Vol.2014-SE-184 No.10
Vol.2014-EMB-33 No.10
2014/5/20

Refinement Calculus に基づく 用語辞書からのフォーマルなモデル構成

大森 洋一[1]　荒木 啓二郎[1]

概要：フォーマルな用語辞書は，通常の用語辞書に加えて，自然言語によるキーフレーズとそのキーフレーズのフォーマルメソッドを用いた意味定義の写像を与える．用語辞書を用いることにより，ソフトウェア開発プロジェクトにおいて各キーフレーズの意味の一貫性をフォーマルメソッドを用いて保証可能となり，さらに同じドメインにおいてキーフレーズとその意味定義の集合が再利用可能なドメイン知識となる．しかし，保守工程や新規のプロジェクトにおいて，既存の用語辞書を再利用して，仕様としてのフォーマルなモデルを記述するする場合，環境の違いによりキーフレーズの意味に一貫性があるとは限らない．本研究では，用語辞書を数理的に定義し，用語辞書に含まれる各キーフレーズのフォーマルな制約に基づいて，意味の一貫性が保たれるための条件を明らかにする．このために，用語辞書におけるフォーマルな要素の意味論，不整合が発生する条件を Refinement Calculus で定義し，そうした不整合を起こさない枠組みを提案する．

キーワード：フォーマルメソッド, 自然言語, 仕様記述, Refinement Calculus

Formal Model Generation from Requirement Dictionary Based on Refinement Calculus

Abstract: A formal requirement dictionary extends existing requirement dictionary to give the map of the key phrase by natural language and its semantics definition using the mathematical logic. The consistency of the semantics of each key phrase can be guaranteed in a software development project by using a formal requirement dictionary. The dictionary can be reused as a set key phrases and corresponding formal definitions in the same domain. However, the semantics of the key phrase in the reused dictionary, which had been constructed during the developing phase of a software project, does not necessarily have consistency in the different environments, such as in maintenance phase or a new project. This paper formally defines the formal requirement dictionary and clarifies the conditions for the consistent semantics of the key phrases by their formal restrictions. We propose that definitions of the semantics of formal elements in the requirement dictionary and the conditions for inconsistency based on Refinement Calculus, and a framework which does not cause such inconsistency.

Keywords: Formal method, Natural Language, Specification Description, Refinement Calculus

1. はじめに

フォーマルメソッドは，計算機システムの仕様を数理的に表記することおよび，その仕様を設計の検証基盤として利用する手法である [9]．フォーマルメソッドを適用し，ソフトウェアの仕様を数理的なモデルとして記述することにより，その数理体系における矛盾や曖昧さを取り除くことが可能となる．このため，特に上流工程からのソフトウェア品質向上に有用であることが知られており，提案初期の頃から利用されてきたプラント制御や交通管制といったクリティカルシステムだけでなく，さまざまな応用分野，適用範囲において多くの適用例がある [8]．フォーマルメソッドによる仕様は，よい仕様に求められる性質のうち，いくつかを自然に満たす．

[1] 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu Univ ersity

すなわち，

a. 要求が正しく記述されていること
b. 記述の解釈に曖昧さを持たず，簡潔であること
c. 満たすべき要求が全て含まれており，それ以外が含まれないこと
d. 内部に矛盾を含まないこと
e. 要求の優先順位が明確であること
f. 仕様の検証が可能であること
g. 修正が容易であること
h. 仕様を満たしているかどうかの検証が容易であること

に対して [6]，次のようになる．

B. 数学的な概念は文化，時代，地域によらず，誰でも同じ意味に解釈できる (上記 b)
D. 背景となる数理体系において，論理的な矛盾があればそれらを検出できる (上記 d)
F. プログラムの意味論が定義され，仕様のツールによる検証が可能である (上記 f)
G. 修正の意味が明確であり，かつ修正の影響する範囲が明確である (上記 g)
H. 数理的なモデルから有限状態機械への変換を経由して，プログラムが仕様を満たすことの証明が可能である (上記 h)

