社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－SE－159 (18)
2008／3／18

仕様書とJavaソースコードの構造の類似性に基づく対応付け

田原貴光† 林 晋平† 佐伯元司†

ソフトウェア開発を効率よく行うために，仕様書とソースコードは互いに対応付いていなければならない．本稿では，自然言語で書かれた仕様書とJavaソースコードの対応付けを行う手法を提案する．ソースコードを文書の一種とみなし，単語の類似性により対応付く箇所を探す．提案手法では，仕様書の章構造とソースコードから抽出できる構造の類似性を用いて対応付けを行うことにより，精度の向上をはかる．適用事例を通して，本手法の有用性を確認した．

Recovering Traceability Links Between a Specification Document and a Source Code Using Their Structural Similarities

Takamitsu Tahara,† Shinpei Hayashi† and Motoshi Saeki†

The specification document and the source code of a software project have to be traceable each other for developing the software effectively. In this paper, we propose a technique how to recover the traceability links between a specification written in a natural language and a Java source code. In our approach, we consider a source code as a kind of a document, and detect the parts of a specification corresponding to the source code fragment using a similarity of word occurrences. Furthermore, to improve the detection precision, we use the similarity of the document structures of the specification and the source code. Through a case study, we validate the feasibility of our approach.

1. はじめに

ソフトウェア開発の各工程では，要求仕様書や設計書，ソースコードやテスト仕様書など様々な文書が作成されるが，それらの文書は互いに関連する部分を参照できなければならない．開発中には，要求変更や設計変更などが頻繁に起こり，それに伴い修正が波及するすべての文書やコードを修正する必要がある．それぞれの文書やコードの関連する部分が対応付いていることで，頻繁に変更される仕様に対して，他の文書やコードの修正箇所を容易に特定できる．また実装と仕様の間の矛盾検出や一貫性の検証も容易に行える．

しかし，文書やソースコード間の関連が失われているプロジェクトが多数存在する．ある文書に変更が生じたときに関連する他の文書やコードの各部分の修正をすること無しに，片方の文書だけを修正してしまうことで，互いの関連が失われてしまうことが多い．

よって，ソフトウェア開発中に生産される仕様書やソースコードの間の関連を効率よく復元する手法が望まれる[1)~4)]．作成済みの対応付けの行われていない文書間に対してその対応付けを行い，その間の追跡性を確保することで，容易に関連箇所の修正や検証を行うことができる．しかし，対応付けを行う作業は一般にコストがかかる．なぜなら，ある文書に含まれる項目に対して対応付く箇所を探すためには，他のすべての文書のすべての項目と比較し，対応付く箇所を探さなければならないためである．よって計算機による自動的な対応付け方法が望まれる．

ソースコードと仕様書の両方を文書の一種と考え，それぞれが含む単語の類似性に基づいた対応付けが行われているが[1)~3)]，精度の改善が望まれる．両文書中の単語の表現の違いに対応するために，識別子の分解や同義語の考慮，翻訳などが用いられるが，それにより異なる文脈で出現する，同一の表現に置き換えられた箇所が対応付いてしまい誤検出が生じる．また，適切な単語が文書中に含まれていないために未検出が生じてしまう．

本研究では，仕様書とソースコードに注目し，この二つの文書間の対応付けを自動的に行う手法を提案する．仕様書の文章に含まれる単語とソースコード上に

† 東京工業大学 大学院情報理工学研究科 計算工学専攻
Department of Computer Science, Graduate school of Information Science and Engineering, Tokyo Institute of Technology

存在する単語の類似性を利用するとともに，精度よく対応付けを行うために，仕様書の章構造とそれに対応するソースコードから抽出した構造の類似性を用いて精度を上げる方法を提案する．

本稿の以降の構成を示す．2 節では単語の類似性による対応付けの問題点とその解決方法のアイデアを述べる．3 節では提案手法について述べ，4 節で，具体例としてユーザインタフェース記述に対する手法の適用方法を述べる．5 節で適用事例とその評価について述べ，6 節で関連研究を紹介し，7 節で本稿をまとめ，今後の課題を述べる．

2. 対応付け精度向上のアイデア

2.1 単語の類似性による対応付けの問題点

文書間の関連する箇所を対応付けるためには，それぞれの文書に出現する単語の類似性が利用される．同じ事柄について書かれた二つの文書の中で，同じ単語が含まれている箇所があれば，それらの項目は同じ内容について述べていると考えられるため，対応付く候補とすることができる．

