ソフトウェア工学 127－7
（2000．5．26）

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

谷　義人† 満田成紀† 鯵坂恒夫†

システムを設計する上でユースケースのようにユーザからの要求記述から始めることは重要であるが、現状ではユースケースによるモデルがシステム内部設計に役立っているとは言えない。本論文ではこのユースケースによるモデルを有効利用することを目的とし、操作モデルの導出を行った。

機械的処理による導出を目的としたので、まず自然言語表記であるシナリオ記述を構造的に記述する規則を定義した。また構造化したシナリオから操作モデルを導出する規則を定義した。

導出の規則が機械的処理に適応していることを評価するためにC言語によるツールを試作し、クルーズコントロールシステムの例に適用することで評価を与えた。この結果、シナリオの構造化に必要な新たな要素と操作モデルへの導出規則に付加させるべきものがわかった。

Structured Scenario Description and Deriving Operation Models for Use-case Based Design

YOSHITO TANI,† NARUKI MITSUDA† and TSUNEO AJISAKA†

Software systems design should get started with requirements description written down from the endusers' viewpoint such as use-cases. The use-case model is however not enough utilzed for detailed design in real processes. This paper aims at effective use of use-case models by deriving operation models from them.

In order to derive operation models automatically, it is firstly needed to make use-case scenarios structured, and then the deriving rules are defined. An experimental tool is implemented to evaluate the structuring methods and deriving rules. According to some experiments on an example system description, several sorts of information are found additionally required to derive complete operation models.

1. はじめに

オブジェクト指向システムモデリングの方法の一つにユースケース[1]を用いた方法がある。エンドユーザの視点から要求を記述するので、非設計者であるユーザの意見をシステムに反映させることができると言う点で大きく注目されている。しかし現状ではせいぜい顧客と設計者のディスカッションの場で参考にされている程度であり、ユースケースを用いた方法がその後のシステム設計を行う上で大きく貢献しているとは考えがたい。これは、ユースケースにはクラス、オブジェクトに関する情報を含まれていないのが大きな要因である。とはいえ、新たにシステムを構築していくには、ユーザからの漠然とした要求のようにシステムの内部構造に関係の浅い情報から始めざるを得ないのも事実である。

そこで本研究では、ユースケースによる分析をより有効利用することを目的とし、ユースケース記述からソフトウェアの操作に関わる情報を導出する。ソフトウェアの操作情報というのは、ユーザインターフェースとソフトウェアの中核との間で行われる界面的な操作に関する情報であり、その後のシステムの振る舞いを決定するための枠組みになり得る。ユースケースによる分析は、ユーザとシステム間の相互の働きかけを明確にすることを目的としており、システムの外部との関わりを表しているということから、操作情報の導出に有効であると言える。

2. ユースケースのシナリオ記述

操作情報の導出のため4章で述べるようにユースケースのシナリオ記述を構造的に記述する規則が必要になるが、まずこのシナリオ記述について概説しておく。

† 和歌山大学システム工学部
Faculty of Systems Engineering,Wakayama University

シナリオとはユースケースにおける実行例をテキストで表記したもので、ユースケースのインスタンスである。シナリオでシステムのすべての機能を表すには、一つのユースケースに対し複数のシナリオを用意する必要がある。また、システムの正常なフローをメインフローといい、異常終了や例外的なフローのことを代替フロー（例外処理）という。

UMLにおけるユースケースのシナリオ記述には、以下の要素を含まなければならない。

ユースケースの目的　達成される目的。システムにおける各ユースケースの役割。

始動方法　どのアクターがどんな状況でユースケースを始動するか。

アクターとのメッセージフロー　アクター、ユースケース間で更新するメッセージやイベント。

ユースケースの代替フロー　例外的な処理の記述。

終了方法　終了するタイミングと、アクターに与える値。

3. ソフトウェアにおける操作情報

ユースケースから導出される操作情報についてソフトウェアを規定する意味の観点から説明する。操作情報を説明するためにソフトウェアの意味階層について先に述べる。

IEEE-CSのタスクフォースP1175によるCASEツール相互接続参照モデルでは、ツール間でやりとりされるソフトウェアの意味情報を次のように識別してる。[4)]