ただし，要求抽出および要求整理に関係する残りの a, c, e といった性質については，フォーマルメソッドを適用して仕様を記述するだけでは解決できないので，すべての関係者による確認と合意が必要となる．

フォーマルメソッドによる仕様記述は，記述言語の習得が必要である，対象について背景となる数理的な知識と洞察が必要とされるといった理由により，その効果は理解されながらも，ソフトウェア開発における一般的な手法とはなっていない．つまり，特に開発の初期段階では自然言語による仕様が用いられており，それ以降の開発工程においても計算機の専門家以外のユーザを含む多くの関係者により共有されるのは，自然言語による仕様である．上記の a, c, e といった性質の確認にも自然言語による仕様が用いられることになる．

しかし，自然言語による仕様には，曖昧さや矛盾が含まれることが多く，それらを解消するには入念なレビューと書き直しが必要であり，なおかつ完全な保証を与えることは難しい．したがって，フォーマルメソッドを活用するためには，自然言語による仕様記述とフォーマルメソッドによる仕様記述に何らかの対応を与える必要がある．

我々は，このような仕様記述の使い分けとそれらの間での用語辞書による一貫性確保を達成するための基本的な手順を，次のように提示している [15].

1. 自然言語による仕様をフォーマルメソッドを用いたモデルへ変換し，その上で数理的な検証を行う．
2. フォーマルなモデルの上で矛盾や記述不足が見つかった場合，それを自然言語記述にフィードバックする．
3. 修正された自然言語による仕様に 1.-3. の手順を繰り返す．
4. モデル上の問題がなくなったら，自然言語による仕様を関係者がレビューし，妥当性を確認する．
5. 自然言語による仕様に含まれる用語とそれに対応するフォーマルメソッドによる定義を用語辞書として保存する．
6. レビューにより修正された自然言語による仕様を修正し，フォーマルメソッドを用いたモデルへ変換し，1.-6. の手順を繰り返す．
7. 再利用性を高めるために，複数の製品開発で得られる用語辞書の融合，分割，更新など保守管理を行う．
8. 同じドメインの問題に対して，用語辞書を再利用する．

図 1 フォーマルな用語辞書を使った仕様記述手順
Fig. 1 Specification Procedure with the Formal Requirement Dictionary

これは，一般的な要求のトレーサビリティの問題であり，フォーマルメソッドを利用することにより生じる問題ではないので，フォーマルメソッドを用いない場合であっても，仕様とプログラムのトレーサビリティを確保するために，さまざまな方法が提案されている．第 2.1 節で，代表的な方法をいくつか挙げる．

本研究では，このようなトレーサビリティ保証にフォーマルな用語辞書を用いる手法について，フォーマルメソッドにおける Refinement Calculus[2] を用いて定式化し，保証のための条件を明らかにする．Refinement Calculus は，ソフトウェア開発における充足関係を記述するための束論理に基づく数理体系であり，単調な述語について完備である．ソフトウェア開発の充足関係は，後工程の生成物が前工程の仕様を満たし，かつ前工程の生成物が後工程の仕様ていることが，必要十分条件になるので， Refinement Calculus により検証できる．フォーマルなモデルとそのふるまいについての理論的な基盤となっている Unifying Theory of Programming も Refinement Calculus を用いており，本稿ではそれをフォーマルな用語辞書も含めた関係に拡張する．

以下，本稿の構成は，第 2 章で，トレーサビリティの観点からフォーマルな用語辞書を規定し，ツールによる具体的な実装について述べ，既存の研究と比較する．第 3 章では，フォーマルな用語辞書，フォーマルなモデル，モデルの上で検証される性質を Refinement Calculus を用いて整理し，それらの関係について考察する．さらに，第 4 章ではトレーサビリティと再利用の観点から辞書の役割を検討し，第 5 章でまとめを行う．

2. 写像としての辞書

ソフトウェア開発にけるトレーサビリティの保証は

- プログラム理解
- ソフトウェア保守
- 要求の充足確認
- ソフトウェア修正時の影響確認
- ソフトウェア再利用

といった観点で良い影響がある．一般に，ソフトウェアのトレーサビリティとして，以下の 2 点が挙げられる [7].