単語の類似性により対応付ける場合，単語の表現の違いに対応するために，識別子の分解や同義語の考慮，翻訳などが用いられる．両文書中で，同じ内容にもかかわらず異なる表現を用いて書かれる場合がある．特に自然言語で書かれた文書とソースコードなど離れた文書では，表現が異なることが多い．

ユーザインタフェース記述を例に説明する．構造的に書かれている仕様書のユーザインタフェース記述を図 1 の (A)，画面を構築する部分のコードを図 1 の (B) とする．

このとき単語の類似性から，仕様書の「3.1.1.1.4. 操作メニュー」を実装している箇所が，コードのコメントにより，変数 *opMenu* に格納された JMenu のオブジェクトであるとわかる．一方，「3.1.1.1.4.1. ファイルの再生」は，実際に対応すべき箇所である変数 *playItem* に格納された JMenuItem のオブジェクトに，「再生」という単語が含まれていないため対応付けることができない．そこで，識別子名 *playItem* がラクダ形式☆ で書かれていることに注目して分解することで，**play** と **item** に分けることができる．また，英単語を翻訳することで **play** から「再生」という単語を得ることができるため，仕様書の「3.1.1.1.4.1. ファイルの再生」と対応付けることができる．

☆ 複数の単語で識別子名を構成させるとき，各単語の頭文字を大文字にしたものを結合して表現する．

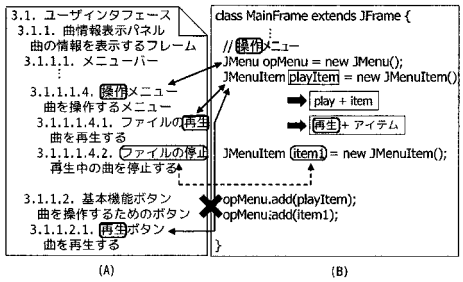

図 1 単語の類似性による対応付け

しかし，表現の違いを吸収して対応付きやすくすることにより問題が生じる．識別子の分解や翻訳により，変数 *playItem* に格納されたオブジェクトを表す単語「再生」を得ることができたが，それにより，仕様書中の「再生」という単語が含まれる箇所すべてが対応付けの候補となってしまう．例えば，図 1 の (A) において「3.1.1.2.1. 再生ボタン」にも「再生」という単語が含まれているため，単語の類似性により，「再生ボタン」と *playItem* が対応付くことで誤検出となる．

また，単語を抽出したときに，その部分の特徴を表すような単語が含まれていないと，対応付けることができない．「3.1.1.1.4.2. ファイルの停止」と対応すべきコードが，変数 *item1* に格納された **JMenuItem** のオブジェクトであるとき，コードの修正時に適切な識別子やコメントを書かなかったために，単語が類似せず対応付けることができずに，未対応となってしまう．

2.2 構造を用いた対応付け精度の向上

仕様書の章構造とソースコードから抽出できる構造の類似性を用いることで，対応付けの精度を向上させることができる．図 1 の (A) の仕様書を見ると，「3.1.1.1.4. 操作メニュー」の節の中に，二つの節，「3.1.1.1.4.1. ファイルの再生」と「3.1.1.1.4.2. ファイルの停止」があることは，この二つの画面要素がメニューバーの中に配置されることを示唆している．よって，「操作メニュー」を親，「ファイルの再生」と「ファイルの停止」を子とする親子関係を構造として得ることができる．一方，図 1 の (B) のコードを見ると，二つのメソッド呼び出し *opMenu.add*(*playItem*)，*opMenu.add*(*item1*) から，*opMenu* に二つの画面要素，*playItem* と *item1* が配置されていることがわかる．このことから，*opMenu* を親，*playItem* と *item1* を子とする包含関係を構造として得ることができる．このとき二つの構造の親要素「操作メニュー」と *opMenu* が対応付いていることで，同じ子要素である「ファイルの再生」と *playItem* を対応付けることができ，異

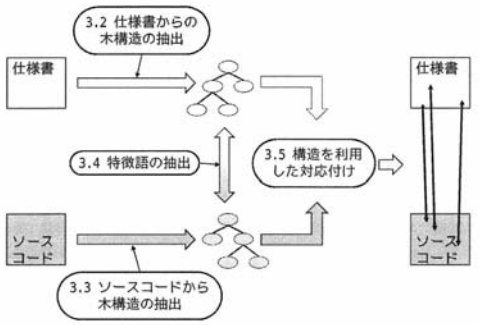

図 2 手法概要

なる構造にある「再生ボタン」との対応付けを排除することができる．また，対応付く構造同士で，単語の類似性では対応付かなかった箇所同士を対応付けることで，「ファイルの停止」と *item1* を対応付けることができる．

