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

2008－SE－159 (1)
2008／3／17

日本語構文解析を用いたユースケース記述から シーケンス図への変換

奥村 和恵† 金澤 典子† 塚本 享治†
†東京工科大学 メディア学部 メディア学科

あらまし
UML などの図を中心に開発を行う、モデル駆動型の開発手法がある。しかし図の作成は慣れない開発者にとって難しい。そこで本稿では仕様書の一種であるユースケース記述に含まれるモデルの要素に着目し、シーケンス図に変換することを取り上げる。まずユースケース記述を構文解析し、単語の係り受けや接続詞を分析して、モデルの要素を抽出する。抽出した要素を XMI 形式に変換し、モデリングツールに読み込ませてシーケンス図とクラス図のクラスと属性を作成する。実際に実験ツールを作り、ユースケース記述から図を自動生成した。生成された図を分析した結果、有益であることが確かめられた。

Generating sequence diagrams from use case descriptions with syntax analysis

Kazue Okumura†, Noriko Kanazawa†, Michiharu Tsukamoto†
†Tokyo University of Technology, School of Media Science

Abstract
There is a method which develops software systems by UML diagrams. It means "Model Driven Architecture". But creating diagrams is too difficult for beginners. This paper pays attention to model entities in the use case description, and generates sequence and class diagrams.
First, the conversion tool parses the use case descriptions, and chooses model entities from the result of syntax analysis. Next, the conversion tool changes from these entities to XMI. At last the UML modeling tool reads XMI and creates sequence and class diagrams.
We developed a conversion tool and practiced experiments. Sequence and class diagrams created by experiment. As a result, the conversion tools has usability of creating diagrams.

1. はじめに

巨大化・複雑化しているソフトウェア開発を簡単にするため、オブジェクト指向などの手法が開発された。この一部に UML などの図を中心に開発を行う、モデル駆動型の開発手法がある。設計を一貫して管理でき、仕様変更などにも対応できる。

しかし図の作成は慣れない開発者にとって難しい。また開発中は進捗成果の見えやすいコーディング作業に時間を掛けがちで、分析と設計段階である図の作成に時間はあまり取られない。さらに仕様変更に伴い頻繁に更新されるのが問題である。

そこで仕様書の一種であるユースケース記述に含まれる、UML 図の要素に着目した。ユースケース記述はユースケースを満たすためにシステムが行う処理が書かれており、ソースコードに近い造りである。本稿は ICONIX 法で開発の中核となる、シーケンス図とクラス図への変換を取り上げる。

ユースケース記述は自然言語であるため、図の要素が上手く抽出できなかった。ユース

ケース記述の文法を限定することで抽出しやすくし、変換ツールを作成して実験を行ったので報告する。

広瀬[1]は同じユースケース記述からシーケンス図への変換を試みた。しかし解析できる日本語文章が限られていたり、フラグメント処理(Loop/Alt)の変換に問題があった。これらの問題を解決したので報告する。

自然言語の研究はかなり以前から行われてきた。その結果、細かい領域に渡って成果が出ていた。[2]から[5]の研究は日本語解析の一部で、正確な構文解析を行うための手法についてである。金山による[2] は読点による構文解析の誤解析について扱っている。この研究は本研究を進める際に、日本語の正確な構文解析の手法に役立った。[3]の研究は品詞列と文構造による構文解析の手法と正確性について分析している。本研究の変換実験も同様に品詞と文構造から、文中の図要素を判別するので参考になった。[4]の研究は接続助詞による複文などの係り受け構造を分析している。本変換実験のテーマのひとつに複文の分析がある。この研究で判明した接続助詞と副文の係り受け構造の関係を、解析の参考にした。[5]の研究は Ko-BaKo による係り受け関係の、正確な解析に応用できた。

[6]の研究はワークフローから画面遷移を再現した。画面遷移の過程をシステムの試作品（プロトタイプ）として、日本語文章から生成した。日本語文章を単語レベルで解析し、作者にとって都合の良い文章に変換してから、試作モデルを作成する。本研究にとってはユースケース記述を解析し、手順を抽出する先例となった。

