情報処理学会論文誌　プログラミング　Vol. 3　No. 5　18–28 (Dec. 2010)

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質向上に向けて

大森 洋一[†1]　荒木 啓二郎[†1]

形式的な論理に基づいたモデルベース開発は，上流工程からソフトウェア品質の向上に有用なさまざまな数理的検証を可能とする．しかし，システム開発における仕様記述のほとんどは自然言語により行われており，形式モデルへの変換が必要である．我々は，自然言語による要求あるいは仕様の記述から形式的なモデルの要素となるキーワードを抽出し，それらの関連を特定するという，よく知られた手順をより精密に定義し，この手順を効率的にサポートするツールを開発した．自然言語記述と形式モデルを対応づけることによって，解釈の一意性と記述の柔軟性を両立させることができる．開発したツールを活用した検証では，自然言語による仕様記述に対して，用語や表記のゆれといった言語的な品質から，記述不足や多義的な用語といった解釈に関する問題まで幅広く改善できることを示すとともに，より高度な自動変換に向けての課題を明らかにした．

Toward a Quality Improvement of Specifications in Natural Language Based on the Semi-equivalent Formal Models

Yoichi Omori[†1] and Keijiro Araki[†1]

Model based development methods supported by formal logics enable various mathematical verifications for the improvement of the software quality from the early stage of software development. Most specifications in system development are, however, described in natural languages, thus it is necessary to translate them into formal models to verify them. We precisely re-defined the translating procedure that keywords are extracted first and relations among them considered next, which is broadly well known, and construct a tool to support this procedure for efficient translation. The tool supports semi-equivalence of the descriptions in natural language and its translated formal model. The verification process of the specifications in a natural language with the translation tool can show various types of defects, from linguistic errors such as multiple notations of an object to semantic inconsistencies such as undescribed or conflicted concepts.

1. はじめに

ソフトウェア開発における仕様策定段階では，自然言語による記述が中心的な役割を果たしている．その理由として，自然言語の次のような特性があげられる．

(1) 弱い抽象化
対象システムの分析において，適切な名付けによる概念の識別は重要な手順である．自然言語による名付けは，概念を象徴するとともに，抽象度や実装に関してある程度の幅を許容する．たとえば，物理的にはさまざまな「把手」が存在しうるが,「把手」という名付けにより，対象の機能と属性をある程度限定しつつ，複数人で概念を緩やかに共有できる．

(2) 連想による洗練
分析の初期段階においては，必ずしも正確に対象システムを理解できるとは限らないが，ほとんどの場合，名付けられた概念は正確な対象の少なくとも一部を含む．自然言語の単語に付随する連想や多様な修飾語によって，必要な要素の追加や不要な要素の削除を進め仕様を洗練させることができる．

(3) 記述環境の充実
自然言語は社会でひろく用いられており，普遍的な環境および手段において記述が可能である．これにより，専用の道具を用いなくても仕様記述や関係者間の意思疎通が可能となる．

しかしながら，自然言語記述と計算機上で効率的に動作するプログラムには，抽象度に大きなギャップがあり，設計過程を通じてこれを埋めなければならない．こうした自然言語の特性は，設計を詳細化し仕様を確定する段階では，曖昧さを含む仕様による誤解された合意を生み，また安易な仕様変更の原因ともなる．現在，その作業の多くはソフトウェア技術者による，プログラム資産の読解やこれまでの経験といった個人能力に依存している．プログラミング環境は高度化してきているとはいえ，抽象度の高い自然言語記述を設計者が解釈し，いきなりプログラムを記述するような状況では，詳細化過程の検証は不可能であり，新規ソフトウェア技術者への技術の伝承も困難となる．

†1 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electric Engineering, Kyushu University

1.1　モデルベース開発

設計の初期段階からなんらかの観点に基づいたモデルを用いて，仕様記述と検証を行うモデルベース開発は，この自然言語記述と仕様のギャップを埋めるのに適している．モデルベース設計を用いる利点として，コンポーネント化による設計の再利用がある．これは，主として開発期間の短縮を目的として，プロダクトライン開発[9] などとの組合せにより，プログラムあるいはライブラリよりも高レベルの設計情報の再利用を目指すものである．プロダクトライン開発では，対象領域における複数回の開発における再利用によるコスト回収を前提として，ソフトウェアドメインあるいは問題ドメインをモデル化する．

また，早期の欠陥発見によるソフトウェア品質の向上があげられる[10),23)]．ここで欠陥とは仕様と設計の不一致を指し，モデル化により要求の記述不足や要求仕様に対する設計の不備を検証可能とする．古典的な V 字モデルでは，要求に対する不備の検出は統合テスト時に行われるので，不具合の修正コストが非常に高くなる．モデルベース設計では，構成時の検証が可能であり，比較的短いサイクルで不具合を修正できる．この場合，単一製品開発であってもモデルベース設計を採用する意義がある．

いずれの場合もモデルの信頼性が重要であり，検証手法が課題となっている．

1.2　フォーマルメソッドの導入

フォーマルメソッドは，数理的な概念に基づいて問題をモデル化する手法であり，モデルベース設計手法の一種として長い歴史を持つ．抽象的なモデルに対して数学的性質が検証可能という特長から，製品だけでなく中間生成物の検証もでき，これまでにもソフトウェア品質の向上を目的とした応用は数多い[3),4)]．

フォーマルメソッドの適用は，大きく記述と検証に分かれる．記述段階は対象とする問題を，注目する性質に沿ってモデル化する．最終的に実行可能なプログラムの生成を考慮する場合，計算機言語への対応のさせやすさから，集合論および述語論理に基づく機能記述あるいはプロセス代数に基づく振舞い記述が多く用いられる．前者には VDM[6)]，Z/B method[1)] など，後者には CSP[12)]，CafeOBJ[8)] などがある．検証段階ではそれぞれの論理系において検証すべき表明を与え，記述されたモデルがその表明を満たすことを確認する．検証手法は，数学的な証明以外に，モデル検査[5),20)]，Petri-net の利用[14)] などがある．