3. 提案手法

3.1 概　　要

本研究では，仕様書に含まれる単語とソースコード上に含まれる単語の類似性と，それぞれから抽出できる構造の類似性から，対応付けを行う手法を提案する．まず，仕様書とコードからそれぞれ木構造を抽出する．各ノードには，対応付けに利用される単語として，そのノードの特徴を良く表している単語を特徴語として抽出，付加する．そして，特徴語の類似性を用いて対応付けるときに，それぞれの構造の類似性から対応付く箇所の候補を限定していくことで，精度を向上させる．手法の概要を図 2 に示す．図中の番号は，本稿の節番号を表す．以降にそれぞれの手順毎に説明する．

3.2 仕様書から木構造の抽出

対象とする仕様書として，自然言語で書かれた章構造を持つ文書を入力とする．例えば IEEE830[5] では，ソフトウェア要求仕様書に書くべき内容を定義している．章ごとに記述する内容が定められていることで，注目する章により機能要求やユーザインタフェース記述，非機能要求といったソフトウェアの特定の側面に対する情報を得ることができる．

注目する内容が書かれた章から，木構造を抽出する．章構造で書かれた仕様書において，対応付けたい情報が書かれた章に対して，節の見出しをノードとして抽出する．そして，節の入れ子関係を，木構造の親子関係となるように木構造を構築することができる．

3.3 ソースコードから木構造の抽出

ソースコードからは，仕様書から抽出した木構造に対応するように木構造を抽出する．ソースコードは，開発するアプリケーションを構築する記述の集まりである．一つの仕様を実現するために，通常複数の処理が行われるため，コードは仕様書に比べ，より粒度の細かいものとなる．よって，コードを静的解析し，仕様書から抽出した木構造に合うように情報を取捨選択して構造を抽出する．

木構造のノードには，そのノードを表すものとして，識別子名やコメントを保持させる．識別子は，プログラムの中でクラス名やフィールド名などを表すために付けられる名前である．識別子名は主にアルファベットや数字を用いて構成される．そのためプログラマは識別子名として，それを読んだだけで対象とするオブジェクトが何を表しているかをわかるように，対象をよく表す単語を用いて構成することが多い．また，コメントは主に，そのコードを読む人が理解できるように書かれる注釈であるため，コードの周辺にあるコメントはそのコードの意図を表している．よって，識別子名やコメントがノードの特徴を表す．

3.4 特徴語の抽出

仕様書とソースコードから抽出した木構造の各ノードに対して，そのノードの特徴を良く表す単語として特徴語を加加する．それぞれのノードには，節の見出しや，識別子やコメントなどをそのノード特有の情報として付加した．そこで，それらの文に含まれる単語を抽出することで特徴語を得ることができる．単語を抽出するために，茶筌[6] を用いて各文を形態素解析し，得られた名詞や動詞をそのノードの特徴語とする．

3.5 構造を利用した対応付け

抽出したそれぞれの木構造の類似性を利用して，ノードの対応付けを行う．仕様書からは章構造の見出しをノードとして木構造を構築し，その構造に対応するように，ソースコードから木構造を抽出した．そのような類似した木構造では，ある二つのノードに関連があり対応付いた場合，そのそれぞれの子孫ノード同士もお互いに対応付きやすいといえる．そこで，トップダウンに木構造の上位のノードから対応付けていき，対応する部分木を検索していく．さらにその部分木の中のより下位な部分木の対応を検索していくことで，下位のノードの対応候補を限定し，誤検出や未検出を減らしていく．

トップダウンによる木構造の対応付けアルゴリズムを図 3 に示す．手続き match の初期入力には，それぞれの木構造のルートノードを与える．それぞれのノードの集合の考えられる組み合わせを combination で求め，その中で最も類似する組み合

```
1  Procedure: match(⟨S, C⟩)
2    S − 仕様書側の木構造のノードの集合
3    C − コード側の木構造のノードの集合
4
5    if (|S| == 1) {
6      if (children(S) == ∅) // 葉ノード
7        link(S, desc(C))
8      else
9        match(⟨children(S), C⟩)
10     return
11   }
12   if (|C| == 1) {
13     if (children(C) == ∅) // 葉ノード
14       link(C, desc(S))
15     else
16       match(⟨S, children(C)⟩)
17     return
18   }
19
20   maxSimilarity = −1
21   highSimilarityCombination = null
22
23   for comb in combination(S, C) {
24     similarity = sim_c(comb)
25
26     if (maxSimilarity < similarity) {
27       maxSimilarity = similarity
28       highSimilarityCombination = comb
29     }
30   }
31
32   for pair in highSimilarityCombination {
33     match(pair)
34   }
35
36 Procedure: link(n,N)
37   Nの中でnと類似するノードを対応付ける
38
39 Function definition:
40   children(n) ::= ノードnの子要素の集合
41   desc(n) ::= ノードnの子孫ノードの集合
42   combination(N1,N2) ::= 組み合わせの集合
43   sim_c(c) ::= 組み合わせcの類似値
```

