検索が可能となっている．しかし，コードを検索する際には，構文情報から得られる意味だけではなく，より高度な検索が求められる．例えば，返り値の型を指定した検索や，引数の型を指定した検索などである．

これらの情報は単純に各要素に対して構文情報を付加しただけでは困難である．本研究では，ソースコードを構文解析し，ソースコード中の各構文要素に構文情報を付加し，XML 記述として生成する．このモデルを CX-model と言う．さらに，ソースコード中の各プログラム要素に対するクロスリファレンス情報を XML 記述として生成する．このクロスリファレンス情報を基にコード検索を行う事によって，高度な検索機能を実現する．

コード検索は，開発者がソースコードを理解するためだけに利用されるのではなく，CASE ツールでも内部で行っている操作である．その操作を容易に実現できれば，CASE ツール開発において，必要な情報を検索する部分を容易に実現する事が可能となる．そのため，CASE ツールでも利用できるように XML 表現を用いる．

2 ソースコードの XML 表現

ソースコードを解析し，構文木に従って XML 文書として表現する研究として JavaML[2]，ACML[5] や JX-model[6]，[7] がある．これらの研究では，ソースコードを構文情報に従ってタグ付けを行い，XML 文書を生成する．各要素には ID がつけられ，定義と参照の関係をこの ID によって関連付けている．この定義と参照の関係を利用してクロスリファレンサやスライシングが実現されている．これらのようにソースコードを XML によって構文情報の註釈を付ける事で，文字列としてのソースコードを意味を持った XML の木構造として表現する事ができる．また，XML で表現する事で，様々な CASE ツールで利用する事ができる．

2.1 JavaML

Badros によって提案された JavaML[2] は，Java ソースプログラムを XML の文書モデルに基づいて表現する試みである．JavaML のスキーマは，Java 言語構文には依存せず，一般的なオブジェクト指向言語に対して適用できるような抽象モデルに基づいて定義されている．定義されている XML 要素は 70 種余りであり，クラス，メソッド，フィールド，文字定数，数値定数，コメントといったソースプログラムに含まれる細粒度のソフトウェア部品をモデル化している．

2.2 ACML

権藤らによって提案された ACML[5] は，ANSI C ソースプログラムの構造と意味情報を表現する XML 文書形式である．CASE ツール作成に役立つ解析情報を多く提供しており，実際にスライサを短時間で作成した事で，その有効性が示されている．

ACML のスキーマでは，ソースプログラム上の表記によらず，同じ意味を持つ構成要素は同じ XML 要素としてモデル化し，50 種余りの XML 要素が定義されている．生成される情報は，コンパイラが生成する記号表に近い複雑な構造を持ち，生成されるファイルサイズは大きい．

2.3 JX-model

吉田らによって提案された JX-model[6] は，ソースコードの手続き内部の細粒度要素までを解析し，プログラム要素の構成関係を XML の親子関係として表現し，定義・参照関係を ID/IDREF 型の属性により表現したモデルである．JX-model では，20 の非終端要素と 7 つの終端要素によってソーソコードを表現している．終端要素は，字句要素であり，子ノードにソースプログラム断片であるテキストノードのみを持つ．非終端要素は構文情報であり，子ノードには他のモデル要素を持ち，テキストノードを持たない．

また，吉田らは JX-model で表現できていないファイルを跨がった要素間の関係を表現するモデルとして JR-model[7] を提案している．JR-model では，JX-model の要素間の関係を RDF を用いて表現している．RDF 関連技術を利用する事によって，RDF によって関連付けられた関係を推論により発展させる事ができるため，手続き的に操作する事なく，宣言的に情報を得る事ができる．

2.4 問題点

JavaML，ACML では，プログラム要素間の関係を ID/IDREF により表現しているが，この関係を逆に辿るような操作ができない．また，ファイルを跨がった操作ができない．例えば，ある関数の呼び出しがどのファイルのどこで行われているか調べるには，全ての関数呼び出しから IDREF 属性を参照して取捨選択をしなければいけない．