(1) 製品に関するトレーサビリティ ...ソフトウェアアイテム（ソフトウェアの部品の最小構成単位）が、製品にどのように組み込まれているかを追跡できること

(2) 文書及びデータに関するトレーサビリティ ...ソフトウェアの要件が製品のどの部分で実現されているかを追跡できること，また逆にできあがっているソフトウェアの構成部品（実行モジュール等）が，どのソフトウェア要件を実現するためのものかを追跡できること

本研究ではこのうち (2) を対象とする．

トレーサビリティの保証が十分でないソフトウェア開発では，要求が一部でも変更されると，対応する後工程の中間生成物をすべて見直さなくてはならない．このような状態では，開発期間がいくらあっても十分でなく，ソフトウェア資産の再利用もままならない．

2.1 関連研究

ファイルより小さい単位でトレーサビリティを保証するための最も直接的な方法は，要求項目から対応する仕様記述の部分へ，さらに設計や実装の該当部分へといったリンクを設定する手法であるが，プログラムが大きくなるとこの方法はすぐに人間による管理が困難となり破綻する．最も野心的な研究は，このようなリンクを自動的に管理することを目指して，自然言語処理などの知能処理技術を応用し，コンピュータ上のツールにより，プログラムあるいは自然言語による仕様の解釈を行う方法である．Antoniol らは，プログラムおよび自然言語仕様をインデックス化し，確率的な関連モデルを適用することで検索する手法を提案している [1]．また，大量の識別子からプログラム理解に重要なキーワードを効率的に発見する手法も検討されている [13]，Ozkaya らは，ツール上での操作をすべて観測することにより，要求モデルと設計モデルの一貫性を保証する環境を提案している [17]．これらの研究は，性能向上のめざましいコンピュータを活用し，トレーサビリティ保証の大幅な自動化を目指すものであるが，現時点ではまだ実用段階にはない．

現在のソフトウェア開発で最も多く用いられているのは Traceability Matrix (TM) を用いる方法である．TM は，仕様と実装といった，比較対象となる 2 種類の中間生成物に対して，縦軸と横軸にそれぞれの管理単位となる項目を並べ，相互に関係のある部分をチェックする多対多の関係を表現した表である [12]．TM にはいくつものバリエーションがあり，複数の表を仕様に関して合成し，正規化したものや [21]，さらに階層化したものなどがある [26]．特に，システム要求と結合テストのように，直接的な対応づけが難しい場合に有用であり，一覧性に優れる．商用の要求管理ツールは， TM を複数の表に関して管理できるよう拡張したり，処理の自動化や高速化を工夫したものが多い [5][3][24]．ただし，要求の変更に対応した TM の更新および妥当性の確認は人間による作業が必要となる．

また， Bug Tracking System (BTS) をソフトウェア開発段階で利用する手法も提案されている [25][28]．特に開発と保守を区別しないアジャイル開発において，こうした手法が利用される場合が多い．アジャイル開発とフォーマルメソッドの併用は大きな研究課題であるが，ある程度の規模を超える開発や入札案件などコミットメントが求められる契約に基づいた開発では，事前に仕様を決める必要がある．BTS では，仕様の個別の項目に相当するチケット単位でのトレーサビリティ保証は行えるが，複数の項目に跨る仕様やソフトウェア全体を通じた一貫性の保証は難しい．

本研究では，フォーマルメソッドの利点を生かしつつ，現在の TM による要求管理よりも効率的な手法を検討する．

2.2 ツールによる実装

フォーマルなモデル記述においては，我々が開発したドメイン用語辞書を作成する作業をサポートする辞書管理ツールを利用した [14]．このツールは 図 2 のような外観を持ち，

- 自然言語による仕様書に含まれる登録した見出し語をマークする「仕様書マーカー」
- 見出し語にフォーマルな意味定義を与える「辞書編集」
- フォーマルに検証可能なフォーマットを生成する「モデル出力」