図 3 構造を利用した対応付けアルゴリズム

わせを求める（23–30 行目）．例えば，match の入力で，仕様書側のノード集合が $S = \{s_1, s_2\}$，コード側のノード集合が $C = \{c_1, c_2\}$ のとき，組み合わせは combination=$\{\{\langle\{s_1\},\{c_1\}\rangle,\langle\{s_2\},\{c_2\}\rangle\}, \{\langle\{s_1\},\{c_2\}\rangle,\langle\{s_2\},\{c_1\}\rangle\}\}$ と，二通り考えられる．$\{\langle\{s_1\},\{c_1\}\rangle,\langle\{s_2\},\{c_2\}\rangle\}$ は s_1 と c_1 が，s_2 と c_2 がそれぞれ対応することを表す．このとき組み合わせの類似値をそれぞれ $\mathrm{sim_c}$ で計算し，正しい組み合わせとして，より類似している組み合わせを選ぶ．ノードの集合の要素数が一個になった場合（5, 12 行目），そのノードの子ノードに対して match を行う（9, 16 行目）．葉ノードに到達した場合は，もう一方の部分木に含まれるノードの中で一番類似するノードを探し，対応付ける（7, 14 行目）．combination を以下に示す．

$$\mathrm{combination}(S, C) = \Big\{\langle S_1, C_1\rangle, \ldots, \langle S_n, C_n\rangle \Big| \forall n \in \mathbb{N},\ 1 \le \forall i \le n\ S_i \neq \emptyset, C_i \neq \emptyset, \sum_j S_j = S, \sum_j C_j = C\Big\}.$$

ここで $\sum_j S_j$，$\sum_j C_j$ はそれぞれ S，C の直和分割を表す．

組み合わせの類似値計算は，部分木の類似値 $\mathrm{sim_t}$ を用いて以下のように定義される．

$$\mathrm{sim_c}(\langle S_1, C_1\rangle, \ldots \langle S_n, C_n\rangle) = \prod_i \mathrm{sim_t}(\langle S_i, C_i\rangle).$$

例えば，組み合わせ $\{\langle\{s_1\},\{c_1\}\rangle,\langle\{s_2\},\{c_2\}\rangle\}$ は，$\mathrm{sim_t}(\langle\{s_1\},\{c_1\}\rangle)\times\mathrm{sim_t}(\langle\{s_2\},\{c_2\}\rangle)$ となる．すなわち，対応付く部分木同士の類似値 $\mathrm{sim_t}$ がどれも大きいほど，その組み合わせはより正しい対応関係であることを表す．

また，部分木の類似値 $\mathrm{sim_t}$ は以下のように計算する．

$$\mathrm{sim_t}(S, C) = \frac{|S_{match}|}{|S|} \times \frac{|C_{match}|}{|C|}.$$

ここで $|S|$，$|C|$ はそれぞれ部分木に含まれるノードの数を表し，$|S_{match}|$，$|C_{match}|$ はそれぞれ，一方の部分木に含まれるノードで，他方の部分木に対応付くノードの数を表す．木構造の類似性から，もし対応する木構造の部分木同士ならば，そこに含まれる子孫ノードは互いに対応付きやすいといえる．そこで $\mathrm{sim_t}$ は，それぞれの対応付く割合を計算している．これは互いに対応するノードの割合が多いほど，二つの部分木は同じ構造を表していることを表す．

そして，$\mathrm{sim_t}$ のそれぞれの割合を求めるとき及び link で類似するノードを探すときに，二つのノードが対応付くかどうかを判断する方法として，特徴語を利用して類似値を計算する．その計算方法として二つの基準を設けた．

