ソフトウェア工学 137－7
（２００２．５．１０）

シナリオ記述言語の定義とソフトウェア開発への適用

藤本 宏，張 紅輝，大西 淳

立命館大学大学院 理工学研究科

ソフトウェア開発においてユーザの参加を促し，関係者の理解を共通のものとするためにシナリオを使用することは有効である．シナリオを使用する際には，シナリオをどのような形式で記述するのか，記述された情報をどうやって活用するのかが重要である．本研究ではエンドユーザでも簡単にシナリオを記述できるように，制限言語と簡単な制御文を用いてシナリオを記述する言語を定義した．そしてシナリオに記述された情報を有効に活用できるように，シナリオを機械的な処理に適した内部表現に変換する．またこの記述言語を用いたシナリオの作成と精製のプロセスを「図書館の貸出し業務」を例題に取り上げて説明する．

Definition of a Scenario Description Language and its Application to Software Development

Hiroshi Fujimoto, Honghai Zhang, Atsushi Ohnishi

Graduate School of Science and Engineering, Ritsumeikan University

Scenario has been widely spread and used in software development for end user's joining and for the communication between software developers and end users. The authors have defined a scenario description language with which an end user can easily describe his scenarios in software development. Since this language is based on the case grammar, a scenario written with this language can be automatically processed and transformed. Processes of making scenarios and tranfoming them will be illustrated with a small library system example.

1. はじめに

近年，システム設計・開発においてユーザの視点を取り入れる技術の重要性が増してきており，そのための手法としてシナリオの使用が注目を浴びている[1]．ソフトウェア開発にシナリオを用いることは要求を明確化し，より詳細な要求の引き出しやユーザと開発者の合意に役立つ．これによりユーザがソフトウェア開発に参加することが容易となり，ユーザにとってより良いソフトウェアの開発が可能となる．

欧州の CREWS（Cooperative Requirements Engineering With Scenario）プロジェクトでは，シナリオに基づく要求工学を実践した 4 カ国 15 の異なるプロジェクトを調査した．現地でのインタビューや，作成された文書・利用した環境やツールについての調査を行い，シナリオの意図や利用は予想以上に幅広く多岐に渡っているという結果が得られ，シナリオを用いた支援ツールと手法の構築が必要だとしている[2]．シナリオに基づいた設計・開発ではどのようにシナリオを表現するのか，どのようにシナリオを利用するのか，といった問題がある．またシナリオは作成するユーザの視点によって変化するため，膨大な量のシナリオが作成され管理も困難である．

そこで本研究では，誰もが容易に理解し記述できるシナリオ記述言語と，その言語で記述されたシナリオを機械的に処理・管理が可能な内部表現に変換する手法を提案する．シナリオの記述言語に関しては，構造化シナリオ記述[3]の提案がなされているが，本研究ではエンドユーザが記述しやすいよう，制限言語と簡単な制御文を用いてシナリオを記述可能な言語を定義する．

表１：格フレーム

動作概念	動作主格	源泉格	目標格	道具格	目的格	検索キー格
オブジェクトの流れ	○	○	○	△		
制御の流れ	○	○	○			
AND 木構造	○				○	
OR 木構造	○				○	
検索	○	○	○			○
生成	○	○	○			
更新	○	○	○			
削除	○	○	○			
依頼	○	○	○			
追加	○	○	○			

※○は必須格，△は任意格を示す

2. シナリオ記述言語

シナリオは，ユーザが目的を達成するためにシステムや他のユーザと協調して行う行動を記述したものである．本研究では，シナリオをユーザやシステムの振る舞いを表す制限言語で記述されたイベント文と，イベント間の時間的順序を表す制御文によって構造的に記述する言語規則を定義する．また階層的に記述できるようにするため，部分シナリオという概念を導入する．

2.1. イベントの記述

イベントは自然言語に近い制限言語で記述されているため，機械的な解析や変換が困難である．そこで我々がこれまでに開発した格文法[4]に基づいた要求フレームモデル，要求言語とその処理系[5][6]を用いて機械的な処理に適した内部表現に変換する．要求フレームモデルは要求仕様の枠組みを与えるものである．