2 節ではユースケース記述から UML 図の要素を抽出する手法について述べ、3 節では抽出した図要素からシーケンス図とクラス図への具体的な変換方法について述べる。4 節で実験結果から分かった、ユースケース記述の文法制限と、ユースケースの分類について述べる。

2. 文の解析と UML 図要素の抽出

ユースケース記述という日本語文章から、UML 図の要素を抽出する。抽出には構文解析による、助詞と修飾構造の情報が必要である。

1.1. Ko-BaKo/Japanese による構文解析

用意した日本語ユースケース記述を構文解析し、解析結果から UML 図の要素を集めた。構文解析には、多機能日本語ライブラリである Ko-BaKo/Japanese[7]を利用した。Ko-BaKo は解析結果を XML 形式で出力するため、XSLT 変換プロセッサで XMI 形式へ変換しやすいからである。

構文解析では図要素の判定に必要な単語に付く助詞と、単語が修飾する語（修飾構造）の情報を解析した。

1.2. 複文の解析

ユースケース記述には単文だけではなく複文も登場した。複文とは複数の単文がひとつにまとまったもので、述語を複数持つ。Ko-BaKo で構文解析を行うと、内包する単文の述語に付く助詞は、複文独特の「SOSHITE」と解析された。この助詞によりひとつの複文を 2 つ以上の単文に分割でき、それぞれの単文で解析を行った。

複文を構成する単文は、主語や目的語が重複のために省略されていることが多い。複文を単純に単文に分解しただけでは、省略された単語が分からなかった。そこで省略された語を文脈に沿って補うようなプログラムを作成した。アルゴリズムの詳細は0節の(4)項で説明する。

1.3. 助詞による UML 図の要素の抽出

UML 図の要素は変換ツールが、助詞から分かる文中の単語の役割(主語、目的語など)によって判断した。変換ツールではまず、構文解析によって解析される助詞により、各単語が文中でどの役割になるかを判断した。そしてこの役割から、単語に UML 図のどの要素が適切かを決めた。この判断プロセスを要約したものが表 1である。

図 1のようにユースケース記述の文から、シーケンス図とクラス図の両方の要素が抽出できる。

表 1　助詞と図要素の対応表

助詞	文中での役割	シーケンス図の要素	クラス図の要素
は、が	主語	送信元オブジェクト	クラス
に	目的語(与格)	受信先オブジェクト	クラス
の	目的語(対格)	引数・戻り値	その他クラス
を、から、など	目的語(対格)	引数・戻り値	属性
まで	文全体	Loopフラグメント	×
なら(ば)	文全体	Altフラグメント	×
解析結果=空白	述語	メッセージ	振舞
解析結果=「Φ()」	修飾句	修飾句	修飾句

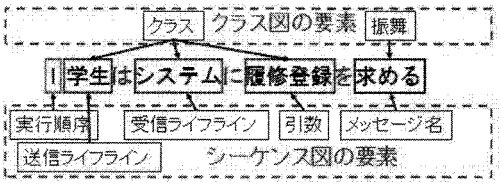

図 1　通常処理文と UML 図の要素の対応

主語となる単語は、助詞が「GA」(が)と解析された単語である。実際の文に付く助詞が「は」でも、Ko-BaKo では「GA」と解析された。文中の主語は、シーケンス図では「送信元ライフライン」というメッセージの送り手である。またメッセージを送信できるのは、クラスだけであるので、主語はクラス図で自動的にクラスと判断できた。与格目的語は格助詞「に」が付き、構文解析では「NI」と解析された。しかし他の助詞「へ」などが付くこともあり、一概に「に」だけとは言えなかった。シーケンス図ではメッセージを受ける「受信先ライフライン」となった。クラス図では主語と同様の理由で、クラスと判断できた。

対格目的語には通常、格助詞「を」などが付く。しかし他にも「から」「より」など、様々な助詞が付くこともあった。対格目的語はシーケンス図では大抵、メッセージで受け渡される「引数」となる。しかし与格目的語が文中にない場合、対格目的語が「受信先ライフライン」となった。クラス図で引数は、クラスが受け渡される場合と、属性が受け渡される場合があった。クラスか属性かの見分けは、記述方法を変えることで判断した。この記述制限は0章に述べる。