このように，形式的あるいは準形式的に定義された仕様記述言語による記述は曖昧さの排除による共通理解に有効であることが知られているが，その記述および検証にはコストがかかる．初期段階の修正は，非技術的な外部条件の変更やアルゴリズム選択など，緻密な検証が無駄になる場合も多いので，フォーマルメソッドの適用には工夫が必要である．

以下，2 章で本研究の目的と新規性，3 章で提案手法をサポートするツールについて，4 章で辞書からの形式モデル生成，5 章で事例研究，6 章でまとめを述べる．

2.　自然言語による仕様の改善

本研究では，仕様策定過程の全体を通じて，自然言語記述とフォーマルメソッドによる形式モデルを併用する手法を提案する．これにより，関係者の共通理解の得やすい自然言語記述と，厳密な検証に基づく安定したモデル化を可能とする形式モデルのそれぞれの長所を両立させる．すなわち，自然言語記述から形式モデルへ変換し，その形式モデル上で数理的な検証を行い，開発の初期段階からのモデル駆動開発を実現するとともに，検証結果のフィードバックによる自然言語記述の改善を繰り返し行い，ソフトウェア設計を詳細化する．

本研究で目的とする自然言語記述の品質向上は，関係者の十分な相互理解を可能とすることであり，そのために

(1)　必要な情報がすべて書かれている，
(2)　記述が統一されており，一貫性がある，
(3)　簡潔にまとまっており，冗長な要素が少ない，

といった性質が望まれる．形式モデルを用いる効果は主に，(2) と (3) に関連した無矛盾性の検証にある．(1) の完全性については，形式モデルだけでは検証できないが，ドメインモデルの再利用による記述漏れ対策について 3.2.2 項で述べる．

2.1　形式モデルとの対応づけ

形式モデルは対応する自然言語記述の解釈を与え，その意味は数理的に一意に定まるので，仕様の誤解による実装の不備を防ぐことができる．また，形式モデルの検証により矛盾がある部分や一意に解釈できない部分が明らかになるので，その部分に対応する自然言語記述を追加したり，書き換えたりして品質を向上させることができる．一般にこの対応づけは，必ずしも 1 対 1 にはならないので，自然言語記述と形式モデルの間は準同型となる．

図 1 に，本研究で想定する開発フェーズを示す．計算機科学の専門家でない対象ドメインの専門家や，経営者や一般ユーザなど非技術者などとのコミュニケーションは自然言語記述を中心に行う．この部分はこれまでのモデリング手法と同様である．これに対して，設計対象のモデルおよび検証対象の性質の記述には専門的な知識が必要であることから，設計者やプログラマとのコミュニケーションの中心となる．形式モデルを対象とした検証は，モデルが数理的な性質を満たしていることを数学的に証明する作業であり，これまでの手法のように経験に頼らなくても，モデルの問題点を発見することが可能となる．仕様確定後の設

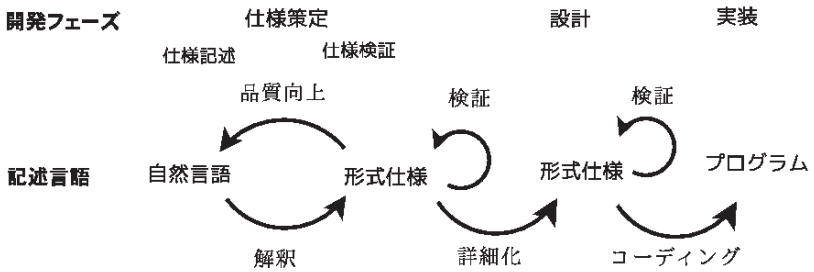

図 1　要求の洗練プロセス
Fig. 1　Supposed process of requirement refinement.

計・実装フェーズでは，形式仕様を参照すれば十分である．

このように複数の文書を用いる場合，それらの一貫性を保ちながら簡単かつ自動的に管理できるツールの利用が前提となる．手作業による変換は，変換ミスや修正の反映忘れといった問題を引き起こすからである．本研究では，自然言語記述と形式モデルの相互変換をサポートし，対応づけを辞書として管理するツールを開発した．

2.2 関 連 研 究

自然言語を用いた仕様記述については，高級言語が使われるようになった 1960 年代から提唱されているが，実用的な研究が始まったのは 1980 年代からである．

1980 年代の研究は，計算機言語を含む形式言語を自然言語へ対応づけることによって，自然言語による仕様に厳密な意味を与えるものが主流であった[19),21)]．

この方式の問題は，自然言語による仕様といいながら，実質的に形式言語で書くことになってしまうところである．形式言語を理解していないと不自然な自然言語の文法や用語の制限が多くみられ，識別子などを日本語で表現するだけで，計算機科学の非専門家が表現したい要求をモデルとして表現するのは困難であった．

1990 年代の研究は，日本語処理研究の進捗と計算機の演算能力の向上に従い，自然言語記述の機械的かつ正確な解釈を目標としている．

和田らは，仕様に書かれる日本語は

- 名詞のほとんどが業務用語である，
- 動詞や形容詞の種類は比較的少数であり，処理や動作を表現する，
- 数理的な比較や条件が記述される，

という特徴があり，若干の文法知識による補完で，ネットワーク型の構文木が構成できることを示した[18)]．

西田らは，構文解析に格フレーム解析を応用し，仕様のように語彙が少ない場合は，曖昧さの排除に特に有効であることを示した[22)]．

小林は，同じく格フレームによる構文解析を，限定した問題領域に応用し，係受けの関係に基づいた語義の曖昧さを解消するのに有効であることを明らかにした[17)]．

これらの方式は，自然言語記述の意味を一意に定め，計算機言語を含む形式言語に対応させる手法であり，ライブラリ開発やドメインを限定したモデリングで成果をあげている．しかし，人間同士でも困難な自然言語解釈の限界や，対象ドメインに直接関係しないところで開発環境が大がかりになることから一般的にはなっていない．

2000 年代に入り，オブジェクト指向設計の広まりにつれて，現実世界のモデルと計算機上の世界のモデルの一貫性が向上させられるようになった．この性質を利用して，加藤木らは日本語のサブセットによる要求記述から C++ 言語への変換を行う OOSF を開発した[26)]．OOSF は図 1 における分析，設計，実装のそれぞれの段階について，日本語ベースの言語を新規に開発し，自動的なプログラム生成を試みている．さらに OOSF を基に統一的な枠組みで扱うよう改良した OOJ が開発されている[25)]．