本研究では要求フレームモデルに基づいた格フレームに基づいて記述されたイベント文を解析し内部表現に変換する．イベントはその振る舞いが示す動作概念と，動作概念に基づく格構造によって表現される．この動作概念と付随する格構造を定義したものが格フレームである．格フレームとして表１に示す 10 種の動作概念の格構造を定義している．もしシナリオ記述者が格フレームに予め定義されていない動作概念を使いたい場合は，新たに動作概念と格構造を格フレームとして定義し利用することが可能である．この言語で記述されたシナリオは，シナリオ中のイベント文を格フレームに基づいて解析し内部表現として保存される．

例をあげると，学生が図書館で本を借りる際の「司書に本を渡す．」というイベント文は，『渡す』という振る舞いから“物の流れ”という動作概念が導かれ，格フレームに定義された格構造より「動作主格」，「源泉格」，「目標格」，「道具格」の４つの格を持つものとして，格助詞を手がかりに解析される．このイベント文では主語が明示されていないが，このように欠落した格や代名詞がある場合は，格構造と文脈情報により補完を行う．

このイベント文は解析の結果，任意格の「道具格」がないとされ，他の格構造にイベントに含まれる情報が入り，図１の内部表現に変換される．

例にあげた方法では単文のイベントしか解析することができない．しかし単文の羅列は読みにくく，また書きにくい．そこで重文や複文で記述されたイベントに関しては，動詞に着目して単文に分割したうえで解析する[5]．これにより，より自然な文章でイベントを記述することを可能にしている．

```
(objectflow “渡す”  (agent  “本”)
                    (source  “学生”)
                    (goal  “司書”))
```

図１：内部表現の例

シナリオの作成に際して，エンドユーザが容易に理解し記述できるようにするためには，特殊な人工言語を用いるのではなく，自然言語やそれに近い形式でなるべく自由に記述可能であることが望ましい．しかしそうすると機械的な処理や解析が困難となり，シナリオに記述された情報を有効に活用することができない．

本研究で提案するシナリオ記述言語では，シナリオが持つ情報の大部分であるイベントを自然言語に近い制限言語で記述することが可能であり，イベント間の時間的順序に関しても簡単な制御文で記述できるため，誰もが記述でき，そこに記述された情報を読み取ることが可能だと考える．また記述されたイベント文を解析し格構造に基づいた内部表現として保存するため，機械的な処理で情報を解析し，必要に応じて変換や検証を行いシナリオに記述された情報を有効に活用することが可能である．

この内部表現は動作概念とその格構造に基づいてイベントに記述された情報が整理されているため表層表現に依存しない．そのため，先に示した図1の例をとると，「動作主格」にあたる「本」を主語とした日本文や，「目的格」にあたる「司書」を主語とした日本文を容易に生成できる．またイベントが能動態で記述されていても，受動態で記述されていても，意味が同じであれば同一の内部表現となる（図 2）．このことを利用して，記述されたシナリオを複数の視点から検証したり，複数の視点から記述された同一のシナリオを比較することで矛盾を発見することが可能である．

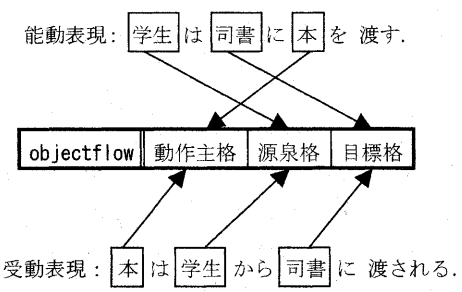

図 2：表層表現に非依存の内部表現

2.2. 部分シナリオ

シナリオの記述に際して，部分シナリオという概念を導入し，シナリオを階層的に記述できるようにする．実際のシナリオではシナリオ中のイベントの代わりに，既に作成された，あるいは今後作成されるであろうシナリオ名を記述する．シナリオを階層的に記述可能とすることで，記述者のシナリオ作成に関する労力を減少し，また作成の困難さを軽減できると考える．

シナリオ記述の労力の減少に関しては，エラー処理や認証といった複数のシナリオに登場し，かつ同様の振る舞いをするイベント郡を部分シナリオとして予め定義しておく，あるいは既存のシナリオから切り出しておく．このように予め用意しておいた部分シナリオを利用してシナリオを作成することで，シナリオ作成における工数を減少することができる．