単文の述語は日本語文法では通常文末に記述され、主語が行う動作を表す。動作は自動的にシーケンス図のメッセージになったので、述語の語幹は「メッセージ名」であった。またメッセージはクラス図では、クラスが行う「振舞」と同じであった。

1.4. 修飾構造による修飾節の抽出

各図の要素を修飾する修飾節は、構文解析によって解析される助詞と修飾構造の解析結果（図 2）から判断した。

構文解析で各単語について修飾する語が明らかになった。また修飾節の助詞には「φ( )」という独特な解析結果が出た。この 2 点から、以下の場合に修飾節と判断した。

(1) 特殊な助詞「φ」が解析結果に含まれる
(2) 図の要素（主語や目的語など）を修飾する単語である

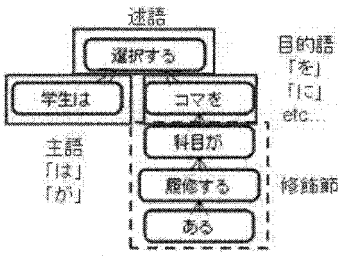

図 2　修飾構造による抽出

3. 文から UML 図へ変換手順

前節で記述した UML 図の要素の抽出方法を使い、変換ツール（図 3）でユースケース記述からシーケンス図とクラス図に変換した。変換は全て XML ファイルと XSLT 変換スクリプトによって行った。構文解析から XMI ファイルの生成までを、Ant によって一括処理した。

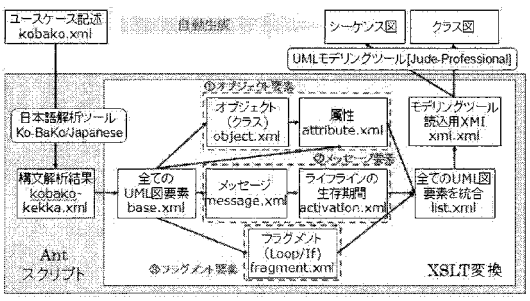

図 3　変換の全体像

(1) 変換ツールでは第一に、XML 形式で用意されたユースケース記述の構文解析を行った。

構文解析の結果から、前節の基準によってUML 図の生成に必要な要素を取り出した。同時に現時点で必要な要素が抜けていないか、DTD チェックを行った。

(2) (1)で単純に文中から要素を取り出しただけでは、必要な要素が全て揃わなかった。従って変換ツールでは不足している要素を、抽出結果からさらに抽出と計算をする必要があった。このプロセスは要素の系列ごとに、以下の 3 グループに分けて行った。

① オブジェクト要素
- オブジェクト（クラス）
- 属性

② メッセージ要素
- メッセージ（振舞）
- ライフラインの生存期間(Activation)

③ フラグメント要素
- フラグメント

(3) 変換ツールは 3 グループに分けた要素を、ひとつの XML ファイル（リスト 1）にまとめた。次ステップの XMI ファイルへと変換しやすくするためである。

(4) 変換ツールは抽出と精製した図要素から、UML モデリングツールへ読み込ませる XMI ファイルを生成する。以上の 5 ステップで変換ツールの役割は終了する。

(5) 変換ツールが生成した XMI を、人が UML モデリングツール[Jude-Professional][9]に読み込ませた。読み込みはモデリングツールのコマンドのひとつ、「XMI からの入力」で行った。読み込むとモデリングツール内で、シーケンス図とパッケージに入ったクラスが自動生成された。パッケージを右クリックして「詳細クラス図の自動生成」を選択すると、クラス図が自動生成された。以上で変換は終了する。

リスト 1　抽出し精製した UML 図の要素

```
<XML>
 <object>                                          オブジェクト要素
  <no>1</no> <name>学生</name>
 </object>・・・・・・
 <attribute class="1" type="int">                  属性要素
  <no>1</no> <name>学籍番号</name>
 </attribute>・・・・・・
 <message>                                         メッセージ要素
  <no>1</no> <step> 1 </step> <name>開始</name>
  <arg type="2">履修登録</arg> <type>create</type>
 </message>・・・・・・
 <fragment>                                        フラグメント要素
  <no>9</no> <step> 7 </step> <type>loop</type>
  <guard>                                          ガード条件
   <no>1</no> <seqence>基本系列</seqence>
   <condition>学生は履修する科目を全て登録</condition>
   <message><no>3</no><no>4</no></message>
   <object><no>1</no><no>5</no></object>
  </guard>
 </fragment>・・・・・・
 <activation>                                      メッセージの生存期間の要素
  <no>1</no> <object>1</object>
<message>1</message>
  <send><no>1</no></send> <receive/> <reply/>
 </activation>・・・・・・
</XML>
```