海外においても，たとえば英語に関しては，1990 年代に Meziane らが自然言語から形式仕様記述言語 VDM のデータ構造の抽出を[13)]，2000 年代には Harmain らがオブジェクト指向に基づくツール開発を行う[2)] など，同時期に同様の研究がみられる．

本研究がこれらの先行研究と異なるのは，次のような点である．

- 自然言語の重視
  先行研究のほとんどは，Saeki らの研究にみられるように，自然言語から形式モデル，あるいはプログラムへの詳細化は一方通行である[11)]．これに対して本研究では，1 章で述べた自然言語記述の利点に注目し，形式モデルの検証結果をそのまま関係者に提示するのではなく，自然言語記述の改善という形でフィードバックする．モデル検証者およびシステム設計者以外は，必要になるまで形式モデルを目にしなくてよい．
- 弱いモデル間の結合
  自然言語記述と形式モデルとの対応づけは強力なルールとして定義されたり，ツールに組み込まれたりしている．本研究では，辞書という形で自然言語記述と形式モデルのマッピングのみを管理する．このため，入力言語や出力モデルの変更は，特殊な場合を除いて辞書の交換で対応できる．また，3.2.2 項で述べるように，複数の辞書を使い分けることで複数のドメインに対応可能としている．

- 既存ツールとの連携
  本研究では，ツールが重要な役割を担っている．しかし，新規に作成したツールは最も上流となる自然言語記述と形式モデルの対応づけの部分だけであり，形式モデルは，たとえば VDM および VDMTools といった[15]，既存のフォーマルメソッドとそのツールを用いることで，プログラム生成や検証はそれらに任せている．本ツールそのものも Eclipse プラグインとして作成しており，他のツールとの連携も容易である．

2.3　研究の新規性

本研究のアイデアは，自然言語による仕様記述の品質改善を行うために，厳密な検証が可能となる形式モデルを用いる点と，自然言語と形式モデルの対応づけを辞書ツールにより併用する点の 2 つである．

自然言語による記述の問題は，書いている内容に矛盾があったり，不足があったりしてもそれを見つける方法がないことである．この理由は，自然言語は冗長な表現が可能だからである．感覚的な表現以外にも，この冗長性を利用して，繰返しにより表現形式を整えたり，言い換えにより部分的に強調したりといった技法が使われており，これが自然言語を用いた記述の分かりやすさの一因となっている．また，単語レベルでも同じ意味を複数の語彙で表現する同義語や，複数の意味を同じ単語で表現する多義語がある．

このため，記述品質改善のためにはレビューや厳密な手順に従ったインスペクションが一般的である．しかし，通常のインスペクションでは，仕様の品質はインスペクタの経験や能力によって設定される検査項目に依存する．本研究では，長い経験を必要とするインスペクションではなく，形式モデルによる検証を用いて，モデル内の矛盾を発見したり，数理的な性質を証明したりすることにより，仕様の確認や保障を行う．形式モデルで利用する数理学は，たとえば VDM の場合は集合論と一階述語論理，CSP の場合は状態機械とグラフ探索の基礎的な部分であり，関連した数学知識があれば，インスペクションなどの経験的な手法とと比較しても，習得は容易かつ短期間で可能である．これが最初のアイデアである．

さらに，形式モデルによる検証は，自然言語の曖昧さを排除し，厳密な仕様を表現する有力な方法として知られているが，自然言語による仕様記述は多くの関係者によって共有され，多くの変更や修正が加えられ，なかなか確定しないことが多い．このため，熟練した専門家により形式モデルへの変換を行って検証する間にも仕様に変更が加わり，その結果は最新の仕様と一致しないものとなって，結果的に無駄になってしまう場合がしばしばみられる．

したがって，本研究においてツールによる効率的な変換は本質的に必須である．これまでの研究では，モデル作成者の解釈に任せるのでなければ，2.2 節でみたように，ツールによる機械的な変換を可能とするために強い関連づけがあった．これに対して，本研究のツールで利用する辞書を介したマッピングは自然言語と形式モデルに弱い関連づけを与える．

本ツールでは，キーワードを辞書の見出し語として登録し，登録された単語はすべてマークする．辞書に見出し語の意味を登録する際に，複数の意味を定義する必要があれば，それは多義語である．また，同じ意味を別の単語に対して定義するのであれば，それらは同義語である．このように，多義語や同義語をみつけて避けることにより，自然言語記述の曖昧さを減らすことができる．また，既出の見出し語の辞書定義を参照することで，多義語になることを避けることもできる．自然言語の翻訳において，このように 1 度訳した単語を参照できるツールは「翻訳メモリ」として知られている．本ツールと翻訳メモリの違いは，翻訳メモリは文章から文章への変換であるため，対応が 1 対 1 であることと，先頭から順に記述される文章というデータ構造以外を想定していないところである．本ツールは出力が形式モデルとなるため，多対多の対応づけを考慮し，クラスなど構造化されたデータを出力するなど，より一般化したシステムとなっている．これが第 2 のアイデアである．

本研究における弱い関連づけによる新規性は，次のようなツールの特性としてまとめられている．

- トレーサビリティの確保
  辞書により，自然言語による見出し語と形式モデルによる形式的な定義の対応がつけられる．これが弱い関連づけであるのは，辞書に非形式定義と形式定義の項目があり，影響のおよばない範囲で，それぞれの記述の改善を辞書を介して独立して行えるからである．すなわち，自然言語記述が修正のうち，形式モデルの修正が必要となるのは，修正した部分の見出し語だけに限定できる．逆に，形式モデルの詳細化は，辞書におけるデータ定義の修正で可能である．
- 文法的情報からの分離
  見出し語は，文法的な品詞によらず，一連の語句も登録可能である．これは，修飾詞と自立語のセットでキーワードを構成する場合に対応するためである．辞書を用いた意味定義は，複数の語句から複雑なデータ構造への多対多の対応記述を可能とする．
  また本ツールでは，より複雑な正規表現によるパターンマッチを可能としており，複数の語句に関係した問題の発見を可能としている．これは，特に自然言語記述のアンチパターンを発見するのに有効であり，本ツールでは複数の辞書を個別に適用可能であることから，専用の辞書を用いて係り受けや表記のゆれといった自然言語記述の書き方自体に関する問題を除くことができる．