の大きく 3 つの機能を提供する．これらの機能は，仕様記述者の作業を補助するものであり，自動化を目的とするものではない．

本研究では，フォーマルなモデルの記述には VDM++を採用した．VDM++ はフォーマルメソッドの一種である VDM を構成する仕様記述言語であり，一階述語論理に基づく意味論をもつ [11]．VDM はモデル規範型仕様記述言語と呼ばれ，対象の仕様をモデルの状態を記述していくことに重点をおいており，数学的な仕様から実装に近い仕様までさまざまな抽象度で記述可能である．

Vol.2014-SE-184 No.10
Vol.2014-EMB-33 No.10
2014/5/20

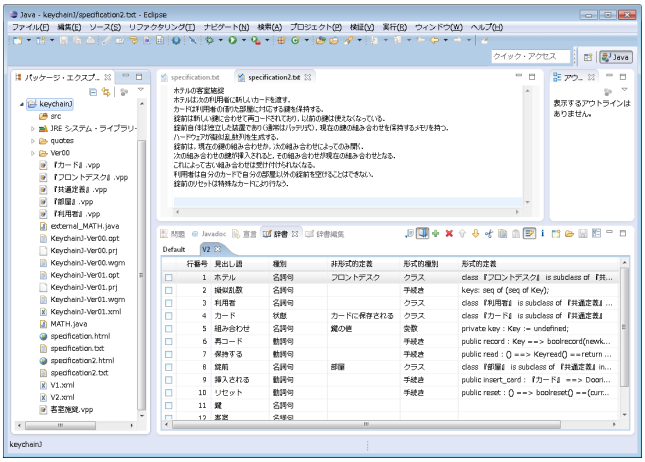

図 2 辞書ツール
Fig. 2 User Interface of Jack-o-Dict

VDM++ は宣言型の言語であり，明示的に指定しない限り，ファイル内の記述順には依存しない．また，クラスによるカプセル化とアクセス制御，継承関係，多態性などオブジェクト指向設計の記法を一部サポートしている．

本辞書管理ツールで実装したフォーマルな用語辞書のひとつのエントリは，見出し語，見出し語の品詞，自然言語による定義，フォーマルな種別情報，フォーマルな意味定義からなる．このうち，自然言語仕様に関連する部分は，見出し語，見出し語の品詞である．見出し語は空白文字も含む任意長の文字列であり，自然言語による仕様書に含まれる同じ語句をマークし，重要な概念の網羅性確認を容易にする．品詞は，名詞句，動詞句，状態を用意する[*1]．これは，日本語および英語のの独立語が名詞または動詞のいずれかであること，状態は英語における補語にあたることによる．見出し語の並びは任意であり，後述のモデル生成の場合のみ特定の順序が要求される．

自然言語による定義は，従来の用語辞書の定義に当たる要素であるが，フォーマルな用語辞書では純粋に人間のための補助的な情報であり，任意の情報を記入してよい．

フォーマルな種別情報，フォーマルな意味定義は，使用する仕様記述言語，今回の実装では VDM++ に依存している．フォーマルな種別情報は， VDM++ のセクションに相当し，クラス，型，定数，状態変数，関数，操作のいずれかである．フォーマルな意味定義は VDM++ による仕様記述そのものであり，意味をもつ最小単位である単文または複文である．VDM++ による意味論で扱えない，文より小さい単位は本研究では対象としない．

[*1] 現在のバージョンでは日本語と英語を想定している．

3. 辞書からのモデル生成

実装した辞書管理ツール自体は， VDM++ の検証機能をもっておらず，これらの情報を用いて，既存の VDM++ 検証ツールに入力可能なフォーマルなモデルを生成する．VDM++ のモデルは Public なクラスごとに，規定のフォーマットのファイルへ出力される．各エントリに書かれたフォーマルな意味定義は，対応する見出し語についてのものであるので，見出し語の並びを変え，ひとつの Public クラスおよびそのメンバの定義をまとめてひとつのファイルへ出力する．文法や型に関する整合性は， VDM の検証ツールである VDMTools[22] または Overture Tools[18] により機械的に検証できるので，レビューにおいて自然言語記述のような書き間違いや型の不整合を人間が確認する必要はない．図 1 の (6) に示すように，モデルで発見された不整合は仕様にフィードバックされる．