概念情報、関係情報　観点、表現モデルに依存しない、ソフトウェアが本来持つ概念的な要素と、それら要素間の関係を示す情報。

局面情報　実体-関連、データ/制御フロー、タイムシーケンス、状態遷移など、特定のモデリングにしたがう情報。

表現情報　局面情報を画面上やファイル上に表現する方法に関する情報。

これら3つの情報は、プロダクトに関する情報である。概念・関係情報、局面情報については、ユースケースによって表現される情報とは遠い情報なのでここでは詳しく述べない。表現情報はユーザインターフェースの部分が管理すべき情報であるが、これはプロダクト側からユーザへの一方的な情報に過ぎない。ユーザからプロダクトへの働きかけに反応するという、ソフトウェアの振る舞いに関する情報（プロセスに関する情報）が不足している。

これを補うために、ユーザからソフトウェアの中核

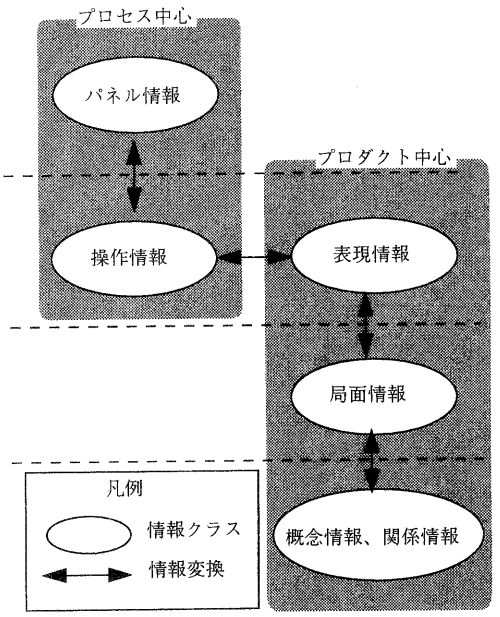

図1　ソフトウェアの意味情報階層
Fig. 1　hierarchy of semantic information

に向けて働きかける情報、**操作情報**を付け加えなければならない。例えばキーボードからの文字列を入力するアクションやマウスのクリックなどのイベントを管理するのが操作情報にあたる。これは、システムを外部からの操作という観点で記述するユースケースに含まれていると推測される。

この操作情報をさらに、ユーザがじかに接触する情報に成形するためには、もう一段の情報授受のしくみを明確にする必要がある。最終的なユーザへの表示や操作を受け入れるインタフェースは、ツールキットなどのソフトウェア資産として利用すべき部品を用いて作成されるのが現実的である。このユーザインタフェースの部品の使い方を指示する、ユーザに一番近い情報を**パネル情報**として整理する。これらの意味情報階層を図示したものを図1に示す。

操作情報は、パネル情報からの操作イベントやアクションを受けそれを表現情報に反映させる役割、及び表現情報をパネル情報に変換する役割を持っている。このシステムの振る舞いを表す情報を、ユースケースの持つイベントフローから導き出すのである。

4. シナリオ記述の構造化

従来のシナリオは各ユースケースに起こり得るイベントフローを自然言語で書き下すものである。これは、

```
description sinario
   :SYM_SINARIO '{' sinario_list '}';

sinario_list
   :flow_set
   |flow_set sinario_list;

flow_set
   :usecase name '{' flow_list '}';

flow_list
   :message_or_control ';'
   |message_or_control ';' flow_list;

message_or_control
   :message_flow_set
   |control_set

control_set
   :if_set
   |do_while_set
   |while_set
   |SYM_STOP;

if_set
   :SYM_IF '('condition')' SYM_THEN
                          '{'flow_list'}'
   |SYM_IF '('condition')' SYM_THEN
         '('flow_list'}' SYM_ELSE '('flow_list'}';

do_while_set
   :SYM_DO '{' flow_list '}'
                        SYM_WHILE '(' condition ')'

while_set
   :SYM_WHILE '('condition')' '{' flow_list '}'

message_flow_set
   :relation SYM_FROM argument SYM_INPUT
   |SYM_USECASE SYM_RUN
   |SYM_EXTRA SYM_RUN
   |relation SYM_WHERE argument SYM_OUTPUT;

relation
   :SYM_USECASE
   |SYM_ACTOR
   |SYM_EXTRA
   |SYM_RETURN;
```