- 重複の容認
  多義性や同義性は基本的に抑制されるべきであるが，意図的な場合はそれらを明示的に辞書に登録できる．
  5 章におけるポットの例では,「湯」と「水」は同じものを指しており，形式定義は同一である．しかし，自然言語表現としては「60 度の水」よりは「60 度のお湯」が適切であり，分かりやすい．このように，意味を形式的に定義しても，自然言語表現をマッピングにより柔軟に対応づけることができる．

3. ツールの構成

本研究の想定する開発手順では，自然言語と形式モデルの相互変換を頻繁に行うことになる．変換は，仕様が未完成の状態でも行われるので，不足した情報を補完するための，設計者による仕様の確認，詳細化が作業の中心であるが，これを補助するツールを作成した．

このツールは，自然言語による仕様書から形式モデルへの変換を行う際に，問題記述の検査網羅度を確認するためにキーワードのマーキングを行い，マークしたキーワードの定義を記述する用語辞書およびキーワードの関連を構造化するデータ辞書の生成を効率的に補助し，形式的なモデルのテンプレートを生成する．

辞書には，マークした見出し語，見出し語の種別，非形式的定義，形式的定義，型などを登録する．見出し語と種別以外の情報は後から追加あるいは修正できる．見出し語を用いて，仕様書に含まれる同じ語を色をつけてマークし，キーワードの網羅度の確認を容易にする．作成した辞書をもとにして，形式的なモデルの変数，関数，手続きなどのテンプレートを生成する．仕様変更や記述追加は要求仕様書に対して行い，記述の高次化とツールの簡素化を実現する．

本ツールの利用者は，1 つ以上のファイルからなる要求仕様書を本ツールに読み込ませ，キーワードの網羅度を確認しながら形式モデルへの変換を行い，マーク付き仕様書と辞書を生成する．これに加え，必要に応じて，形式モデルを出力することもできる

本ツールは Eclipse プラグインとして実装しており，さまざまなパースペクティブと組み合わせることができる．**図 2** に示すツールの使用画面の例では，Java プロジェクト開発環境と組み合わせて使用している．

3.1 仕様書マーカ

本ツールは，自然言語記述から単語を辞書へ登録し，辞書の単語をマークする仕様書のマーカと，登録された単語を管理し，形式モデルへの変換を行う辞書ツールからなる．

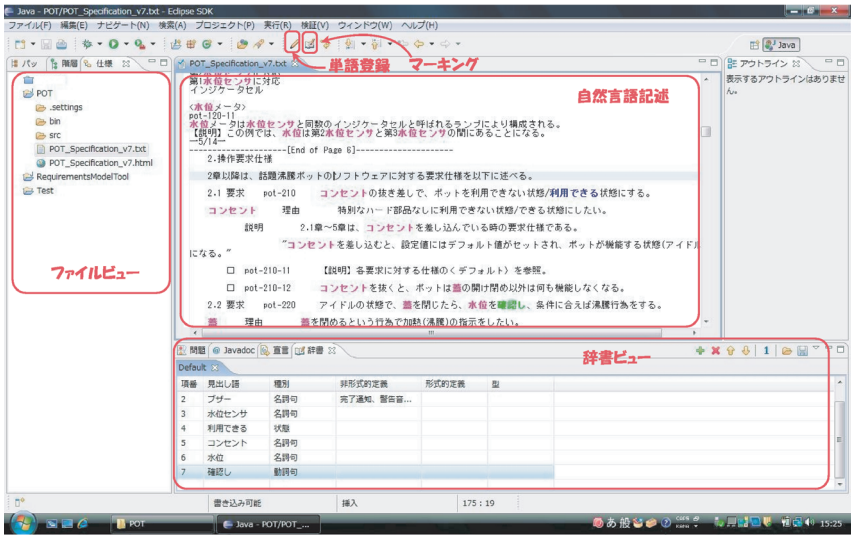

図 2　モデル変換補助ツール
Fig. 2　Support tool for model conversion.

仕様書のマーカは，プロジェクト内の自然言語記述をファイルビューから読み込むことができる．また，同プロジェクト内で，使用する辞書を指定できる．仕様の修正が頻繁に行われることを想定して，ロールバックを容易にするために，指定したフォルダ/ディレクトリにファイル名が同じファイルがあれば，一意になるまで (1)，(2) のように連番を付加し，作成したファイル名の確認するメッセージを表示する．

3.1.1 登 録 機 能

単語登録の手順は，読み込んだ自然言語記述に含まれる対象領域をマウスなどで選択したのち，登録確定操作により登録するものとし，登録前は対象領域を修正できる．ここでは自然言語記述の任意の範囲を指定して登録可能であり，辞書の見出し語として追加される．

登録はマウスの右ボタンから呼び出されるメニュー，あるいはツールボタンにより確定し，リアルタイムに指定した辞書に追加される．辞書データは，3.2 節の辞書ツールで編集したり，あらかじめ用意しておいたりすることもできる．

3.1.2 マーク機能

マーキングボタンが押されたら，辞書に登録されている見出し語に対して，自然言語記述

中の同一文字列にすべてマークをつける．マークは最初に出現したものとそれ以降に異なる色をつけられる．また，設定ファイルにより，名詞句，動詞句，状態といった品詞によって異なる色を指定できる．これにより，自然言語記述中のキーワードの視認性を向上させる．

この機能を利用して，たとえば日本語文章そのものの品質向上を図ることも可能である．表記のゆれの確認はツールによる全自動検査が有効であり，長音の有無，送り仮名，ひらがなとカタカナなど，類似の見出し語について辞書を用意し，切り替えながらそれらを用いてマーキングすることで，問題の生じる可能性が高い自然言語表現を見つけやすくし，誤りを予防することができる．