シナリオ作成の困難さに関しては，業務の具体的な振る舞いが分からない場合に，その部分を抽象的な表現で記述可能にすることで軽減されると考える．シナリオは業務の振る舞いのうち，記述者が捉えうる範囲の情報を明確にする．しかし一連のシステムの振る舞いのうち，記述者以外のアクタが主に作業をする振る舞いに関しては，具体的な業務が分からないため，イベントを記述することが困難な場合がある．このような場合に，記述者が把握できない業務に関しては部分シナリオとして抽象的な表現のままにしておき，その業務に精通した人にその部分シナリオの内容を記述してもらう事で，困難なく正確なシナリオを作成することが可能となる．

2.3. イベント間の時間的順序と記述規則

イベント間の時間的順序は以下を想定する．

- 順接 (sequence)
- 選択 (selection)
- 繰り返し (iteration)
- 並行 AND 分岐 (and-fork)
- 並行 OR 分岐 (or-fork)
- 並行 XOR 分岐 (xor-fork)
- 並行同期 (join)

これらの時間的順序を制御文としてシナリオ中に明示する．しかしイベント間の時間的順序はその殆どが順接であるため，順接は明示しない．またシナリオの途中で目的が達成し得ない状況になった場合を表現するために終了を表す制御文でそれを明示する．そしてシナリオの記述し易さや，記述の労力の減少のために部分シナリオという概念を導入する．これに関しては，後に詳しく述べる．

この記述規則に則ったシナリオの BNF を図 3 に示す．以下に簡単な説明を加える

- ‘＊’，‘＃’ はシナリオ名の先頭
- ‘・’ はイベント文の先頭
- ‘end’ は終了
- ＜TITLE>はシナリオ名
- <EVENT>はイベント文
- <CONDITION>は条件文

```
<scenario>::='＊'<TITLE> '{' <content>* '}'

<content>::=<if_cont>
            |<while_cont>
            |<and_cont>
            |<or_cont>
            |<xor_cont>
            |('・'<EVENT>)*
            |<segment>
            |'end'

<if_cont>::='if' '('<CONDITION>')' 'then'
              <content> ('else'<content>)? 'fi'

<while_cont>::='while' '('<CONDITION>')'
                       'do'<content>'od'

<and_cont>::='AND-fork' ('・'<EVENT>)+'AND-join'

<or_cont>::='OR-fork' ('・'<EVENT>)+'OR-join'

<xor_cont>::='XOR-fork' ('・'<EVENT>)+'XOR-join'

<segment>::='＃'<TITLE>
```

図 3：シナリオ記述の BNF 定義

3. 視点変換を用いたシナリオ作成手法

シナリオはそれを記述する人の視点で作成され，その記述者が認知する範囲でのシステムやユーザの振る舞いといった情報が記述される．しかしシステムには多くの人間が関わるため，あるシステムの振る舞いに対して複数のユーザがそれぞれ別個に関わる場合は単一の視点で作成されたシナリオだけでは情報が不足する．そこである視点で記述されたシナリオを，他の視点で記述されたシナリオに変換し，その視点からシナリオの検証や補完，詳細化を行う．これをシステムに関連する全てのユーザの視点で行うことで，システムの振る舞いに関する十分な情報を持ったシナリオが作成できると考える．

この手法を，「図書館の貸出し業務のシナリオ」を例に挙げて紹介する．なおここでは何らかの図書館システムが稼動しており，司書がいる窓口に学生が本を持っていき，本の貸出し業務が行われるといった図書館を想定している．

3.1. 「学生」視点のシナリオ

最初に，システムの実際の利用者である学生の視点から作成されるシナリオを考える．ここでは学生が実際に本の貸出しを受ける際の行動をシナリオとして明確化してもらう．作成されるシナリオを図 4 に示す．

```
＊図書館の貸出し業務 {
・司書に貸出し手続きを頼む．
・司書に本と学生証を渡す．
＃貸出し手続き
・司書から本と学生証を受取る．
}
```

図 4：「学生」視点で作成したシナリオ

このシナリオでは，学生が直接システムとやり取りをしていないため，システムの振る舞いが分からない．また司書が行う貸出し手続きに関して，学生が認知できないため，それを部分シナリオとしている．このままでは情報が少なすぎるため，このシナリオは役に立たないようにも見えるが，しかし逆にいうと「図書館の貸出し業務」におけ