図 2　構造化シナリオ記述の BNF 定義
Fig. 2　BNF of structuralized scenario description

エンドユーザからの要求を獲得するためだが、計算機による機械的処理での情報の変換や、半自動的に新たなモデルを引き出す作業を行うに、は自然言語のままでは意味解析が難しい。そこで、シナリオ記述に示されている情報を損なうことなく、かつ自動変換を可能にするための記述規則をもうけることにする。これにより、今までは単にシステムの概要の理解にしか役立っていなかったユースケースによる記述を、後のシステム設計に直接利用しようというのが本研究の主旨である。構造化されたシナリオの BNF を図 2 に示す。

4.1　メッセージフロー

シナリオに記述されているのはシステム内部とシステム外部のイベントの流れである。UML ではこれをメッセージフローとして扱っている。例えば「マウスがクリックされた」というイベントは、「ユーザからシステムに対し『マウスクリック』というメッセージが出力された。」と言い換えることができる。本研究の構造化規則もこれに従い、すべてのイベントの発生と観測をメッセージの入出力として扱う。

ここで重要なのは、外部アクターとシステムとのメッセージフローを記述することであってシステム内部の処理は記述すべきではないことである。ユースケースはシステムを外面からとらえたモデリング技法だからである。当然シナリオにおいても、必要以上に内部同士のメッセージフローを記述するべきではない。記述する必要がある場合は、ユースケース図に使用、拡張関係として現れるものだけであるといえる。

ユースケース	U:ユースケース名
アクター	AC:アクター名
例外処理	E:例外処理名
変数	V:変数名
定数	S:定数名
メッセージ	「文字列」

表 1　メッセージフローで使用されるタグ
Table 1　tag used message flow

メッセージフローの関連先や送受される情報には、構造化したシナリオを計算機上で字句解析する際にユースケース、アクター、変数、定数等を区別するためにタグをつける。使用するタグは表 4.1 に示すものである。

ユースケースにおいて記述されているのはシステムの外部要素や他のユースケースとの間のメッセージフローである。よって、各ユースケースが関連、関係する可能性のある要素は外部アクター、他のユースケースもしくは代替処理（例外処理）、実行中のユースケースを呼び出した呼出元の 3 つであるといえる。メッセージフローの関連先として記述できるのもこの 3 つに限定している。

呼出元の記述にはそのまま「呼出元」という予約語を用意した。送受信するメッセージはテキストによるメッセージの他、変数も送受信できるように定義した。これは将来的に内部設計段階において変数として扱われると予想されるものをあらかじめ区別しておけば、後々の設計に役立つからである。ユースケース名、アクター名等の各名前、メッセージの文字列はアルファベットで記述する。これは、字句解析の際に日本語表記では区別しにくいからである。これらの日本語表記は今後の課題とも言える。

4.2　ユースケース間の実行

あるユースケースが他のユースケースや代価フローを実行する場合は特に「U:USECASE1(E:EXTRA1)を実行する。」のように記述することにする。これは、後述の操作モデルとして状態遷移モデルを導出する際に、他のメッセージフローの文とユースケースの実行文を区別したいからである。このことは 5 章で詳しく述べる。シナリオ記述文として記述できるのは他の要素の実行、メッセージの受信、メッセージの送信の 3

```
"USECASE1"{
  AC:ACTOR から 「start」 受け取る。;
  U:USECASE2 を実行。;
  U:USECASE3 を実行。;
  ユースケース終了。;
}
```

図 3 構造化シナリオの記述例
Fig. 3 an example of scenario discription

通りとする。

各ユースケースのメッセージフロー群はユースケース毎に「{ }」で囲み、ダブルクォーテーションで囲んだユースケースの名前を前に付ける。メッセージフローの区切りはセミコロンで区切る。ユースケース毎のメッセージフローを集めて「{ }」でくくったものをシステムのシナリオとする。例外処理に関しても、通常処理のユースケースと区別はしているが、ユースケースと同じように例外処理毎のメッセージフローを記述し、例外処理が起ると思われるタイミングでメインフローから実行する。

