状態遷移設計の基礎

神奈川大学 理学部 情報科学科
野口 健一郎

1．はじめに

初めはハードウェア設計用の技法だった状態遷移図や状態遷移表は、通信用ソフトウェアの設計にも広く用いられてきた。また、OS の設計への適用も試みられ[1]，その後さらに広い分野のソフトウェア設計にも適用されるようになった。[2] [3]

ここでは、状態遷移技法を用いた設計、すなわち状態遷移設計の基礎を、やや理論的に述べてみたい。それにより、状態遷移設計の特性が分かり、またその有用性がより明確になることを期待する。

2．外部仕様と状態遷移技法

設計の重要な過程として、外部仕様を明確に、漏れなく決め、またそれをきちんと表現することが必要である。なお、外部仕様には

- ***入力データと出力データの形式（構文規則＝シンタクス）***
- ***入力データと出力データの関係（意味規則＝セマンティクス）***

の二つの側面があることに注意しよう。

外部仕様を考え、記述するための手法をみてゆく。

（1）シーケンス図

外部仕様を考え、記述するとき、内部の構造、設計に立ち入らずにできることが望ましい。外部仕様を最も「外部的」に記述する手法として**シーケンス図（時系列図）**がある。[2]

これは設計対象物とその利用側とをそれぞれ縦棒で表現し、その間でのデータのやり取りを時系列的に表現したものである。（図１）

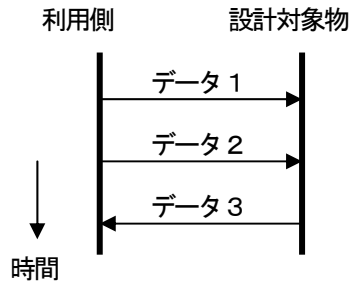

図１　シーケンス図

シーケンス図の利点は、

- ***入力と出力の関係、すなわちセマンティクスが、内部の動作・処理に関係することなく、わかりやすく表現される。***

しかし、根本的な欠点として、

- ***シーケンス図で記述できるのは有限個のシーケンスのみである。無限のシーケンスを持つ外部仕様（普通はそうである）のすべてを記述することは、理論的にできない。***

このように限界はあるものの、

- ***（シーケンスの種類が無限個であっても）シーケンス図を用いて外部仕様の主要なシーケンスを記述することは、理解を助け、特に初期の設計には有用である。***

（2）状態遷移図（表）

コンピュータの大きな特徴は記憶を持つことである。設計対象物が記憶を持つとき、入出力のシーケンスの種類は理論的には無限個になる。そこで、**内部状態**の概念を導入することにより、無限の入出力シーケンスを有限の表現として記述できるよう

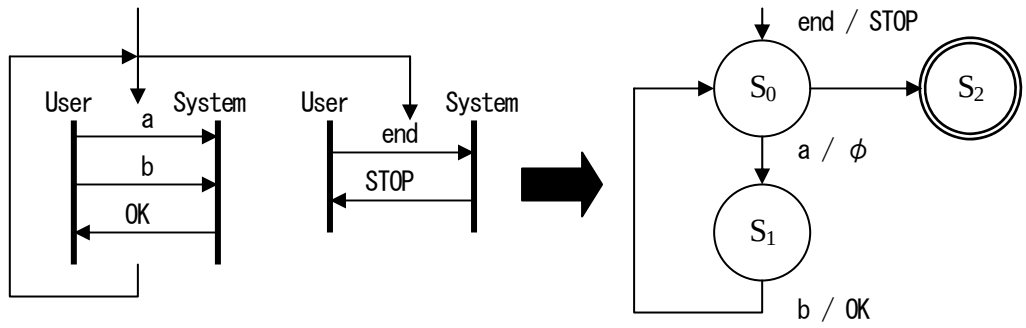

図２　繰返し型シーケンスを状態遷移図で記述

にしたものが**状態遷移図**（または**状態遷移表**）である(図２)。ただし、これで表現できるものは理論的には**有限状態機械**であり、内部状態数が有限個である、という制約がある。

状態遷移図の特徴を挙げる。

- *シーケンスの種類が無限個であっても、シーケンスに繰返し型の規則がある場合に適用でき、入出力の関係、すなわちセマンティクスがきちんと記述できる。*
  - *図であるため、個々のケースごとにそれにふさわしい構造的な表現をとることが可能である。外部仕様の構造について理解しやすく、外部仕様を考え、設計する道具として有用である。シーケンス図の次に使う道具としてよい。*
  - *内部状態数が無限の場合、または非常に多い場合には適用できない。ただし、主要な外部仕様は限られた数の内部状態で扱える場合も多く、その場合には適用できる。*