るユーザの一人である学生にとっては，現状ではこれだけの情報を持っていれば業務が利用できるということを示していると言える．

このシナリオを内部表現に変換し保存する．その際，各イベントの主語が抜けているため，業務の主体である学生を主語とみなして解析を行う．解析の結果，このシナリオは図5に示す内部表現に変換される．

```
(title "図書館の貸出し業務") {
(apply "頼む" (agent "貸出し手続き")
              (source "学生")
              (goal "司書"))
(objectflow "渡す" (agent "本，学生証")
                   (source "学生")
                   (goal "司書"))
(scenario "貸出し手続き")
(objectflow "受取る" (agent "本，学生証")
                     (source "司書")
                     (goal "学生"))
}
```

図5：シナリオの内部表現

3.2. 視点の変換

本の貸出し業務には学生以外にも司書というユーザがいる．そこで司書の視点でシナリオを検証し補完するために，学生視点で記述されたシナリオを司書視点で記述されたシナリオに変換する．変換後のシナリオを図6に示す．

ここでは学生視点のシナリオの内部表現を，それぞれのイベントに関して司書を主語とした日本語に変換する．

```
＊図書館の貸出し業務 {
・司書は学生から貸出し手続きを頼まれる．
・司書は学生から本と学生証を受取る．
＃貸出し手続き
・司書は学生に本と学生証を渡す．
}
```

図6：「司書」視点に変換したシナリオ

3.3. 「司書」視点でのシナリオの検討

学生の視点から司書の視点に変換したシナリオをもとに，司書の視点で業務を考えシナリオの検証と補完をしてもらう．司書が記述を加えた後のシナリオは図7のようになる．

```
＊図書館の貸出し業務 {
・司書は学生から貸出し手続きを頼まれる．
・司書は学生から本と学生証を受取る．
＃貸出し手続き
・学生に返却期日を伝える．
・司書は学生に本と学生証を渡す．
}

＊貸出し手続き {
・学生証をシステムに通し認証する．
while (貸出し手続きが終わっていない本がある)
do
    ・システムに貸出し登録を入力する．
od
}
```

図7：「司書」視点で記述を加えたシナリオ

司書が学生の記述したシナリオを検証することで，学生のシナリオでは抜けていた「学生に返却期日を伝える」というイベントが補完された．このイベントは学生の視点でも記述されているべきであり，これが抜けていたのは学生の誤りである．これは他のユーザの誤りの発見だといえる．また学生の視点では詳細が認知できないため部分シナリオとしていた「貸出し手続き」に関して，司書の視点ではその詳細が認知できるので，部分シナリオ「貸出し手続き」の内容が記述された．

3.4. シナリオの検討

ここまででこのシナリオに関わるユーザそれぞれの持つ情報が統合されたシナリオを記述することができた．ここで更にシナリオの質を向上させるために，このシナリオをもとに様々な状況を想定して議論を重ねることが有効である．その場合はシナリオの読みやすさのため，特定の視点で記述されたシナリオではなく，各イベントを能

動態に変換したシナリオが良いであろう.

この例の場合は, 認証や貸出し登録の際にエラーが発生すると想定されるため, エラー処理のイベントを追加する. ここでエラー処理に関しては, どちらの場合でもシステムが同じ振る舞いをすると考えられるため, 部分シナリオ「エラー処理」を作成し元のシナリオに追加する. 追加後のシナリオは図8のようになる.

```
＊図書館の貸出し業務 {
・学生は司書に貸出し手続きを頼む.
・学生は司書に本と学生証を渡す.
＃貸出し手続き
・司書は学生に返却期日を伝える.
・司書は学生に本と学生証を渡す.
}

＊貸出し手続き {
・司書は学生証をシステムに通し認証する.
if (認証が失敗)
    then ＃エラー処理
fi
while (貸出し手続きが終わっていない本がある)
do
    ・司書はシステムに貸出し登録を入力する.
    if (貸出し登録が失敗)
        then ＃エラー処理
    fi
od
}

＊エラー処理 {
・システムは画面にエラー原因を表示する.
・司書は学生にエラー原因を伝える.
・司書は学生に学生証を渡す.
end
}
```

図8：検討後のシナリオ

3.5. 手法の評価

シナリオの視点の変換により, 複数のユーザがそれぞれ似通ったイベントを記述する必要がなくなり, 全体としてシナリオ作成の労力を減少することができた. また他のユーザが記述したシナリオをそのまま読むのではなく, 自分の視点からのシナリオに変換したものを読むことで, 実際に自分の認知しているシステムの振る舞いと比較できるため, より効果的なシナリオの検証が可能であろう. そして複数のユーザがそれぞれに持っていたシステムの振る舞いに関する情報を一つのシナリオにまとめて記述することで, ユーザも含めたソフトウェア開発に関わる全員のシステムへの認識を明確かつ共通のものとすることができる.