4. 日本語ユースケース記述文法の提案

制限が何もないユースケース記述では、正確な解析や変換ができなかった。そこで UML 図への変換専用記述として、以下(1)から(7)に述べるユースケース記述文法の制限を設けた。記述制限を守らなければ、変換した UML 図が実際の処理を反映しなかったり、正確な図が生成できなかった。

- メッセージが繋がっていない
- メッセージが一種類しかない
- 余分なライフラインが多い

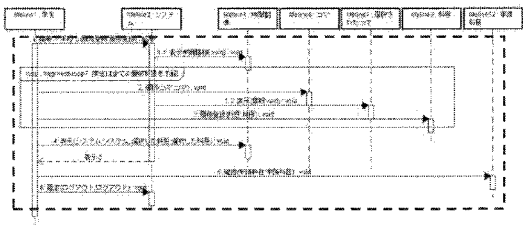

図 4　不正確なシーケンス図

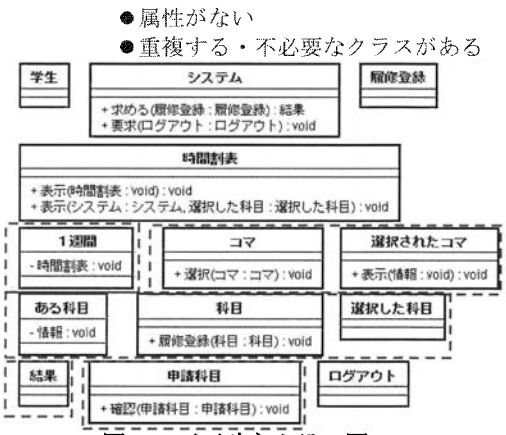

図 5　不正確なクラス図

(1) ユースケース記述はサブシステム毎に書く

通常、ひとつのシステムは複数のサブシステムから構成される。ソースコードも機能ごとに複数のモジュールが独立して動作する。変換結果のシーケンス図とクラス図は、ICONIX 法[7]ではソースコードと単体テストになる。これを考えると図の素になるユースケース記述は、ソースコードに近い作りであるサブシステム毎に書く必要があった。

(2) 内部処理を記述

通常のユースケース記述には、アクタとシステムのインターフェイス部分しか書かれない。しかし(1)と同様の理由でこれ以外の処理、特にシステムの内部処理の情報が必要であった。またソースコードに近づけるためには、例外処理も同時に記述する必要があった。

(3) 文中のオブジェクト名の統一

この変換ツールでは、ユースケース記述文中の単語がそのまま、オブジェクト（クラス）または属性として抽出される。文中の語句が統一されていなければ、現実世界ではひとつのオブジェクトが、生成結果では表現の違いから複数のオブジェクトに分かれてしまう。これを避けるために、あらかじめクラス名だけを記述したドメインモデルを作成する。文中ではドメインモデルの中にある用語だけを、使うようにすればよい。

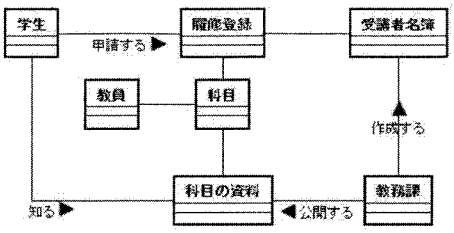

図 6　用語の統一に使うドメインモデル

(4) 文中の単語を省略しない

日本語では文中で重複する単語は、省略する習慣がある。省略対象となるのは主語や目的語であり、シーケンス図の生成において重要な役割を持つ語であった。主語はメッセージの送信元ライフラインを、目的語は受信先ライフラインを示し、文中になければ図の正確さに大きく影響した。