ユースケースには始動点と終了点を明記する必要があるのは前章で述べた。本研究での構造化規則において、始動点は始動するアクターからの特定のメッセージ受信として始動点を記述する。終了点はメッセージフロー中に終了であることを明記することにする。

図 3 に簡単な記述例を示す。

4.3 制御構文

シナリオはユースケースのインスタンスであると言うことは既に述べた。しかし、機械的な処理でユースケースの変換を行うには、一つのユースケースに対し一意に決まるシナリオが必要である。ここでは一つのユースケースには、一つのシナリオを持って表現することにする。シナリオが複数必要な理由は、シナリオはユースケースにおけるある一つの実行経路に過ぎないからである。これをユースケースと一対一に定義するために、シナリオ記述に if、while 等の制御構文を用いる。これにより一つのシナリオ内に考え得る実行経路のパターンを収めることができる。以下に規則として定義した制御構文とその記述方法を列挙する。

if-else 文 条件によるフローの分岐と、条件外の処理をわけて記述する。

もし (条件式) なら { 分岐後フロー }

else 文は以下のように記述する。

もし (条件式) なら { 分岐後フロー } その他 { 条件外フロー }

while 文 繰り返し処理を記述するのに用いる。

繰り返す (条件式) { 繰り返しフロー }

do-while 文 繰り返し処理を記述するのに用いる。システムにおいて同一の処理を一回以上行う場合というのはよくあることなので、while とは別に用意した。

行う { 繰り返しフロー } 繰り返す (条件式)

各制御文の条件式は変数同士の比較、変数と定数の比較の形で記述する。分岐、繰り返し条件をシステム内部にとってより明確なものにするためである。比較演算子は「=、<、>、⊂、⊃、!」の 6 種類を定義している。集合包含関係を定義したのはユースケースのシナリオを考えている段階では未だ抽象的な式としてしか表せない場合もあるからである。また、条件式は空文でも良い。処理によっては無条件で繰り返す場合なども考えられるからである。

ユースケースが有用な点として、ユーザに理解しやすいというのは既に述べた。これはシナリオの問題点として挙げた、自然言語によって記述されていることが要因の一つである。これを受けて本研究における構造化規則でも、なるべく自然言語 (日本語) による表記を残したまま構造化することを試みた。しかし後で述べる、構造化したシナリオの計算機による言語解析をする際に、日本語のままでは字句解析が難しい部分があったので、一部アルファベットによる記述にしている。

5. 操作モデルの導出

操作情報とは、ユーザインタフェースを定義する情報の中で、システムの振る舞いを直接的に表現するものである。一方、ユースケースに記述されている要求記述もシステムの表層の振る舞いである。したがってユースケースのシナリオから操作情報が導出できると考えられる。本研究では前述の構造化したシナリオ記述より操作情報を導出する。

5.1 状態遷移モデル

操作情報とはユーザやアプリケーションの中核に対する働きかけを表す情報で、ユーザインタフェースにおける操作論理 (プロセス) を定義するものである [4]。操作情報はプロセスの動作を示す情報であるため、定義するには動的な要素に重点をおく必要がある。一般的にシステムの動的な側面を表現するのに用いられているのは状態遷移モデルである。ここでも状態とイベントという概念で、操作情報の表現を行うことにする。操作情報として定義すべき要素は以下のものである。

状態 プロセスの実行状況を記号化したもの。

遷移 状態の順序関係を表す。(前状態、遷移、次状態) の 3 つによってプロセスの実行を意味する。

操作イベント　入出力操作の「時点」を表す。遷移の起る時点を表すことによって入出力操作とプロセス実行との間の関係を明確にする。

操作パラメータ　入出力操作においてユーザとの間でやりとりされる値。操作イベントに関係付けられる。

5.2 変換規則

構造化されたシナリオから状態遷移モデルを導出するのにはまず、シナリオ記述中で状態が遷移する時点を読み取る必要がある。シナリオ記述中では、以下の3つの時点で状態が遷移していると考えられる。