(1) 仕様書側のノードに含まれる特徴語が，ソースコード側のノードに含まれる特徴語に部分一致する割合

(2) 仕様書側のノードに含まれる特徴語を単名詞に分割した単語が，ソースコード側のノードに含まれる特徴語に部分一致する割合

通常は (1) で計算する．仕様書側のノードに含まれる特徴語がより多く含まれるほど，その二つのノードは対応関係があると判断する．しかし，表現の違いに対応するために辞書などを用いてもうまく対応付かない時がある．このとき，二つの木構造は類似することから，多くのノードは対応付くはずである．よって，そのような場合でも対応する箇所を探すために，(2) を用いて，類似するノードを探す．

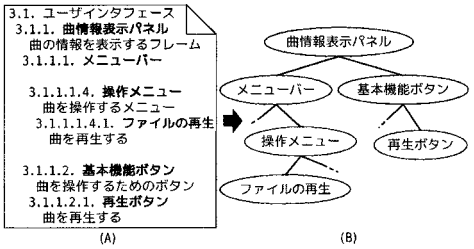

図 4　入力となる仕様書と抽出した木構造

4. ユーザインタフェース記述における対応付け

4.1 概　　要

具体例として，ユーザインタフェース記述における対応付け方法について説明する．仕様書には，顧客からの様々な要求がまとめられて記述され，その中には機能要求や非機能要求，外部インタフェースなど，ソフトウェアで実現するために必要な情報が含まれる．章構造を持つ仕様書では，記述する内容ごとに章を設けて記述される．よって，具体的に対応付けを行う場合は，章ごとにコードから構造を抽出する方法を考えなければならない．そこで本節では，IEEE 830[5] の 2 章 1 節「製品の概要」および 3 章「具体的な要求の説明」の中で記述すべきであるとされている，ユーザインタフェース記述に注目し，その構造を抽出する方法を述べる．

4.2 仕様書からのユーザインタフェース記述の抽出

ソフトウェア仕様書に書かれるユーザインタフェース記述のうち，アプリケーション画面のレイアウト定義を入力とする．仕様としてユーザインタフェースを記述する場合，例えば図を利用するなど，記述方法はプロジェクトにより異なる．そこで本研究では自動で木構造の抽出を行うために，画面のレイアウトに対して章構造を用いて記述されている仕様書を対象にする．各見出しに画面の構成要素を記述し，その説明を文章で記述したもので，ある画面が他の画面の要素となる場合は，それがわかるように節の入れ子で表現されたものとする．例えば図 4 の (A) ように記述されているとすると，「操作メニュー」の中の要素として「ファイルの再生」という要素が存在することを表す．

画面のレイアウト定義から木構造を構築する．3.2 節で述べたように，各見出しをノードとして抽出し，節の入れ子関係を親子関係として木構造を構築する．図 4 の (A) から木構造を抽出すると，(B) のようになる．

続いて各ノードに対する特徴語を抽出する．3.4 節で述べたように，各ノードに含まれる見出しから名詞や動詞を抽出する．例えば「ファイルの再生」からは，「ファイル」と「再生」を特徴語として得られる．

4.3 ソースコードからの GUI 構造の抽出

仕様書から抽出した木構造に合わせて，ソースコードから木構造を構築する．仕様書の木構造は，各ノードが画面の構成要素となる構造を持つ．構造の類似性を利用した対応付けを行うために，ソースコードからは，GUI を構成するためのそれぞれのオブジェクトがノードとなるような木構造を抽出すればよい．

Java プログラムでは，GUI を構成するために提供されているライブラリを用いて GUI アプリケーションを構築する．そのライブラリの特徴を利用することで，必要な情報を，静的解析をすることにより抽出することができる．木構造を構築するにあたり必要な要素として，次のように GUI 構成要素とその親子関係を構築するメソッドに分けられる．

- GUI 構成要素の一般化
  - パネル型（フレーム型，レイアウト型）
  - ボタン型
- GUI 構成要素の親子関係の定義
  - 親.method(子) 型
  - 子.method(親) 型
  - method(親, 子) 型
  - 親 (子) 型
  - 子 (親) 型

例えば親.method(子) 型は，あるメソッド method を呼んだオブジェクトが親となり，引数で指定されたオブジェクトが子となるような親子関係を構築するメソッドを表す．