3.1.3 検索機能

マーカの本質的な機能は，自然言語記述の文字列検索である．

本ツールでは，複数の自然言語を対象とすることを考慮して，品詞分けなどの文法的な知識を使っていない．ツールの実装では，Eclipse に用意されている単純な文字列一致を行う検索機能を利用しており，正規表現を利用することもできる．

複数の見出し語が同じ文字列に一致する場合，

(1)　優先度の高い辞書
(2)　同一辞書内の最長一致
(3)　同一辞書内の先頭に近い単語

という順に優先度に応じてマーキングされる．

3.2 辞書管理

辞書ツールは，辞書ビューから操作可能であり，登録する文字列を入力として受け取り，指定された辞書へ追加する．

登録した見出し語には，品詞，自然言語による非形式的定義，形式言語による形式的定義を追加できる．品詞には，名詞句，動詞句，状態の選択肢を用意し，それぞれ，クラスあるいはオブジェクト，関連あるいは操作または関数，状態変数として形式モデルへ変換したテンプレートを生成する．

3.2.1 見出し語の編集

辞書ツールの編集機能として，

(1)　項目間の編集
(2)　項目の編集
(3)　文字列編集

がある．

```
"項番","見出し語","種別","種別 No","非形式的定義","形式的定義","型"
1,"蓋","名詞句",1,"","",""
2,"ブザー","名詞句",1,"完了通知，警告音など","",""
3,"水位センサ","名詞句",1,"","",""
4,"利用できる","状態",3,"","",""
5,"コンセント","名詞句",1,"","",""
6,"水位","名詞句",1,"","",""
7,"確認","動詞句",2,"","",""
```

図 3　辞書フォーマット
Fig. 3　Dictionary format.

項目間の編集には，見出し語のソート，項目の追加・削除・移動・コピーなどがある．

項目の編集は，見出し語の種別によりやや異なる．共通の要素としては，見出し語をキーとし，見出し語の種別，非形式的定義，形式的定義，型があり，たとえば，動詞句の場合は上記に加えて，関連元，関連先もリンクする．また，各項目のうち，記述されていない要素を検索し，記入を促す機能も用意している．

文字列編集は，一般的なワードプロセッサ同様に，検索，置換，対話的置換などである．

図 3 に示すように，辞書は CSV 形式であり，外部エディタなどを用いても修正や追記は容易である．

3.2.2 複数辞書の切替え

本ツールにおける辞書は，入力となる自然言語記述と出力となる形式言語記述のマッピングを表現している．したがって，辞書を交換することにより，複数の入力言語あるいは出力言語を切り替えることができる．

このために，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

を記述するフィールドがある．対象領域は，ドメインであり，設計対象だけでなく，文章検査など任意のドメインを識別する．また，プロジェクトや組織による検査項目の違いからくる見出し語の違いは，利用組織により識別する．入力言語，出力言語は辞書の見出し語の記述言語，形式モデルの記述言語である．

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．また同一の優先度

で複数の辞書を指定することもできる．

ここで，入力言語は対象の自然言語全体ではなく，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特定ドメイン言語（Domain Specific Language, DSL）に相当するので，相当するドメイン単位で辞書の再利用が可能となる．

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辞書を再利用することにより，キーワードの見落としを防ぐことができる．

4. 相互変換の手順

本研究で想定する開発プロセスは，ツールによる全自動化ではなく，設計者や要求記述者がより良いモデル記述へ詳細化していくというものであり，開発環境は軽量なツールの組合せで実現する．こうした協調作業においては，完全でなくても短いサイクルでお互いへフィードバックすることで，要求記述者による自然言語記述の充実と，設計者による形式モデルの検証を分担し，効率的に並列化することも可能となる．本ツールを用いて辞書に登録した見出し語から，形式モデルのテンプレートを出力することができる．しかし，ソフトウェアのモデリングには，抽象度や重要性に関する階層性がしばしばみられる．特に対象ドメインの知識が不足している場合は，自然言語記述に含まれる要素を 1 度に形式モデルと対応づけるよりも，設計者の理解に応じて，重要な概念から徐々に形式モデルに取り込んでゆくのが望ましい．

これまでの試行の結果，本ツールを用いた形式モデルへの変換法として．次の手順がうまくいくようである．ただし，これはモデルへの要素の追加手順であり，辞書への登録はこの順でなくてもよい．この手順は提案されているフォーマルメソッドにおけるモデリング法[7]に，階層的なグループ化を追加した手順となっている．

(1) キーワード（名詞句）の抽出
自然言語記述から重要な概念を表すキーワードを選択し，さらに名詞を抽出する．最終的には，すべてのキーワードをモデルに取り込むので，品詞の正確さにはこだわらなくてよい．

(2) グループ化 I
継承関係，集約関係をもつ名詞をグループ化する．これらの関連は，不変である可能性が非常に高い強固な関係であるので，以後，基本的に一体として扱う．

(3) 類義語・多義語の整理
自然言語記述に含まれる類義語・多義語を整理する．用語の一貫性は自然言語記述の品質に大きな影響を与える．

(4) グループ化 II
いくつかの評価軸に基づいて，キーワードをより大きな群へまとめる．
一般的な評価軸としては次のようなものがある．
- サブシステム ... 関連のあるキーワード
- 抽象度 ... システム一般，システム間，システム内など記述の汎用性
- 時間 ... 設計時，起動時，実行時などキーワードが有効な時期

(5) キーワード（状態）の追加
状態の変化する範囲に関する要件をモデルに追加し，データ型を定義する．この時点でも，キーワードの品詞について見直す．

(6) キーワード（動詞句）の追加
キーワードとなる動詞句のすべてが，名詞句であるキーワードの関連に相当することを，関連の方向および多重度も含めて確認し，見落としている関連があれば補足する．

(7) パラメータ化
パラメータ化により，共通性の高い属性，関連などをまとめられないか検討する．

(8) 形式モデルの詳細化
形式モデルのテンプレートを出力し，完全なモデルになるように自然言語記述から情報を補完する．これは通常のモデリング作業と同じであるが，形式モデルは冗長な表現がほとんどないため，不足している情報が見えやすい．また，意味論が定義されているので計算機言語への変換が容易であり，実装可能性を検討しやすい．こうした特性により，形式モデルを記述するだけでもメリットがある．