3.1 機能とモデル

Unifying Theory of Programming (UTP) は Tony Hoare と He Jifeng により提唱された表示的意味論，操作的意味論，代数的意味論を統一的にに扱う枠組みであり，フォーマルな仕様記述，設計，実装が意味論上は区別なく扱えることを示す [4]．UTP により， VDM で記述された仕様は，ごく一部の例外を除いて，仕様を満たすプログラムを Refinement 関係により導出できることが保証される．

VDM++ は一階述語論理を背景としているが，VDM++ で記述された仕様としてのモデルの提供する機能をふるまいとみなすことができる．すなわち， VDM++ は仕様を有限状態機械としてモデル化する．ある操作 op は前状態 s と後状態 s’ により， op = (s, s’) として定義される．このようなモデル記述を VDM では陰仕様と呼び， s を事前条件， s’ を事後条件と呼ぶ [10]．VDM はこうした操作を契約として解釈する [27]．つまり， op を状態遷移とみなし， 「s が満たされている時に op を呼び出すと，必ず s’ が満たされる」また「s が満たされていない時に op を呼び出すと結果は不定である」ことになる．VDM の操作以外の要素は，関数は操作の事前条件と事後条件が一致する特殊な場合として，その他の要素は個々に独立したデータ構造として定義される．

事前条件・事後条件は関数または操作に固有であるが，その他のデータについては，不変条件を考えることができる．不変条件 inv は型，状態変数，関数，操作についてモデル中で常に成り立つ性質である．状態変数 sv に対する不変条件 inv(sv) は，いつ評価しても真になる条件であり，型 T に対する不変条件は， T に属する変数 t について真になるすなわち $\forall\ t\ in\ T\ \&\ inv(t) = True$ である．

3.2 Refinement Calculus

フォーマルメソッドの提案初期から， Edsger Dijkstra や Niklaus Wirth による段階的詳細化手法が既に提案されており [20]，数理的な証明を用いて，フォーマルな仕様記述から実行可能なプログラムを演繹的に得る手順が知られていた．段階的詳細化の手順においては，各ステップで下位の仕様が上位の仕様を満たすことを証明する．この 2 項関係を Refinement $A \sqsubseteq B$ とよび， B が A を充足するという関係を表す．しかし，このような証明は数学的な訓練と知識を必要とし，誰にでもできるものではなかったので広く利用されるには至らなかった．つまり，形式手法の利点は理解されていたものの，適用コストの高さが普及を妨げる大きな要因であった．

Refinement Calculus[2] は，

Refinement Calculus では有限状態機械 S, S' ，状態 $s1, s2 \in S \wedge s1', s2' \in S'$ 操作 $op \in S \wedge op' \in S'$ に対して次の性質が成り立つ．

- $s1 \sqsubseteq s1' \Leftrightarrow s1 \supseteq s1'$
- $s1 \sqsubset s1' \Leftrightarrow s1 \supset s1'$
- $S \sqsubset S'$ のとき， $\forall sinS \sqsubseteq \exists s'inS'$
- $op = (s1, s2) \wedge op' = (s1', s2')$ のとき， $op \sqsubseteq op' \triangleq s1 \sqsubseteq s1' \wedge s2 \sqsubseteq s2'$
- $s \sqsubseteq s' \wedge s' \sqsubseteq s'' \Rightarrow s \sqsubseteq s''$ (推移律)