以上の各要素に対応するライブラリの部品をソースコード上から探し出すことで，木構造を構築することができる．例えば Swing ライブラリ[7] の場合，**JFrame** や **JDialog** などがフレーム型，**JPanel** や **JMenuBar** などがレイアウト型，**JButton** や **JMenuItem** などがボタン型となる．また，*javax.swing.Component* クラスの持つメソッド *add* は，例えば *panel.add(button)* と呼ぶことにより *panel* の中に *button* を配置するためのものであり，*panel* が親，*button* が子となるような親子関係を構築する．よって *add* は親.method(子) 型となる．構築された木構造の例を図 5 に示す．図中のコードにおいて，太字で書かれたクラス名は GUI 構成要素を表し，下線が引かれたメソッドは GUI 構成要素の親子関係を定義するメソッドを表す．

続いて，木構造の各ノードに対する特徴語を抽出す

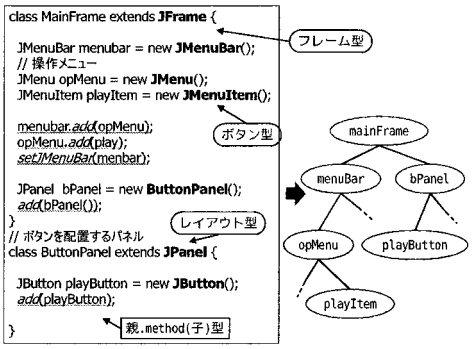

図 5　ソースコードと抽出した木構造

る．3.3 節で述べたように，コードから利用できる情報として識別子名とコメントがある．特に GUI オブジェクトに関連する要素として，以下のものが利用できる．

- インスタンス化されたオブジェクトを格納する変数名
- インスタンス化されたオブジェクトのクラス名
- コンストラクタの文字リテラル引数
- 変数宣言の直前のコメント
- インスタンス化された箇所の直前のコメント
- インスタンス化されたクラスのコメント

これらをそれぞれのノードに付加させる．

3.4 節で述べたように，各ノードに含まれる識別子やコメントから名詞や動詞を抽出する．例えば *opMenu* の場合，識別子名の *opMenu* の他にオブジェクトが生成されている直前のコメントから，「操作」と「メニュー」が得られる．

5. 適用事例

5.1 目的と問題設定

本研究の有用性を確認するために，二つの事例に対して仕様書とソースコードの対応付けを行い，その精度の向上を確認した．事例は，我々の研究室で作成したミュージックプレーヤー（以下 MP）と，オープンソースのスケジュール管理ソフト，JWorkSheet[8] である．各アプリケーションは表 1 の通りである．また評価方法として以下の値を測定した．

再現率（recall） ソフトウェア仕様書の項目 n 箇所中，正しくソースコード側と対応付いた箇所 m の割合 $m/n \times 100(\%)$ を表す．

精度（precision） 対応付いた箇所 p のうち，正しく対応付いた箇所 q の割合 $q/p \times 100(\%)$ を表す．

単語の類似性から対応する箇所を探すときに，なるべく対応付きやすくするために以下のような処理を行った．

- 識別子がラクダ形式で書かれているとして分解
- 辞書を利用して同義語を考慮
- 辞書を利用して英語を日本語に翻訳

上記の処理の有無を切り替えて本研究の提案手法を適用する．

表 1　事例に用いたアプリケーションの内訳

		MP	JWorkSheet
仕様書	言語	日本語	英語
	抽出したノード数	46	40
コード	ステップ数	2500	4800
	抽出したノード数	50	51

5.2 実　　装

事例の評価を行うために，提案手法を実現するツールを Java で実装した．本ツールは，4.2 節で述べたような章構造を持つ画面のレイアウト定義が書かれた仕様書と，仕様書を元に作成されたソースコードを入力とし，仕様書の各見出しに対するコード上の対応するオブジェクトの出現箇所を XML 形式で出力する．コードの静的解析は，Eclipse[9] の JDT を利用した．

5.3 結　　果

各評価結果は表 2 のようになった．表中の「構造なし」は構造の類似性を使用した処理を行わなかったことを表し，「処理なし」は特徴語に対する識別子の分解や辞書の利用などの処理を行わなかったことを表す．

本手法の構造の類似性を用いずに対応付けたとき，特徴語に対する処理の有無で精度は，MP の場合 20.67 から 20.83，JWorkSheet の場合 13.33 から 27.00 となり，あまり大きく変わっていない．一方，JWorkSheet の場合，特徴語に対する処理の有無で再現率が 5.00 から 87.50 に大きく上がっている．MP は，コード中にコメントが豊富に含まれており，適切な特徴語を多く抽出できたため，特徴語に対する処理を行わなくても再現率が 80.43 と大きな値を示した．