(9) フォーマルメソッドによる形式モデル検証
モデルの内的矛盾がないか，モデルが表明を満たしているかを検証する．形式モデルでは，モデル内の関連や制約に関する矛盾を数理的に判定し，ツールにより検出できるので，対応する記述を改善することにより，一貫性の高い自然言語記述が可能となる．

(10) 要件定義書へのフィードバック
形式モデル化に際して行った整理やモデルに追加した要素を自然言語記述へ反映させる．これは，仕様記述者が明示的に行うことで，自然言語記述の意味的な改善と記述の分かりやすさの両立を図る．

ただし，上記の項目 (8) および (9) は，本ツールの機能ではなく，既存の形式モデルに関する CASE ツールを利用し，既存のフォーマルメソッドと同じ手順を適用する．この手順を繰り返すことで，自然言語記述の改善と形式モデルの充実を進め，関係者の合意がとれた時点で確定した仕様となる．

本章の手順で達成される本研究の目標は，キーワードおよびキーワード間の関連についての無矛盾性の確認である．自然言語記述が求める対象をきちんと記述しているかどうかは分からないので，本手法で問題が見つからなかったからといって完全性を保障できるわけではない．言い換えれば，本手法では，キーワードの発見については従来と同様に行い，そのキーワードが適切なものかどうか確認する手段を提供する．そこで問題がみつかった場合は，なんらかの修正を行い，再び本手法を適用して確認する．

それぞれのキーワードについては，自然言語記述のすべてを走査することにより，漏れなく同じ意味で用いられていることを確認できる．多義語や同義語あった場合，それらの解消方法やそもそも解消すべきかは仕様記述の責任者による．

キーワード間の関連については，問題があれば形式モデルの検証過程を通じて，型の不一致や状態遷移の異常といった内的矛盾として顕在化する．このとき，キーワードの選択ミスあるいは記述不足，キーワードの定義が誤っている場合，関連そのものの定義が誤っている場合が考えられる．形式モデルの修正により解消できた場合は，その内容を辞書によるトレーサビリティを通じて自然言語記述にフィードバックする．

外部から与えられた仕様として満たすべき表明については，問題があれば証明不能あるいは状態遷移の異常として顕在化する．この場合も，表明とモデルのどちらに問題があるかを確認し，修正した内容を辞書を通じて自然言語記述へフィードバックする．

このように，辞書により限定された対応部分のマッピングを保存し，形式モデルで検証した性質が自然言語記述でも成り立たせることが重要である．

5. 事例研究

ツールを活用した自然言語記述改善について，組み込みソフトウェア開発を例に簡単な評価を行った．

5.1 電子ポット

SESSAME による「話題沸騰ポット（GOMA-1015 型）要求仕様書第 7 版」を例として，マークしたものの一部を**図 4** に示す．

4 節の手順により，状態機械としてモデル化した．

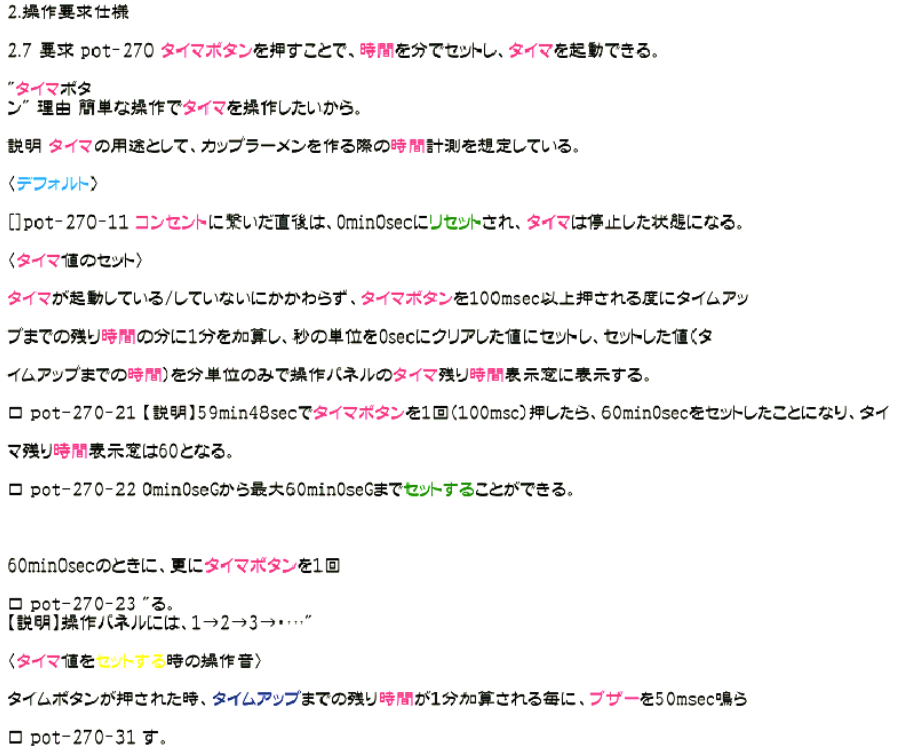

図 4　電子ポットへの適用
Fig. 4　Application example to an electric pot.