この問題は JR-model ではプログラム要素間の関係を RDF により関連付ける事により，解決している．しかし，関連付けを行っているだけなので，それらの情報を参照するには，その関連を順に追っていく必要がある．そこで，本研究では，個々のプログラム要素に対して，関連している情報をクロスリファレンス情報として蓄積する手法を提案する．このクロスリファレンス情報については 4.2 で述べる．

3 既存のコード検索手法

3.1 正規表現による文字列検索

正規表現による検索は古くから利用され，現在も使われている最も単純な検索手法である．この検索は grep を代表とする UNIX のコマンドやエディタの機能として利用できる．この正規表現による検索手法はソースコードに対してもよく利用され，統合開発環境のエディタの 1 機能として用意されている．しかし，ソースコードには各要素に意味があり，開発者はその意味を考慮した検索をする事が必要である．正規表現による文字列検索では，様々なパターンによりマッチさせる事が可能であるが，得られる結果は意味を考慮していないために膨大となってしまう．また，その中から必要な情報を探すのは大変である．

軽量で便利な検索手法ではあるが，これらの検索ツールは，プログラムに特化した検索ではなく，文字列から正規表現にマッチする箇所を検索するものなので，プログラム特有の構文に関係無く検索が行われる．しかし，プログラムを検索する場合，構文情報を考慮した検索を行う事が必要となる．例えば次のような場合がある．

- 正規表現にマッチする変数を検索したい
- 正規表現にマッチする関数宣言を検索したい
- 引数に正規表現にマッチする型を持つ関数を検索したい

単純に文字列マッチングによる検索ではこれらのプログラムが持つ意味が無視されてしまうため，たくさんの検索結果が得られる．プログラム中の意味が異なる文字列まで検索結果として得られる．このため得られた検索結果から必要な情報だけを抽出する事が必要となる．

3.2 意味を考慮した検索

意味を考慮した検索として, Semantic Grep[3], gonzui[8], Eclipse の検索機能を挙げる．Semantic Grep では，ソースコードを構文解析した結果と関数の呼び出し関係や包含関係などの関係を利用した検索ができる．しかし，検索結果は文字列として出力するのみで，ソースコードとのリンクができていないため，該当箇所は自分で探す必要がある．

gonzui 意味を考慮した検索が可能で，検索する対象を関数や変数など選択する事ができ，これらを区別して検索する事ができる．Web を利用して検索する事ができ，ソースコードの参照もできる．

しかし，プログラム要素間の関連性を考慮した検索はできない．

複数の言語に対応したオープンソースの統合開発環境 Eclipse がサポートする Java の検索機能も意味を考慮した検索が可能である．具体的には下記に示す検索が可能である．

- 検索対象の文字列が Java の要素のいずれかを指定する事が可能
- 検索対象の文字列が宣言，参照，両方のいずれかを指定する事が可能
- フィールドに対しては読み取りアクセスか書き込みアクセスかの指定が可能

Eclipse の検索機能は既存の検索ツールに比べ，検索対象を十分に絞り込む事ができるが，プログラム要素間の関連性を考慮した検索はできない．

4 XML 記述によるソフトウェアリポジトリ

XML 記述によるソフトウェアリポジトリは，ソースコードに構文情報を付加した CX-model とクロスリファレンス情報から構成される．本章ではこれらについて述べる．これらの XML 記述の生成には，CASE ツールプラットフォーム Sapid [1]，[4] を利用した．

4.1 C 言語のソースコードモデル CX-model

C 言語を対象として，ソースコードの構成要素を抽象化したモデル CX-model を設計した．また，このモデルの構成要素を表 1 に示す．

CX-model では全部で 15 個の非終端要素と 8 個の終端要素によって C 言語のソースコードを表わしている．CX-model の設計には，我々の研究グループにより開発された JX-model [6] を参考にした．CX-model は JX-model 同様に CASE ツール開発者の視点によりプログラム要素を抽象化して表現し，構成される木構造が見通しよくなるように工夫している．ファイル内部の関連の表現としては，識別子の要素に対して，その定義箇所へのリンクを表わす属性 defid が存在する．