ユースケースの実行　他のユースケースを実行することは、制御の焦点が実行中のユースケースから他のユースケースへ移るということである。これはユースケースに現れるある機能から別の機能へと制御が移っていることであり、操作情報においても状態が遷移していると言える。

if文による分岐　if文によってイベントの流れが分岐する、すなわちフローが分れると、操作情報においてもフローごとに遷移先の状態が異なっていると言える。

while文による繰り返し　while文による繰り返しでは、繰り返し部分の終点から繰り返しの始点へ状態が遷移していると考えられる。繰り返し部分において状態の遷移が無い場合は、同じ状態間での遷移があると考えられる。do-while文においても同じことが言える。while文との違いは遷移のタイミングの違いである。

シナリオ記述中にこれらの制御点が現れると、遷移後の新たな状態を用意し遷移前の状態からの遷移を確定する。このとき、状態に対しての明確な名前は定まらない。シナリオ記述では、操作過程での状態遷移は確定できるがその状態がどういう状態かを説明するような記述は含まれていないからである。

シナリオ中のメッセージフローは、状態遷移モデルの中のイベントにあたると考えられる。メッセージ受信は入力イベント、メッセージ送信は出力イベントとして考えられる。状態が遷移してから次の状態への遷移が現れるまでの間のメッセージフローをその遷移間の操作イベントして関係付けることにする。メッセージフローで送受されるメッセージ、変数は操作パラメータにあたる。また制御構文中の分岐、繰り返し条件も操作イベントの入力イベントにあたると考えられる。これは状態が遷移するときの条件にあたる。図4に変換例と導出された状態遷移図を示す。

5.3 導出ツールの試作

本研究では上記の変換を機械的に行うために、構造化したシナリオ記述から操作モデルを導出するツールを試作した。[5][6]。このツールは、定義した構造化シナリオを読み取り、操作モデルとして4.1節で説明した状態遷移の文法に基づいたテキストを出力するものである。以下にユースケースの各要素に対応した出力処理を説明する。

始点　シナリオのユースケース毎の始まりに始点である「start」を出力。

終点　ユースケース毎の終わり、またはシナリオ中の「ユースケースの終了。」を読み取り、終点である「end」を出力。

メッセージフロー　メッセージの入出力を読み取り、入出力イベントとして一時的に変数に保存する。

他のユースケースの実行　他のユースケースの実行文を読み取り、新しい状態と、前の状態からの遷移を出力する。

while文、do-while文　繰り返しの終りから繰り返しの始めへの状態遷移を出力する。また条件による繰り返しから出る分岐も出力する。このとき条件を入力イベントとして出力する。

if-else文　条件による分岐への遷移と、分岐外への遷移を出力する。条件を入力イベントとするのはwhile文、do-while文と同じ。

また、メッセージフローに関しては前節で説明した状態の遷移が起るごとに遷移に関係付けて出力する。次節でツールによる導出例を挙げる。

5.4 導出例

この節では、シナリオ記述から操作情報モデルの導出としてクルーズコントロールシステムの例を挙げる。クルーズコントロールシステムとは自動車の速度を指定速度に保つためのシステムで、シナリオ上に明確な時系列が現れにくい例として適用した。