(1)　キーワード（名詞句）の抽出
84 個のキーワードが見つかり，そのうち 38 個であった．
しかし，最終的に名詞句となったのは 28 個だけであり，その他は状態としてモデル化した．

(2)　グループ化 I
沸騰ランプ，保温ランプなどのインジケータ，タイマボタン，時間表示窓などのタイマ，などをグループ化した．ただし，ハードウェアに関する部分がほとんどであり，モードに関して「高温」「節約」「ミルク」をグループ化したものと，水位に関して

「第 1 水位センサ」から「第 4 水位センサ」および「満水センサ」をグループ化したものを除いて，今回のモデルには影響していない．

(3)　類義語・多義語の整理

キーワード以外では,「水量」と「水」の混用などが見つかったが，例題として利用した第 7 版はかなりこなれており，選択したキーワードに明らかな類義語や多義語はなく，モデルには影響していない．

たとえば，このシステムにおけるタイマは，残り時間 0 を知らせるだけのものであり，ある値の「時間」をセットし，秒または分単位で減算される機能しかないので，キーワード「時間」とキーワードでない「残り時間」は，同じ意味で使われていることが分かる．ここで，元の自然言語記述の「時間」を「残り時間」に統一するべきかどうかは，分かりやすさと記述のくどさのバランスを考慮し，プロジェクトで選択すべき事項である．

(4)　グループ化 II

この事例では考慮していない．

(5)　キーワード（状態）の追加

(6)　キーワード（動詞句）の追加

キーワード（状態）およびキーワード（動詞句）は，新たに追加したものはなく，最初に抽出したキーワードの品詞を変更した．いくつかのキーワードは状態でなく，状態の値として名詞句としている．たとえば,「モード」は状態キーワードであり,「高温」「節約」「ミルク」は状態の値である．

(7)　パラメータ化

n 個の水位センサに関するキーワードをパラーメータ化した．

本事例では形式定義は VDM++ の OOA 機能を利用してクラスとしてパラメータ化している．

(8)　形式モデルの詳細化

図 5 に対応する形式モデルの一部を示す[24]．

本事例の記述には VDM の仕様記述言語である VDM++ を用いた．

テンプレートとして，辞書の形式定義のフィールドに記述した最外クラスをそれぞれ 1 つのファイルとして出力した．

詳細化にあたっては，テンプレートのクラスを部品として，システム全体の振舞いクラスを新規に作成し，状態遷移に関する事前条件や事後条件を追加した．

```
class『ポット』

types
『温度』 = real inv temp > -20 and  temp < 120;
『水位』 = int inv height > 0;

instance variables
ヒーター : <アイドル> | <沸騰行為> | <保温行為> | <エラー> ;
コンセント : <差された> | <差さっている> | <抜かれている>  ;
蓋 : <開> | <閉>  ;
タイマー : 『タイマー』;
ロック : <されている> | <されていない>  ;

保温モード : <高温> | <節約> | <ミルク> ;
水位 : 『水位』;
水温 : 『温度』;

operations
public 給湯する : () ==> ()
給湯する () ==
is not yet specified;
pre 水位 > 0;
```

図 5　VDM による形式モデル
Fig. 5　VDM formal model.

(9)　フォーマルメソッドによる形式モデル検証

VDM++ による記述について，VDM-Tools 上で文法，型の検査を行い，矛盾がないことを確認した．

本事例では，組み込みシステムであることを考慮し，特に状態遷移の条件について重点的に検証した．

この結果，後述するブザーの鳴らし方が不明である，エラー状態への遷移条件の記述がないといった不備をこの段階で見つけることができた．

(10)　要件定義書へのフィードバック

後述のような曖昧な点があり，要件定義書を改善する余地があることが分かった．

この例について次のような曖昧な点が指摘できる．

- POT221

「温度制御をしない」とは，アイドル状態を指すのか，それともポットの利用ができな

い状態を指すのかが不明．ここから，蓋を閉めた場合の挙動に影響する．

- POT230-11
  ブザーが 100 msec 鳴った後，ブザーが鳴り止むのかどうかが不明．また，ブザーが「鳴らす」という操作を行われているのか,「鳴っている」という状態になっているのかも解釈の余地がある．
- POT260-11
  「温度エラー」は定義されていない用語である．
- POT500-21
  コンセントを抜いてもブザーは 30 秒鳴り続けるとは考えにくい．
- STM
  エラー処理中のエラーが考慮されていない．全体にエラーからの回復手順の記述が不足している．

これら以外に,「利用できる/利用できない」と「利用できる/できない」が混在していたり,「温度」と「水温」が同じ意味で使われていたりといった類義語が発見できた．

5.2 変換手順

この事例から得られた知見は，次のようなものである．

- すべての単語をキーワードとする必要はない．
  手作業で検証する場合，設計者が理解できるように，キーワードは 100 語以内が望ましい[16]．それで十分な検証ができない場合，対象とするシステムが大きすぎるので，サブシステムに分割する．
  この例では，キーワードは 84 語抽出されたが，動詞の格変化をまとめたり，ハードウェア要素を除いたりすると，形式モデルの要素としては 40 語程度になった．
- 品詞はそれほど重要ではない．
  要件整理では，キーワードの抽出をもれなく行うのが最重要であり，厳密な品詞の確定は形式モデル化の段階で検討すればよい．特に日本語では,「名詞+する」といった用法があり，これが動作（関連）か名詞（オブジェクト）かはよく考慮する必要がある．
- 動詞の主語はそれほど重要ではない．
  関連は複数のオブジェクト間で成立するので，自然言語による要件定義の段階ではいずれのオブジェクトが主語になることもありうる．ただし，形式的なモデル化の段階では，関連の主体を見極める必要がある．
- 状態は状態変数をキーワードとする．
  状態は，状態変数の値の集合により定義される．したがって，形式モデル記述の際に複数のキーワードのそれぞれが，1 つの状態変数の値としてまとめられる場合もある．

キーワードの抽出は，計算機システムの設計者よりも，対象ドメインの専門家が行うのが望ましい．しかし，抽出されたキーワードをどのようにモデル化するかは，ソフトウェアのモデリングに関する知識が必要であり，品詞などがドメインの専門家と一致しない可能性もある．そのような場合も，提案手法では両者の結合を弱い対応づけにとどめているので，自然言語記述と形式モデルを独立に考えることが可能である．

6. まとめ

本稿では，自然言語記述の品質改善のために，フォーマルメソッドによる形式モデルを用い，その検証結果をフィードバックする手法を提案し，さらにその手順をサポートするツールを作成した．このツールは Eclipse 上で動作し，既存のフォーマルメソッドのツールとも連携する．

電子ポットの例題では，完成度の高い第 7 版においても，いくつかの曖昧な点が見つかった．提案手法では，ツールの活用により，自然言語記述から形式モデルの作成および形式モデル上の検証結果の自然言語記述へのフィードバックが効率良くできる．