モデル要素のうち，終端要素は字句情報であり，子ノードにソースコード断片であるテキストノードのみを持つ．非終端要素は構文情報であり，子ノードに他のモデル要素を持ち，テキストノードを持たない．字句情報は元のソースプログラムをそのまま保存しており，XML のタグを全て除くと，元のソースプログラムを復元する事ができる．

CX-model のインスタンスは Sapid の C 言語の前処理前のプログラムに対するモデル I-model と前処理後のモデル P-model を利用する事によって，前処理前の開発者が記述したソースコードに対して構文情報を付加し，XML 記述として生成される．

表 1: CX-model の構成要素

要素名	C ソースコード上の要素
File	ソースコードファイル全体を表わす
Include	include 文を表わす
MacroDef	マクロ定義を表わす
Typedef	typedef を表わす
Function	関数定義を表わす
Global	大域変数を表わす
Local	局所変数を表わす
Pram	関数の引数を表わす
Type	型を表わす
Tag	タグを表わす
Member	メンバの宣言を表わす
Enum	列挙子の宣言を表わす
Stmt	文を表わす
Label	ラベル定義を表わす
Expr	式を表わす
pp	include 文およびマクロ定義以外のプリプロセス命令を表わす
ident	識別子を表わす
literal	リテラルを表わす
comment	コメントを表わす
kw	キーワードを表わす
op	演算子を表わす
sp	空白文字を表わす
nl	改行文字を表わす

4.2 クロスリファレンス情報

クロスリファレンス情報とは表 2 に示した個々のプログラム要素に対する情報とする．

表 2: プログラム構成要素とそれに対する情報

ファイル	ID
関数	定義，引数，使用変数，使用マクロ，呼び出し関数，被呼び出し関数，コメント
変数 型定義 列挙子 構造体のメンバ	定義，参照箇所，コメント
マクロ	定義，マクロ本体，マクロ引数，参照箇所，コメント
定数	出現箇所の位置情報

これらの情報の詳細を下記に示す．

- 定義情報
  識別子名，型情報，定義の位置情報
- 引数情報
  識別子名，型情報
- 使用変数情報
  変数の種類 (大域変数，局所変数)，型情報，変数のクロスリファレンス情報へのリンク
- 使用マクロ情報
  マクロのクロスリファレンス情報へのリンク
- 呼び出し関数情報
  呼び出し箇所の位置情報
- 被呼び出し関数情報
  呼び出されている箇所の位置情報
- コメント情報
  コメントの文字列
- 参照情報
  参照箇所の位置情報，参照されている関数のクロスリファレンス情報へのリンク
- マクロ本体情報
  マクロの本体の文字列
- マクロ引数情報
  マクロ引数の文字列
- 型情報
  型名，型
- 位置情報
  CX-Model へのリンク，ソースファイル名，行番号

4.3 CX-model とクロスリファレンス情報の対応

CX-model とクロスリファレンス情報の各要素に付けられる全ての ID は共通した ID を利用している．CX-model のインスタンスは `<fileId>.xml` という名前のファイルに書き込まれる．`fileId` は個々のファイルを識別するための ID である．クロスリファレンス情報との対応はこれを利用し，クロスリファレンス情報の CX-model へのリンクは `fileId` とそのファイルの中のどの要素へのリンクかを属性値 `fileId` と `refId` によって表現している．

この相互の対応関係を利用し，XSLT を用いて HTML に変換する事により，我々の研究グループで開発された SPIE[9] と同じ機能を持つソースプログラムブラウザを実現する事ができる．XSLT で変換した HTML を表示したブラウザの画面を図 1 に示した．本ツールの実装は SPIE[9] とは異なり，XML から XSLT により，HTML を生成しているため，XSLT を変える事により，図 2 のような見た目にする事も容易にできる．