5.4.1 クルーズコントロールシステムへの適用例

クルーズコントロールシステムのユースケースと要求記述を以下に列挙する。

始動・終了　ユーザがエンジンをかけると共にシステムを起動。エンジンが切られるとシステム終了。

速度の調節　車の走行速度をユーザが任意に指定する速度に保つ。

速度を上げる　指定速度より、走行速度が遅いときに速度を上げる。

速度を下げる　指定速度より、走行速度が早いときに速度を下げる。

速度測定　速度を調節するために現速度を測定する。

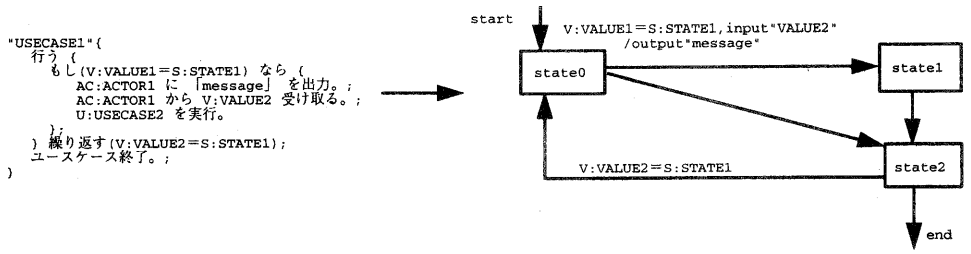

図 4　シナリオから状態遷移への変換例
Fig. 4　trans example

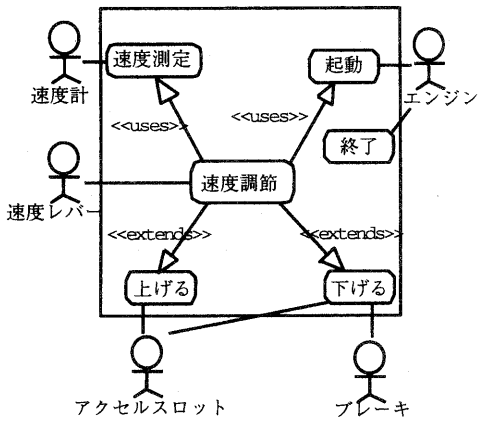

図 5　クルーズコントロールシステムのユースケースモデル
Fig. 5　use-case model : cruise control system

アクターとして考えられるのは、エンジン、アクセルスロット、ブレーキ、速度計、速度設定レバー、である。ユーザであるドライバーは直接的にシステムに触れるわけではなく、上記のアクターを通してシステムとやりとりをするのでアクターには入らない。図 5 はこれをもとにモデリングした図である。

次に具体例であるシナリオを記述する。

1. ドライバーが車のエンジンをかけると共に、クルーズコントロールシステムが起動。
2. ドライバーが、速度設定レバーで速度の設定をする。
3. 現速度よりも目標速度の方が高いので加速。
4. 途中で、高速道路に入ったので、設定速度を変更。
5. 高速道路を降りたので、設定速度を元に戻す。
6. 目的地に到着、エンジンを切ると、クルーズコントロールシステムも終了。

このシナリオ記述はユースケースのインスタンスとしてのシナリオである。上記の例はシステムの実行経路の一つに過ぎない。現実的な作業としてシナリオを記述するにはもっとたくさんの実行経路を考える必要があるが、ここではシステムの構造自体は問題ではないので省く。これらをもとに構造化シナリオを記述する。構造化したシナリオ記述と、ツールにより導出された操作情報、導出されたテキストをもとにダイアグラムで表したものをを図 6 に示す。

各ユースケース毎に遷移モデルが導出されるが、ユースケース間の関係によっては同一の図中でつなげることのできないものもある。例えば「end」というユースケースは他のどのユースケースが実行されている場合であっても条件を満たせばシステムを強制的に終了させる。このようにユースケース間に優先順位がある場合は同一の図中でつながらない。このようなユースケース間の遷移モデルを関係付けるためには、優先順位を反映させるため階層的な構造を用いる必要がある。またダイアグラム中の遷移には、終点を表す「end」にたどりつかない遷移があってはならない。これはユースケースには必ず始点と終点を定める必要があるからである。ユースケースによっては状態が現れず始点から終点への遷移の間に操作イベントが並ぶだけのものもある。これは、そのユースケースが実行状態には影響しない操作イベントしかやりとりしないからである。