重要な単語が省略された場合に備えて、省略語を推測するプログラムを、変換ツールに組み込んだ。プログラムが省略語を判断する基準は 2 点ある。

1 点目の基準は、「ユースケース記述では主語と与格目的語は交互に書かれる」ということであった。主語が省略された場合、前文の与格目的語が第一候補となる。逆に与格目的語が省略された場合、前文の主語が第一候補である。

2 点目の基準は、「送信元オブジェクトと受信先オブジェクトは重複しない」と仮定したことであった。ただし実際のシーケンス図では、同じオブジェクトへメッセージが送信されることも多々あった。

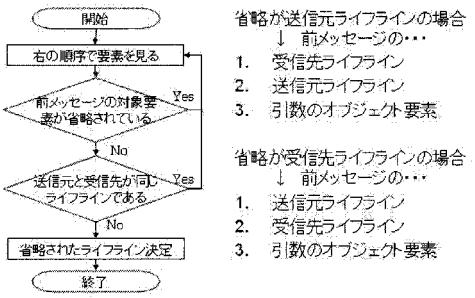

図 7　省略単語を補うアルゴリズム

この推測プログラムで合う場合もあったが、全ての場合で正しいとは限らなかった。結局は単語を省略せずに書くことが一番良かった。

(5) フラグメントの書式を指定

シーケンス図で繰り返し(Loop)と条件分岐(Alt)を示すフラグメントは、通常のメッセージを表す文よりも変換しにくい。そこで正確に変換するためには記述形式を指定した。指定形式は以下である。フラグメントの対象となる処理は、ユースケース記述の種類とメッセージの番号で指定した。

表 2　フラグメントの書式を指定

Loop	”ガード条件”まで基本フローx から y を繰り返す
Alt	”ガード条件 1”ならば基本系列 x へ、 ”ガード条件 2”なら例外系列 x～y へ

(6) 属性を注釈形式で補足

ユースケース記述内に全ての属性は書かれない。そこで不足する属性を、注釈という形式で行外で補足する。通常の処理文とは分けて書くので、メッセージへの変換に影響はない。記述形式は以下である。

「”クラス名”の属性は、属性 1、属性 2、…」

(7) 細かい書式制限

上記以外にも正確な構文解析や要素の抽出をしやすくするために、細かい文法制限を設けた。例えばカタカナ語や固有名詞などには「」を付ける、半角マイナスは構文解析の都合から使わないなどである。また文中に句点を含まず、主語を述語の直前に書いた方がより正確な解析が行えた。

上記(1)～(7)の制限に沿ったユースケース記述が表 3である。

表 3　改良したユースケース記述

1	学生は履修登録を開始する。
2	システムは前学期の GPA から履修登録の履修上限科目数を計算し、学生に履修登録の履修上限科目数を通知して、学生に時間割表を表示する
	学生の属性は、学籍番号とパスワードと、氏名、学部、入学年度
3	学生は履修する科目があるコマを選択する
4	システムは履修する科目があるコマの科目情報を学生に通知する
	履修する科目の属性は、科目 ID、科目名、単位数、開講期、備考
5	学生が履修する科目をシステムに選択すると、システムは履修する科目の科目名をデータベースに登録する
6	システムは履修登録の登録結果と履修する科目の科目名を学生に一覧表示する。
7	学生は履修する科目を全て登録するまで、基本系列 3～6 を繰り返す。
8	システムは学生を受講者名簿に登録して、学生に履修する科目を一覧表示する。
	受講者名簿の属性は、学籍番号科目 ID、作成年、学期
9	学生は履修する科目をシステムに確認する
10	学生はシステムにログアウトを要求する
11	学生がログアウトできたら基本系列 12 へ行き、ログアウト出来なかったなら基本系列 10 へ戻る
12	学生は履修登録を終了する

※背景色付が書式適用部分

5. 変換実験と考察

1.5. 実験の成果と課題

3節の変換手順及び0節の規則に従う変換ツールを作り、実際に文から図への変換実験を行った。その結果ユースケース記述から、図 8のシーケンス図と

図 9のクラス図が自動生成された。

変換に成功したのは、変換元であるユースケース記述の形式と文法を制限したからである。特に内部処理の記述とオブジェクト名の統一は、より正確な図に変換するのに大きな効果があった。内部処理の記述により、インターフェイス以外の処理もシーケンス図に盛り込まれた。またオブジェクト名の統一は同じクラスが重複して生成されるのを防いだ。