図 1，2 の右のフレームがクロスリファレンス情報を示している．このフレームの中のソースコードへのリンク箇所をクリックする事で，ソースコードの該当箇所を参照する事ができる．図 3 にその画面を示す．

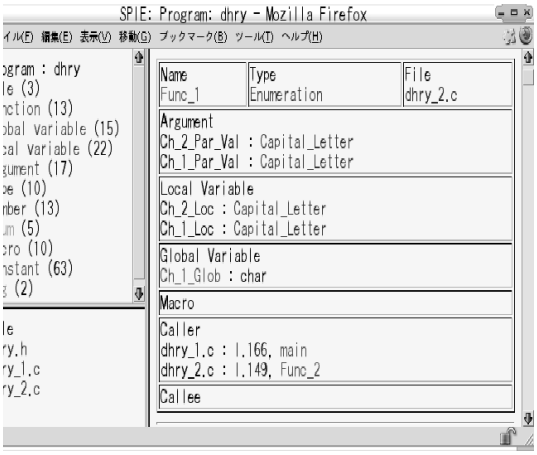

図 1: SPIE と同じスタイルの出力

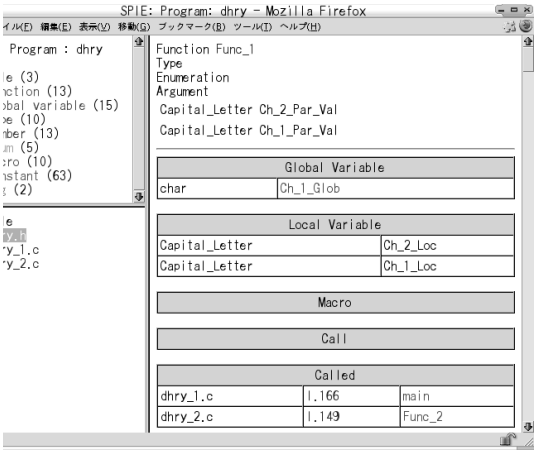

図 2: javadoc と同じスタイルの出力

5 XML リポジトリを用いたコード検索

5.1 コード検索

クロスリファレンス情報は，ファイル，関数，大域変数，局所変数，関数の引数，型定義，構造体のメンバ，列挙子，マクロ，定数，タグそれぞれに対して生成されるため，それぞれを区別して検索する事が可能である．また，変数が左辺値として参照されているか，右辺値として参照されているか，左右同時参照か，アドレス参照値かの区別を参照情報として生成されるため，これらを区別する事も可能である．

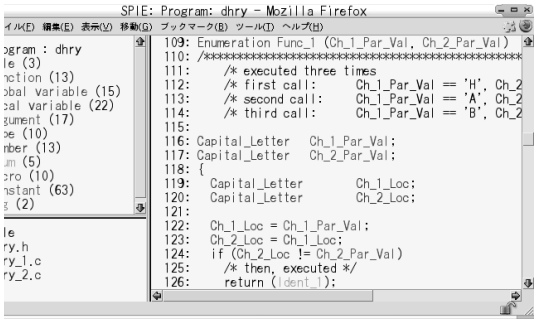

図 3: ソースコードを参照した画面

ANSI C では左辺値を，“オブジェクトとは名前付きのメモリ領域であり，左辺値はオブジェクトを参照する式である．左辺値式の明らかな例は適当な型と記憶クラスを持つ識別子である”，と定義している．しかし，ここでの定義は異なるので，下記に示す．

- 左辺値
  代入式の左辺に現われる変数
- 右辺値
  代入式の右辺に現われる変数
- 左右同時参照値
  “a++” のように値と記憶領域を同時に参照する式
- アドレス参照値
  記憶領域を表わす演算子 “&” を持つ式

これらの検索は Eclipse の検索機能で実現されているが，クロスリファレンス情報を用いる事により，このような検索に加えて下記の検索を行う事ができる．

- 大域変数，局所変数，関数の引数，構造体のメンバ
  - 型を指定した検索
- 型定義，タグ
  - ある型のメンバを持つ要素の検索
- 関数
  - 返り値の型を指定した検索
  - 引数の型を指定した検索