6. 考　　察

適用例の導出結果をもとにシナリオの構造化と操作モデルの導出について考察をする。

6.1 構造化文法の考察

本研究での操作モデルの導出では、定義した操作情報での状態・遷移・操作イベント・イベントパラメータはほぼ導出できた。しかし、計算機での機械的導出のため状態の名前は登場順に番号を割り付けただけのものである。また操作イベントも、メッセージ入出力に対し、output・input の区別しかない。これらは、以後の設計を機械的に行う上では問題ではないのであるが、ユースケース記述の特徴でもあるエンドユーザ

```
"control"{
   繰り返す(V:engine=S:ON) {
      U:speed を実行。;
      U:speed から V:speednow 受け取る。;
      AC:lebar に 「decidespeed」 を出力。;
      AC:lebar から V:desirespeed 受け取る。;
      もし (V:speednow < V:desirespeed) なら {
         U:speedup を実行。;
      } その他 {
         もし(V:speednow>V:desirespeed) なら {
                              U:speeddown を実行。;
         };
      };
      U:speed を実行。;
      U:speed から V:speednow 受け取る。;
   };
  ユースケース終了。;
}
```

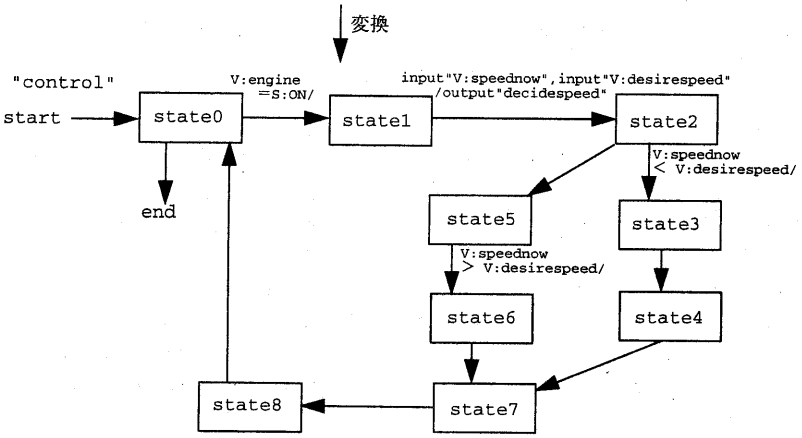

図 6 クルーズコントロールシステムの構造化シナリオ
Fig. 6 structuralized scenario description for cruise control system

が理解しやすいと言う点では問題がないとは言いきれない。

システムの外面的な振る舞いとしてユーザに理解しやすいモデルであるのが理想である。しかしこれらの情報、状態名や操作イベントの種類に関してはシナリオ記述には本来含まれていないものである。シナリオは個々の機能におけるイベントフローに過ぎないので、状態に関する情報や操作の種類などは記述に含まれていない。導出後のモデルに状態名やイベントの種類を反映させるにはシナリオ記述の時点で、ある程度導出後を見越して情報を盛り込まなければならない。

例えば状態名に関して言えば、シナリオ記述の時点で状態が遷移すると思われる点に特定のラベルを記述しておいてそのラベル名を用いる等の方法が考えらる。ラベル以上に状態数が導出される時はラベルに番号を振ってやると、通し番号のみよりは理解がしやすい。

イベントの種類に関しては、構造化の文法にメッセージ入出力だけでなくある程度のキーワードを用意して、入出力と関連させて記述させる方法が考えられる。例えば以下のように記述する。

AC:lebar から V:desirespeed Trigger 受け取る。;

ここでのキーワード「Trigger」は、そのイベントが瞬間的なものであることを表す。メッセージの種類を瞬間的な信号・断続的処理を始める・断続的処理をやめるの三種類に分け、入出力だけでなく 3 種類のメッセージを入出力すると分類すると、導出後のモデルでの操作イベントでも入出力の 2 種類よりもより理解しやすいのではないだろうか。またメッセージの種類が上記のようにわかれていることで、後の設計にも役立てることができる。

6.2 変換規則の考察

操作情報としての状態遷移モデルはシステムの外面的な振る舞いを示すべきものである。操作情報の要素として状態・遷移・操作イベント・操作パラメータを導出したのだが、遷移に関連付けられている操作イベントで、外面的振る舞いに関するイベントだけでなくシステム内部でやりとりされるであろうものも含まれている部分がある。理由として考えられるのは、変換規則においてすべてのメッセージ入出力を同列のイベントとして扱っている点であると思われる。個々の