特徴語に対する処理を行った場合，構造を用いた対応付けをすることで，精度は MP の場合，76.08 から 64.40 と減少したのに対し，JWorkSheet では 14.28 から 90.24 と上昇した．

特徴語に対する処理を行った場合，精度は構造を用いることで，MP の場合 20.83 から 64.40，JWorkSheet の場合 27.00 から 90.24 と大きく上昇した．再現率は構造を用いると，MP の場合 84.78 から 80.43，JWorkSheet の場合 87.50 から 82.50 とわずかに減少した．

表 2 実行結果

		MP		JWorkSheet	
		構造なし	構造あり	構造なし	構造あり
精度	処理なし	20.76	76.08	13.33	14.28
	処理あり	20.83	64.40	27.00	90.24
再現率	処理なし	80.43	76.08	5.00	5.00
	処理あり	84.78	80.43	87.50	82.50

5.4 評価・考察

本手法を用いることにより，対応付けの精度を上げることができた．まず，特徴語の類似性判定時に対応しやすくするために，辞書や単語の分解などを用いた場合，取りこぼしを少なくし，より広く対応付けができる．結果から特徴語に対して対応付きやすく処理を行うことで，確かに再現率が上昇していることがわかる．このとき，本手法による構造の類似性による対応付けを用いない場合に比べ，用いた場合は，特徴語に類似性判定時の処理をすることで向上した再現率はわずかに減少しているが，精度が大幅に上昇していることがわかる．これにより構造の類似性を用いることで，再現率の減少を抑えつつ精度を向上させることができるといえる．

一方，本手法による構造の類似性を用いた場合に，特徴語の類似性判定時における処理の有無による精度の変化は，二つの事例で異なる結果を示した．ここから，単語に対する処理はアプリケーション毎に効果が異なり，目的に応じて使い分けなければならないことがわかる．

6. 関連研究

6.1 上流工程で生産される仕様書同士の対応付け

Alex らは，二つの仕様書のそれぞれの項目に対して対応する組み合わせを探す手法を提案し，ツール RETRO を実装した[2),3)]．このツールでは，対応付けの工程全てを自動化することはできないとして手動による処理も採用し，最終的には人間が対応付けを決定する．この手法では，二つの仕様書から抽出した単語から，情報検索技術を用いて対応する箇所を探す．さらに，本研究と同様に汎用辞書を用いることで表記のゆれに対応している．分析者は自動的に対応付いた候補を見て，間違った対応付けを取り除き，漏れた対応付けを追加する．修正された情報をもとに Rocchio のレレバンスフィードバック[10)] を用いて検索のためのベクトルを更新し，再び類似度計算を行い対応付けの候補を見つける．以上のステップを繰り返すことで対応付けを更新し，その精度を向上させる．

この研究では我々の研究と違い，要求レベルの仕様書同士を対象とし，自然言語で書かれた文書同士を対応付ける．そのため，ある程度単語が含まれていることから情報検索技術が使える．精度を向上させるために我々は構造に注目したが，Alex らは手動による処理を採用している．

6.2 仕様書とソースコードとの対応付け

Zhang らは，オントロジを用いてソースコードと仕様書を対応付ける手法を提案している[4)]．記述論理[11)] を用いることで，あらかじめ定義されている規則に加えユーザが規則を追加することで，知りたい情報を推論により得ることができる．オントロジはソースコード用のものとドキュメント用のものの 2 種類を用意しておき，あらかじめ二つのオントロジの意味的に対応するクラスやプロパティを対応付けておく．ソースコードのフィールドやメソッドといった構文要素をソースコード用オントロジと対応付け，仕様書から抽出した名詞や動詞をドキュメント用オントロジと対応付けることで，推論により仕様書とソースコードの対応する部分を見つけることができる．

Zhao らは要求仕様のうち機能要求に注目し，それとソースコード上の関数群を対応付ける手法を提案している[1)]．この手法では，TF-IDF と cosine 類似度から機能要求に対してソースコードの関数ごとに類似度の高いものを探し，ソースコードから構築したコールグラフを用いて，探し出した関数から呼ばれる関数を集めることで，足らない関数を補完する．

これらの研究では，ソースコードと他の文書を対応付ける手法を提案している．我々の手法に Zhang らの用いたオントロジを組み合わせることで，特徴語の対応付け精度の向上が期待できる．また，本研究はユーザインタフェース記述に対して対応付けを行うため，Zhao らの機能要求に対する対応付けと組み合わせることで，仕様書の様々な記述内容に対して精度よく対応付けることができる．