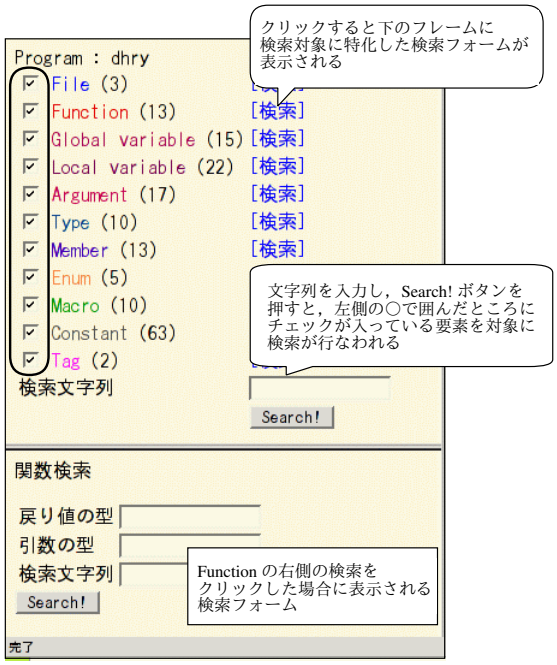

図 4: SPIE に組み込んだ検索 form

また，クロスリファレンス情報として生成される情報は，検索するまでもなく，容易に参照する事が可能である．このような検索機能を提供する事により，コーディングや保守時のソースコードの理解する際の手助けとなる．

5.2 コード検索とソースプログラムブラウザ SPIE

クロスリファレンス情報を利用したコード検索機能をソースプログラムブラウザ SPIE に組み込む事で，ソースコードの理解支援ツールの 1 機能として実現した．SPIE の左側に検索用のリンクおよび検索フォームを用意し，文字列を入力して “Search!” ボタンを押すと検索が行われる．図 4 は，SPIE の左側のフレームである．

まず，各プログラム要素名が書かれた上側のフレームについて説明する．プログラム要素名の左側のチェックボックスは，検索対象とするか否かのチェックである．デフォルトではチェックが入っており，全ての要素が検索の対象となる．検索文字列のところに文字列を入力し，“Search!” ボタンを押すと，検索が行われ，結果が下のフレームに表示される．この検索は部分マッチであり，一部でもマッチすると，結果として表示される．

図 4 には表示されていないが，左辺値，右辺値，左右同時参照値，アドレス参照値の 4 項目に対してチェックボックスが用意され，チェックされているもののみが結果として得られる．

また，前節で述べた型を指定した検索は，個々のプログラム要素に対して検索方法が変わるため，別途検索フォームを用意した．この検索は，プログラム要素の右側に書かれた “検索” と書かれたところをクリックする事によって，下側のフレームに検索対象に特化した検索フォームが表示される．図 4 には，Function の右側の ” 検索” をクリックした場合を示している．

実際に検索した結果を図 5 に示した．

検索対象は dhrystone-2.1 のソースコードで，検索に用いた文字列は “Arr” である．プログラム要素全てに対して，左辺値，右辺値，左右同時参照値，アドレス参照値全てを対象とした検索結果である．検索結果は各プログラム要素毎に表示され，プログラム要素名に続いて，マッチした結果が表示される．図 5 では検索語部分にアンダラインを引いているが，実際には，色で検索語を示している．

また，マッチしたソースコード中の要素を文字列として出力するだけでなく，リンクによって，実際のソースコードを参照できるようになっている．さらに，宣言箇所だけではなく，使用箇所も結果として出力し，ソースコードへのリンクが生成される．図 5 のハイフンに続いて書かれているのが使用箇所に当たる．