複数の操作に関しては，

図 3 にこれらの性質を図示する．

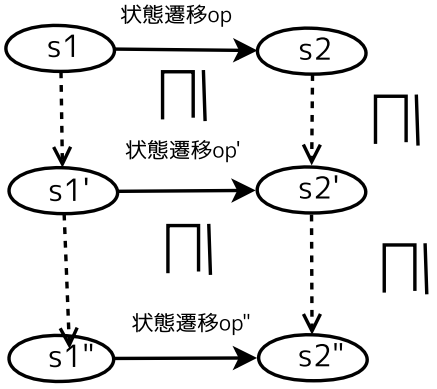

図 3 Refinement 関係
Fig. 3 Refinement Relation

4. 辞書の責務

フォーマルメソッドによる数理モデルは，自然言語による仕様に数理的な意味を与える辞書とみなすことができるという考え方は，目新しいものではない [16][19]．そして，極端な場合，フォーマルな用語辞書によって自然言語による仕様書全体からフォーマルなモデル全体への対応づけも可能である．しかし，このような粗い対応づけはトレーサビリティの保証や再利用の観点から意義が小さい．

逆に，フォーマルな用語辞書を用いることにより最小の粒度，すなわち，自然言語による仕様記述のうち，意味をもつ最小単位と，そのフォーマルな定義を対応づけることにより，これ以上ない高い精度で対応関係を保持することができる．たとえば，「りんご」と「それ以外」を選別するシステムでは「りんご」が意味をもつ最小単位であり，「赤いりんご」と「青いりんご」を選別するシステムでは，「赤いりんご」「青いりんご」が意味をもつ最小単位となる．このように，対象となるソフトウェアの達成目標により識別されるべき語句のパターンは異なり，文法的な処理だけでは不十分である．

4.1 辞書の定義

フォーマルな用語辞書は，従来の用語辞書のように自然言語による仕様記述全体とフォーマルな仕様記述全体の対応関係を管理するのではなく，対応関係そのものを辞書の形で管理する．従来の手法では，仕様が複雑になり，どちらの記述が大きくなっても，対応関係の管理コストは両者の大きさの積に比例するが，本手法では，両者の対応関係そのものを管理するので，同じ語句が何度も現れる場合には管理コストは変わらない．

また，自然言語による仕様からフォーマルな仕様を対応させる正写像に対して，逆写像を定義することができる．これにより，フォーマルなモデルの不整合をモデル内で修正した場合にも，逆写像をたどることにより，元の自然言語による仕様の対応部分をみつけることが容易である．従来の自然言語による用語定義では，自然言語とプログラミング言語のセマンティック・ギャップが大きく，プログラムとの対応づけにプログラマの解釈が必要となるので，自然言語による仕様と対応するプログラム記述が 1 対多あるいは多対 1 の関係になってしまい，逆写像をうまく定義できない．もちろん，フォーマルな用語辞書においても，逆写像を定義するためには自然言語による仕様の最小意味単位とフォーマルな仕様における最小意味単位が対応付けられていなければならない．

キーフレーズ KPH は次のように定義される．

$$KPH \triangleq regexp(CHR) \tag{1}$$

ここで， CHR は自然言語で許される空白文字含む文字集合, regexp() はある文字集合上の，任意の文字列を含む正規表現で得られる全ての文字列を返す関数である．

品詞 POS は，名詞句 noun , 動詞句 verv, 状態 state のいずれかである．

$$POS \triangleq \{noun, verb, state\} \tag{2}$$

フォーマルな意味は， VDM++ のセクションのいずれかである種別 SEC

$$SEC \triangleq \{class, types, values, instance\ variables, functions, operations\} \quad (3)$$

および言語マニュアル [23] に規定される VDM++ の文 VPP により定義される．

つまり，本研究のフォーマルな用語辞書は，写像

$$\{(KPH, POS) \rightarrow (SEC, VPP)\} \quad (4)$$

として定義される．つまり，フォーマルな用語辞書では，品詞が異なれば異なる定義を与えられるので，同じ語句であっても，名詞であるクラスと，動詞であるコンストラクタに別の定義を与えるといったことができる．