6.3 構造の対応付け手法

XML をはじめとする構造化文書では，同文書の別バージョンとの違いを明らかにするための差分検出法が多く提案されている．

Lee らは，構造化文書の二つのバージョン間の編集距離を効率的に求める手法を提案している[12)]．まず，編集前の構造化文書の葉ノードが編集後の構造化文書のどのノードに対応するかを求め，そこから親ノードを辿りながら，親ノード同士を対応付ける．続いて得られた対応付け関係から，幅優先で編集前の文書の各ノードがどこに移動や複写，または削除されたのかを探索していくことで，二つの文書間の編集距離を求める．

木構造は構造化文書として書き表すことができるため，この編集距離による二つの構造の差異から類似性をはかることができると考えられ，提案手法の構造を利用した対応付けに応用することができる．

7. おわりに

本研究では，ソフトウェア要求仕様書とソースコードの対応付けを，それぞれに含まれる単語の類似性と構造の類似性に注目して行う手法を提案した．仕様書の注目する内容によりコードから抽出する構造が異なるため，その一つであるユーザインタフェース記述における構造の抽出方法を示した．また，適用事例を通して本研究の有用性を確認した．

今後の課題を以下に示す．

Interactive なツールの実装　適用事例の評価からわかるように，アプリケーション毎に単語の類似性判定時における適切な処理が異なる．目的により簡単に使い分けられるよう適宜切り替えの出来るツールが望まれる．また事例の評価に用いたツールは，ソフトウェア仕様書とソースコードの対応付けを行うものであり，対応付いた結果は XML ファイルとして出力しているだけである．実際に対応付けを行った場合は，その結果から修正箇所を特定したり，要求がすべて実装されているかなどを検証したりする作業が行われる．そのような作業を効率的に行えるように，結果のグラフィカルな表示による視覚的な支援が必要になる．

仕様書の各内容に対する対応付け　提案手法では，具体的な構造の抽出方法としてユーザインタフェース記述を例に示した．しかしソフトウェア仕様書には，ユーザインタフェース以外にも機能要求や非機能要求などがあり，それぞれ対応付ける必要がある．ユーザインタフェース以外の項目でも，それぞれの項目ごとに適切な構造を抽出することで，対応付けの精度を上げることができると考える．

適用事例の追加　さらに実験を通して本研究の有用性を評価する必要がある．本研究では，有用性を確かめるために二つの事例を用いて評価した．しかし，それだけでは十分ではなく，さらに様々なオープンソースソフトウェアに対して適用し，提案手法により対応付けの精度が向上することを確認する．

参考文献

1) Zhao, W., Zhang, L., Liu, Y., Sun, J. and Yang, F.: SNIAFL: Towards a Static Non-Interactive Approach to Feature Location, *Proc. 26th Int, Conf. on Softw Eng.*, pp.293–303 (2004).
2) Hayes, J.H., Dekhtyar, A., Sundaram, S.K. and Howard, S.: Helping Analysts Trace Requirements: An Objective Look, *Proc. 12th Requirements Engineering Conference*, pp. 249–259 (2004).
3) Dekhtyar, A.: Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods, *IEEE Trans. Softw Eng*, Vol.32, No.1, pp.4–19 (2006).
4) Zhang, Y., Witte, R., Rilling, J. and Haarslev, V.: An Ontology-based Approach for the Recovery of Traceability Links, *Proc. 3rd Int, Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering* (2006).
5) IEEE: IEEE Recommended Practice for Software Requirements Specifications (1998). IEEE Standard 830.
6) 松本裕治, 北内啓, 山下達雄, 平野善隆, 松田寛, 浅原正幸：日本語形態素解析システム chasen「茶筌」. http://chasen-legacy.sourceforge.jp/.
7) Robinson, M. and Vorobiev, P.: *Swing*, Manning Publications Co. (2002).
8) Paul Ponec: jWorkSheet - the time tracker. http://jworksheet.pponec.net/.
9) The Eclipse Foundation: Eclipse.org home. http://www.eclipse.org/.
10) Baeza-Yates, R.A. and Ribeiro-Neto, B.: *Modern Information Retrieval*, Addison-Wesley (1999).
11) Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D. and Patel-Schneider, P.F.: *The description logic handbook: theory, implementation, and applications*, Cambridge University Press (2003).
12) Lee, K.-H., Choy, Y.-C. and Cho, S.-B.: An Efficient Algorithm to Compute Differences between Structured Documents, *IEEE Trans. Knowl. and Data Eng.*, Vol.16, No.8, pp.965–979 (2004).