このように提案する手法は, ユーザ要求の効率的な獲得や, 要求分析の支援に役立つ. また作成されるシナリオを通して関係者が共通の理解を構築することで, ソフトウェア開発を円滑に進めることができるようになると考える.

4. 研究プロジェクトの紹介

我々は, シナリオを用いたソフトウェア開発支援を目的として研究を進めており, 以下の研究を行っている.

- シナリオ記述言語の定義
  エンドユーザでも簡単に記述できる記述言語を定義し, その言語で記述されたシナリオを機械的処理に適した内部表現に変換する
- シナリオ作成支援手法
  シナリオ作成時の工数の減少や, ユーザからより詳細な要求を引き出すことを目的に, シナリオの視点の変換と異なる視点のシナリオの統合を行う
- 作成されたシナリオの検証手法
  シナリオのルールを予め定義しておき, 作成されたシナリオに関してそのルールを用いて検証することで, シナリオの矛盾や抜けを発見する

本稿では「シナリオ記述言語」と「視点の変換・統合」に関して述べた. ここで本稿では扱わなかった「シナリオのルールによる検証」に関して, 簡単な説明を挙げる.

○ シナリオのルールによる検証[7]
シナリオ中のイベント間の論理，時間関係をルールとして予め定義しておき，記述されたシナリオがこのルールを満たすかどうか調べることで，シナリオの時系列の矛盾や抜けを検出する．ルールとイベントの対応に，イベントの格フレームを利用することで，自動的に検証を行うことが可能である．

この他にも，「シナリオのアクティビティ図への変換」や，我々がこれまでに開発した「ビジュアルな要求仕様のアニメーション環境」を用いて本研究で扱うシナリオをアニメーションで視覚化することを検討している．

5. おわりに

本稿では，シナリオ記述言語，およびそれを用いたシナリオ作成手法に関して述べた．記述言語はエンドユーザでも簡単に記述できるように設計されており，またシナリオは機械的な処理が容易な形式に変換されるため，作成されたシナリオの検証や解析を支援できる．

またシナリオ作成手法の例として，実際に「図書館の貸出し業務」のシナリオを作成する手順を紹介した．この手法により，現状システムの明確化に関して，非専門家でも簡単に質の良いシナリオを作成することができることを示した．

今後の課題として，シナリオをより簡単に記述できるようにするため，完全に自然言語で記述されたシナリオを解析し，イベントと時間的順序を抽出することを目指している．

参考文献

[1] 郷 健太郎,John M.Carroll,今宮 淳美：ユーザの視点を取り入れる技術 システム開発におけるシナリオの役割，情報処理 Vol.41 No.1，pp.82-87 (2000)

[2] K.Weidenhaupt, K.Pohl, M.Jarke, and P.Hamer : Scenario Usage in System Development:A report on Current Practice, Proc. of ICRE Third International Conference on Requirements Engineering, pp.222 (1998)

[3] 谷 義人,満田 成紀,鯵坂 恒夫：ユースケースにおけるシナリオ記述の構造化と操作モデルの導出，情報処理学会研究報告 2000-SE-127-7 (2000)

[4] Fillmore,Charles.J. : The Case for Case; in Bach and Harms(Ed.), Universals in Linguistic Theory, pp1-88 (1968)

[5] 大西 淳：要求定義のためのコミュニケーションモデル，情報処理学会論文誌 33 巻 8 号,pp.1064-1071 (1992)

[6] A.Ohnishi : Software Requirements Specification Database Based on Requirements Frame Model, Proc. of the IEEE second International Conference on Requirements Engineering (ICRE'96), pp.221-228 (1996)

[7] 張 紅輝,大西 淳：シナリオの変換・統合とルールによる検証，情報処理学会研究報告 2000-SE-136-5 (2000)