ここで，フォーマルな用語辞書におけるフォーマルな意味定義 VPP が操作 op の場合を考える．第 3.1 節で見たように，関数は操作の特殊な場合であり，その他の要素はほぼ自明だからである．op の事前条件を P, 事後条件を Q とした時， VPP は op = (P, Q) をひとつのエントリに保持する．ここで，Bool 型の自由変数 free に対して，

$$(P, Q) \triangleq (free \wedge P) \Rightarrow (free' \wedge Q) \quad (5)$$

となるように VPP を定義した場合，

$$(P1, Q1) \Rightarrow (P2, Q2) \equiv [P2 \Rightarrow P1] \wedge [P2 \& Q1 \Rightarrow Q2] \quad (6)$$

$$(P1, Q1) \equiv (P2, Q2) \Leftrightarrow [(P1, Q1) \Rightarrow (P2, Q2)] \wedge [(P2, Q2) \Rightarrow (P1, Q1)] \quad (7)$$

となる．

したがって，まず，フォーマルな用語辞書のエントリを再利用する場合，自由変数 free と P および Q に対してそれぞれの変数名が独立していなければならない．VDM++ では操作は明示的に状態変数を読み書きするので，これらの状態変数の名前空間を保護する必要がある．さらに，

$$(P1, Q1) \vee (P2, Q2) \triangleq (P1 \wedge P2, Q1 \vee Q2) \quad (8)$$

であるから， 複数の操作に対して，事前条件 P1 および P2 が衝突しないようにしなければならない．

このように，名前空間に対してフォーマルな用語辞書のモジュール化により再利用性を高めるために，辞書の先頭には，

1. 対象領域
2. 利用組織
3. 入力言語
4. 出力言語

をヘッダ情報として記述する．対象領域は，現実の問題および計算機の問題のいずれかのドメインであり，設計対象のみならず，文章検査など任意のドメインを識別する．また，同じドメインであっても，プロジェクトや組織によってモデル化の視点が異なる場合があるので，そうした観点の違いは利用組織によっても識別する．入力言語，出力言語は，それぞれ用語辞書の入力となる自然言語，フォーマルなモデルの記述言語である．複数の辞書を同時に利用する際には，適用する辞書には優先度をつけ，対象ドメインの優先度を表現できる．

ここで，用語辞書に含まれる入力言語の集合は，対象の自然言語全体に対して，特定の問題に対応したサブセットとみることができる．このとき，ある用語辞書を利用することは，文法を元と同じ自然言語，語彙をその辞書に含まれている範囲に特定した Domain Specific Language (DSL) に相当する．これを利用して，既存のプロジェクトから類似度の高いドメインの用語辞書を再利用することができる．

5. おわりに

本稿では，ソフトウェア開発における仕様の自然言語記述とフォーマルなモデルの対応づけを管理する，フォーマルな用語辞書を利用したモデルの生成およびフォーマルな用語辞書の再利用を可能とする条件を検討した．提案手法では，フォーマルな用語辞書を定式化し，写像としての有効性を評価した．この結果，

- フォーマルな用語辞書による意味の一貫性は，写像としての定義となり，写像をたどることでトレーサビリティを保証できる．
- フォーマルなモデルの保証できる範囲において，複数回の Refinement の適用も可能である．
- 操作の再利用は，状態変数や内部変数の符号の衝突が起こらないようにする．

といった利点があることがわかった．

謝辞 本研究で使用した辞書ツール開発にご協力いただいた吉村康晴氏をはじめ，九州ビジネス株式会社のみなさまに感謝する．本研究の一部は， JSPS 科研費 24220001 ，基盤研究 (S) 「アーキテクチャ指向形式手法に基づく高品質ソフトウェア開発法の提案と実用化」の成果による．

参考文献

[1] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D. and Merlo, E.: Recovering traceability links between code and documentation, *IEEE Transactions on Software Engineering*, Vol. 28, No. 10, pp. 970–983 (2002).
[2] Back, R.-J. and Wright, J.: *Refinement Calculus: A Systematic Introduction*, Springer (1998).
[3] Borland: Caliber, https://www.borland.com/products/caliber/.