ユースケースにとっては変換規則のように入出力として同列なのだが、システム内部で行われるやりとりと、ユーザとの間でやりとりされるものは切り離して考えるべきである。

ユースケースを用いたモデリングにおいて重要なのは、ユーザとシステムのやりとりであって、導出すべき操作情報もユーザとシステム間のプロセスに関するものであるからである。システムの要素として考えられるものの中から、ユーザとの操作において必要であると思われるものだけを選びだし設計するための変換規則であるべきであるが、今回の変換規則ではそれに付随してシステム内部に関する情報も同時に導出してしまうようになっている。

これを改善するには、変換規則において、メッセージフローをシステム内部のものか外部とやりとりされるものかを区別して、外部とのメッセージフローのみを遷移に関連づけるべきである。構造化文法ではメッセージ送受の関連先を区別しているので、関連先によってシステム外部とのやりとりであるか内部でのやりとりであるかは判別できる。これをもとに遷移に関連づけるかどうかを判別するようにすればよい。

6.3 ユースケース間の関係

今回は適用例で動的システムの例としてクルーズコントロールシステムを、事務処理系の例として履修登録システムを適用した。履修登録システムの例では、ユースケース同士の実行順序はシナリオに登場する順序として表されているので大きな問題はない。しかしクルーズコントロールシステムでは順序を特定できないものがある。

クルーズコントロールシステムでは、エンジンが切られるとシステムも終了すると要求記述されている。この終了は処理の流れで終了になるのではなく、システムがどの状態でもエンジンが切られると強制的にシステムを終了するというものである。このため個々のユースケースのシナリオに記述することが難しいので、適用例でのモデリングでは「終了」というユースケースを別に用意して対処した。このとき「終了」というユースケースと他のユースケースの間には明らかに優先順序が存在する。これを明確に導出モデルへ反映させるためには、導出されたユースケース毎の状態遷移モデルを階層的に記述するのが望ましい。そのためには、ユースケース記述の段階で、ユースケース間の優先順序に関する情報を含ませてやる必要がある。しかしUMLでのユースケースの定義にはこういったユースケース間の順序関係に関する記述の定義がなされていない。新たにユースケース間の関係に関する定義をする必要があるのではないだろうか。

7. おわりに

本研究ではユーザからの要求記述であるユースケースをシステム設計に役立てることを目的としてきた。そのためユースケースにおけるシナリオ記述の構造的記述を定義した。構造化のポイントは、イベントをメッセージフローとして定義したことと、制御構造により複数のシナリオを一本化したことである。この構造化したシナリオ記述からシステムの外面的振る舞いを表す操作情報として状態遷移モデルの導出を考え、変換への規則と手順を定義した。この変換規則に基づき、機械的にモデルの導出を行うツールを作成し、例に適用して評価した。

今後の課題は導出後の状態遷移モデルを操作情報として完全な情報を含むモデルにすることである。このため考察で述べたような不足している情報を何らかの形で補う必要がある。この情報はユースケースでの記述には含まれていない情報なので、ユースケース記述を拡張する方法や、他のシステムモデリングの記述と組み合わせる方法等が考えられる。

また導出された操作情報はミドルウェアを利用したユーザインタフェースの設計に利用できる。操作情報はユーザインタフェースの振る舞いを決定するための中心的存在となる。これはユーザインタフェースを構成するためのパネル情報や表現情報は操作情報と整合するものでなければならず、これらの設計法も提案する必要がある。

参考文献

1) Hans-Erik Eriksson・Magnus Penter 著, 杉本宣男・落合修・武田多美子 訳:UML ガイドブック. 株式会社トッパン,1998.
2) オージス総研 著, 千籐雅弘 監修:かんたん UML. 株式会社 翔泳社,1999.
3) 中田育男 著:コンパイラ. 産業図書株式会社,1981.
4) 満田成紀・鯵坂恒夫. シナリオ定義に基づくユーザインタフェース設計法. ソフトウェアシンポジウム'99 論文集,pp156-164,1999.
5) M.T.Skinner 著, 春木良且 訳:C++基礎講座. 株式会社 インプレス,1994.
6) Levine John.R・Mason Tony・Brown Doug 著, 村上列 訳:lex&yacc プログラミング. アスキー,1994.