[1] の課題であったフラグメント処理も正確に変換できた。[1]では図上のフラグメントの開始位置と縦横の長さだけを指定していた。しかし今回の変換ツールでは従来の情報に加えて、メッセージの固有 ID をフラグメントの構成情報に追加した。その結果正確なフラグメントが生成でき、生成されたフラグメントをモデリングツール内で動かしても、メッセージが連動して移動した。

問題は、クラス図ではクラス間の関連性が変換されなかったことである。関連性の情報は変換元の文に記述されてないので、抽出できなかった。ドメインモデルなどの全く別の情報源から、人が手動で入力する必要がある。

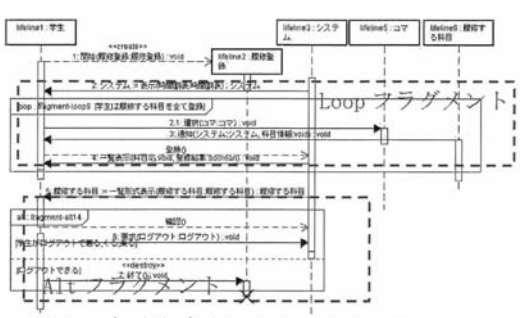

図 8　自動生成されたシーケンス図

図 9　自動生成されたクラス図

1.6. ユースケース記述のパタンと型の適用

ユースケース記述はユースケースレベルと、記述内の処理文レベルの 2 段階に分類できた。

ユースケースレベルでは、ログインとログアウト、検索の 2 種のパタンが見つかった。また CRUD(生成,検索,更新,削除)という 4 種へ分類できた。しかし UML 図（特にシーケンス図）に変換する際、種類に応じたひな型（図 10）を適用するという課題が残った。

ユースケース記述内の処理文は、起動処理、終了処理、通常処理の 3 種に分類できた。起動処理と終了処理は日本語文章に特徴が見られた。シーケンス図に変換する際、この特徴からメッセージの種類（Create, Destroy）を判別し、より詳しいシーケンス図が作成できた。

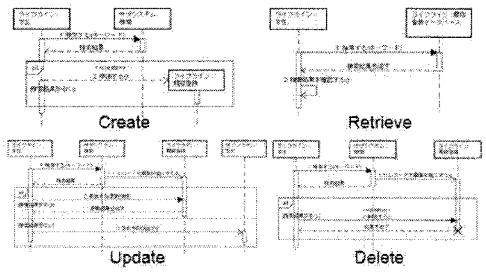

図 10　CRUD 別のシーケンス図の型

6. おわりに

変換ツールを作り、日本語ユースケース記述から UML 図の自動生成が可能になった。ユースケース記述に文法制限を設けることで、より正確な図の生成ができた。また UML 図へ変換する際に、ユースケースの種類及び各文の処理内容によって、一定のひな型を適用できるものと思われる。

参考文献

[1] 情報処理学会第 69 回全国大会, 4M-2, pp.325-326 (2007)

[2] 情報処理学会研究報告, 2005-NL-170, Vol.2005, No.117, pp.61-66 (2005)

[3] 情報処理学会研究報告, 144-14, Vol.2001, No.69, pp.99-104 (2001)

[4] 情報処理学会研究報告, 2003-NL-158, Vol.2003 No.108, pp.81-86 (2003)

[5] 情報処理学会研究報告, 132-11, Vol.1999, No.62, pp.81-86 (1999)

[6] 情報処理学会第 68 回全国大会, 3K-1 (2006)

[7] ダグ=ローゼンバーグ, マッド=ステファン, “ユースケース駆動型開発実践ガイド”, 株式会社翔泳社

[8] “Ko-BaKo/Japanese”, ㈱日本システムアプリケーション, http://www.jsa.co.jp/LANG/ko-bako/index_frame.htm

[9] “Jude Professional 5.0.2（モデルバージョン 25)”, ㈱チェンジビジョン, http://jude.change-vision.com/jude-web/index.html