6 おわりに

我々は高度なコード検索機能として，プログラム要素間の関係を利用したコード検索を実装した．本研究では，ソフトウェアリポジトリとして，ソースコードに構文情報を付加した CX-model とクロスリファレンス情報を XML により生成し，クロスリファレンス情報を基に検索を行う事によって，プログラム要素間の関係を考慮した検索を容易に行う事ができた．実際に，ソフトウェアリポジトリを利用して，XSLT を用いてソースプログラムブラウザを作成し，コード検索機能を付加した．

今後の課題を以下に示す．

- 検索機能の充実

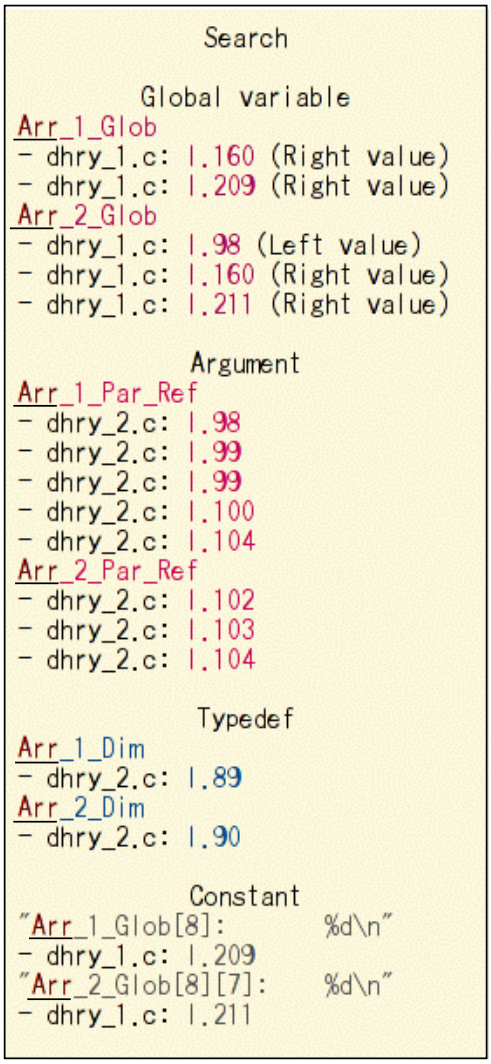

図 5: 検索結果

- XML 記述によるソフトウェアリポジトリの他の CASE ツールへの適用
- 他の言語への適用

謝辞

本研究は 2005 年度南山大学パッヘ研究奨励金 II-A の研究助成のを受けて行いました．ここに記して感謝致します．

参考文献

[1] Sapid home page. http://www.sapid.org/.

[2] G. J. Badros. Javaml: A markup language for java source code. *Proceedings of 9th international World Wide Web conference on Computer networks*, 2000.

[3] R. I. Bull, A. Trevors, A. J. Malton, and M. W. Godfrey. Semantic grep: Regular expressions + relational abstraction. *Proceedings of 9th Working Conference on Reverse Engineering*, 2002.

[4] N. Fukuyasu, S. Yamamoto, and K. Agusa. An evolution framework based on fine grained repository. *Proceedings of International Workshop on Principles of Software Evolution*, pp. 43–47, 1999.

[5] H. Kawashima and K. Gondow. Experience with ansi c markup language for a cross-referencers. *Proceedings of Domain-Specific Language Minitrack, 36th Hawaii International Conference on System Sciences.*

[6] 吉田, 山本, 阿草. Xml を用いた汎用的な細粒度ソフトウェアリポジトリの実装. 情報処理学会論文誌, 44(6), 2003.

[7] 吉田, 山本, 阿草. 宣言的なプログラム解析が可能な rdf に基づく細粒度ソフトウェアリポジトリ. 情報処理学会研究報告 ソフトウェア工学, 2005(147 (5)), 2005.

[8] 高林. gonzui: ソースコード検索エンジン. *http://gonzui.sourceforge.net/*.

[9] 大橋, 山本, 阿草. 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. 電子情報通信学会ソフトウェアサイエンス研究会, 98(28), 1998.