[4] Hoare, C. and Jifeng, H.: *Unifying Theories of Programming*, Prentice-Hall (1998).
[5] IBM: Rational DOORS, http://www-06.ibm.com/software/jp/rational/products/doors/productline/.
[6] IEEE-830: Recommended Practice for Software Requirements Specifications, IEEE Std. 830-1998 (1998).
[7] IPA/SEC: トレーサビリティ確保におけるソフト開発データからの効果検証実施報告書, 技術報告, 独立行政法人 情報処理推進機構, http://www.ipa.go.jp/files/000026863.pdf (2013).
[8] IPA/SEC: 厳密な仕様記述における形式手法成功事例調査報告書，技術報告，独立行政法人情報処理推進機構，http://www.ipa.go.jp/sec/reports/20130125.html (2013).
[9] Jones, C. B.: Software Development based on Formal Methods, *Proceedings of the CRAI Workshop on Software Factories and Ada*, LNCS, Vol. 275, Springer-Verlag, pp. 153–172 (1987).
[10] Jones, C. B.: *Systematic Software Development using VDM*, Prentice-Hall, 2 edition (1990).
[11] Larsen, P. G.，Mukherjee, P.，Plat, N.，Verhoef, M.，Fitzgerald, J.，酒匂寛 (訳)：VDM++によるオブジェクト指向システムの高品質設計と検証，翔泳社 (2010).
[12] MacMillan, J. and Vosburgh, J. R.: Software Quality Indicators, Technical report, Defense Technical Information Center (1986).
[13] 大場　勝，権藤克彦：プログラム理解を支援するコンセプトキーワードの自動抽出法 ckTF/IDF 法の提案，情報処理学会論文誌， Vol. 48, No. 8, pp. 2596–2607 (2007).
[14] 大森洋一，荒木啓二郎：自然言語による仕様記述の形式モデルへの変換を利用した品実向上 に向けて，情報処理学会論文誌：プログラミング， Vol. 3, No. 5, pp. 18–28 (2010).
[15] 大森洋一，日下部茂，林　信宏，荒木啓二郎：ドメイン用語辞書の再利用に向けたグループ化，情報処理学会研究報告ソフトウェア工学研究会， Vol. 2013, No. 15, pp. 1–8 (2013).
[16] 大森洋一：分散ストレージの安全性検証，情報処理学会研究報告，2009-EMB-14, No. 3, pp. 1–8 (2009).
[17] Ozkaya, I. and Akin, O.: Tool support for computer-aided requirement traceability in architectural design: The case of DesignTrack, *Automation in Construction*, Vol. 16, No. 5, pp. 674–684 (2007).
[18] Project, O.: http://www.overturetool.org/.
[19] 佐原　伸：構造化日本語仕様書としての VDM 仕様, SEC ジャーナル， Vol. 7, No. 1, pp. 23–27 (2011).
[20] Wirth, N.: Program development by stepwise refinement, *Communication of ACM*, Vol. 14, No. 4, pp. 221–227 (1971).
[21] 清水吉男：[入門+実践] 要求を仕様化する技術・表現する技術，技術評論社 (2010).
[22] SCSK システムズ：http://vdmtools.jp/.
[23] SCSK システムズ：http://www.vdmtools.jp/uploads/manuals/langmanpp_a4J.pdf.
[24] スパークスシステムズジャパン：RaQuenst, http://www.raquest.jp/.
[25] 平鍋健児，野中郁次郎：アジャイル開発とスクラム顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント，翔泳社 (2013).
[26] 中西恒夫，久住憲嗣，福田　晃：派生開発からプロダクトライン開発への漸次的移行プロセス XDDP4SPL におけるコア資産管理手法，情報処理学会研究報告，Vol. 2013-EMB-28, No. 9, pp. 1–6 (2013).
[27] 中島　震：形式手法入門，オーム社 (2012).
[28] 小川明彦，阪井　誠：Redmine によるタスクマネジメント実践技法，翔泳社 (2012).