その際，モデリングそのものの難しさは残るので，形式モデルの構成と対象ドメインの記述をそれぞれの専門家が分担し，協力して行うのが望ましい．本手法は，そのような状況にもよく対応できる．

今後の課題として，入力となる自然言語記述にしばしばみられる階層性などの構造を，形式モデルに対応づける技法や，ネットワーク上で複数人での協調作業をサポートする版管理システムの組み込みなどがあげられる．

謝辞　本研究は科研費（21300009），基盤研究（B）「ソフトウェア開発の現場で使えるフォーマルメソッドに関する研究」の助成を受けたものである．

参考文献

1) Abrial, J.-R.: *The B-Book: Assigning Programs to Meanings*, Cambridge University Press (1996).
2) Gaizauskas, R. and Harmain, H.M.: CM-Builder: An Automated NL-Based CASE Tool, *Proc. 15th IEEE International Conference on Automated Software Engineering*, pp.45–54 (2000).
3) Gerhart, S., Craigen, D. and Ralston, T.: Experience with formal methods in crit-

ical systems, *IEEE Software*, Vol.11, No.1, pp.21–28 (1994).
4) Hinchey, M.G. and Bowen, J.P.(Eds.): *Applications of Formal Methods*, Prentice Hall (1995).
5) Holzmann, G.J.: The Model Checker SPIN, *IEEE Trans. Softw. Eng.*, Vol.23, No.5, pp.279–295 (1997).
6) Jones, C.B.: *Systematic Software Development using VDM*, Prentice-Hall (1990).
7) Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M. and Fitzgerald, J.: *Validated Designs For Object-oriented Systems*, Springer (2005).
8) Ogata, K. and Futatsugi, K.: Modeling and Verification of Distributed Real-Time Systems Based on CafeOBJ, *Proc. 16th IEEE International Conference on Automated Software Engineering*, IEEE Computer Society (2001).
9) Pohl, K., Böeckle, G., vander Linden, F.J.（著），林　好一，吉村健太郎，今関　剛（訳）：ソフトウェアプロダクトラインエンジニアリング，エスアイビーアクセス (2009).
10) SESSAME WG 2：組込みソフトウェア開発のためのオブジェクト指向モデリング，翔泳社 (2006).
11) Saeki, M., Horai, H. and Enomoto, H.: Software development process from natural language specification, *Proc. 11th International Conference on Software Engineering*, pp.64–73, ACM (1989).
12) Schneider, S.: *Concurrent and Real Time Systems: The CSP Approach*, John Wiley & Sons, Inc. (1999).
13) Vadera, S. and Meziane, F.: From English to Formal Specifications, *British Computer Society, Computer Journal*, Vol.37, No.9, pp.753–763 (1994).
14) Yin, M.-L., Hyde, C.L. and James, L.E.: A Petri-net approach for early-stage system-level softwarereliability estimation, *Proc. Annual Reliability and Maintainability Symposium*, pp.100–105 (2000).
15) CSK システムズ．http://vdmtools.jp/
16) 宇田川佳久：情報システム開発におけるトレーサビリティの事例と今後，情報処理学会誌，Vol.51, No.2, pp.150–158 (2010).
17) 小林吉純：日本語による電話サービス仕様記述における表現の多様性と意味の同一性の認識，電子情報通信学会論文誌，Vol.J82-D1, No.8, pp.1035–1048 (1999).
18) 和田　孝，土田賢省，杉山高弘，阪田全弘，富田兼一，宮下洋一：プログラム自動生成のための日本語仕様記述言語，情報処理学会研究報告，Vol.1988-PRO-016, No.27, pp.33–40 (1988).
19) 杉尾俊之，武内　惇，椎野　努：日本語をベースにした仕様記述言語 NBSG における手続記述について，情報処理学会研究報告，Vol.1983-SE-034, No.55, pp.73–78 (1984).
20) 中島　震：SPIN モデル検査—検証モデリング技法，近代科学社 (2008).
21) 大石東作：ソフトウェアの「日本語による仕様書」作成支援システム，情報処理学会研究報告，Vol.1988-SE-059, No.32, pp.1–7 (1988).
22) 西田富士夫，高松　忍，谷　忠明：要求仕様における日本語表現と形式表現間の相互変換，情報処理学会論文誌，Vol.29, No.4, pp.368–377 (1988).
23) 日経エレクトロニクス編：組み込みソフトウエア 2007 モデルに基づく開発方法論のすべて，日経 BP (2006).
24) 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，Vol.2009-EMB-14, No.3, pp.1–8 (2009).
25) 池田陽祐，三塚恵嗣，加藤木和夫，大木幹生，畠山正行：オブジェクト指向分析記述言語 OONJ の記述例を用いた簡潔な表現技法の開発，情報処理学会研究報告，Vol.2010-MPS-78, No.17, pp.1–8 (2010).
26) 加藤木和夫，畠山正行：オブジェクト指向日本語一貫プログラミング環境，情報処理学会論文誌，Vol.40, No.7, pp.3016–3030 (1999).

(平成 22 年 5 月 14 日受付)
(平成 22 年 8 月 23 日採録)

大森　洋一（正会員）

1993 年京都大学工学部情報学科卒業．1998 年奈良先端科学技術大学院大学博士課程修了．同年高知工科大学工学部助手．2004 年九州大学助手．2007 年同助教，現在に至る．博士（工学）．高次システム設計および組み込みシステムソフトウェアの信頼性に関する研究に従事．電子情報通信学会，IEEE Computer Society 各会員．

荒木啓二郎（正会員）

1976 年九州大学工学部情報工学科卒業，1978 年同大学院修士課程修了．九州大学助教授，奈良先端科学技術大学院大学教授等を経て，現在，九州大学大学院システム情報科学研究院教授．ACM，IEEE，Formal Methods Europe，日本ソフトウェア科学会等の会員．日本学術会議連携会員，IEEE Fukuoka Section Past Chair，ソフトウェア技術者協会幹事，プロジェクトマネジメント学会九州支部副支部長，VDM 研究会会長，等．工学博士．形式手法に基づくソフトウェア開発，IT 人材育成等に従事．