状態遷移表の特徴を挙げる。

- *状態遷移図の第１点と同じ。*
- *表であるため、内部状態と入力を組み合わせたすべてのケースについての動作が記述できる。したがって、外部仕様の細部まで明確になる。細部にわたる外部仕様を考え、設計する道具として有用である。*
- *状態遷移図の第３点と同様。さらに、次節で述べる拡張状態遷移表を用いることにより、これがやりやすい。*

３．内部設計と拡張状態遷移表

内部状態数が無限の場合、状態遷移図や表で完全な外部仕様を記述することはできない。また、内部状態数が非常に多い場合も、実用上だめである。記述できないのは外部仕様のうちのセマンティクスである。**これらの場合には、外部仕様を完全に記述するには、内部設計を進めて、セマンティクスを「内部的」に記述するしかない。**すなわち、外部仕様設計と内部設計が同時に進むことになる。

内部状態数を、実用上取り扱える範囲にまで減らす手法は二つある。一つは、設計対象物を複数の部分に分割することである。各部分ごとに、その部分の外部仕様を状態遷移技法で表す。10,000 個の内部状態を持つものをうまく２分割できた場合、各部分はたかだか100 状態ですむ。なお、複数の部分への分割に伴い、もともとの外部仕様には無かった各部分間の入出力(内部入力、内部出力)が出てくる。

もう一つの手法は、変数（内部変数）の導入である。１ビットの変数（フラグ）の導入により、うまい場合は状態数が半減する。４桁の１０進数の導入で、うまい場合は状態数が１万分の１に減る。

変数の導入により、状態遷移表は次のように拡張される。

- *出力を記述する部分が、変数の操作も含んだものとなる。*
- *状態遷移の判断に、変数の値の判定が必要に*

なる。（もともと多数の内部状態があったものを変数に縮退したのだから、これは当然である。）

このように変数を扱えるように拡張した状態遷移表を**拡張状態遷移表**と呼ぶ[2]（表1）。なお、変数は外部仕様には直接現れない、「内部的」なものである。

以上の二つの手法を組み合わせて用いることにより、内部状態数を実用上取り扱える範囲にまで減らすことができ（一つにまでなってもかまわない）、外部仕様を隅々まで明確にし、完全に記述することができる。それと同時に、内部設計のかなりの部分も進んだことになる。

この段階までの設計は、原理的には設計対象物がハードウェアかソフトウェアかには関係しない。ソフトウェアで実現するためには、次の設計段階として、状態遷移表ないしは拡張状態遷移表を、プログラムへと変換（理論的にはシーケンシャル処理へと変換）する作業が必要になる。[2]

4．オブジェクト指向＋状態遷移設計

最後に、近頃広まってきたオブジェクト指向に基づく設計と状態遷移設計の関係に触れたい。オブジェクト指向の考えはかなり以前からあるが、オブジェクト指向言語としてC++が広まり、またJava言語が登場するに至り、主流になりつつある。

オブジェクト指向の基本は、システムを複数のオブジェクトに分割し、かつオブジェクト間のインタフェースを明確にする、というシステム構成法の面にある。これは妥当な考え方である。しかし、オブジェクト指向の場合、オブジェクト間のインタフェース（それはオブジェクトの外部仕様である）としてシンタクスだけに注目し、セマンティクスは無視する傾向にある。（例えば、Java言語のinterfaceや、分散オブジェクト用のインタフェース定義言語（IDL）など。）

設計を進めるためには、むしろセマンティクスのほうが重要であり、またそれを完全に定義することの難しさは前節までの説明で理解されたと思う。そして、インタフェースのセマンティクスを記述する手法として、状態遷移技法が適用できることにも気付かれるだろう。すなわちオブジェクト指向設計と状態遷移設計は、組み合わせて使うことができるものである。

表1　拡張状態遷移表（一つの記述形式）

入力＼内部状態 / 条件		・・・	S_i	・・・
：				
I_a				
I_h	変数の値の判定１		変数の操作 出力 → S_j	
	値の判定２			
：				

オブジェクト指向設計のためのモデリング技法として UML (Unified Modeling Language) が注目されてきている。その中には、sequence diagram (シーケンス図) や statechart diagram (状態遷移図) も含まれている[4]。それらの必要性および位置付けを、本稿の読者は容易に理解されよう。

5. おわりに

本稿では状態遷移設計の基礎をやや理論的に説明することを試みた。その背景となる理論はオートマトン理論である。状態遷移設計は理論的な基礎がしっかりしており、かつ実用性もあるものとして、着実に使われていこう。オブジェクト指向設計とも矛盾することなく、組み合わせて使えるのは、理論的な基礎がしっかりしていることの証ともいえよう。

参考文献

[1] 野口健一郎、元岡達:オペレーティング・システムの記述に関する一考察、情報処理、Vol.14, No.2, pp.98-105 (1973)

[2] 野口健一郎:ソフトウェアの論理的設計法、共立出版 (1990)

[3] 渡辺政彦:拡張階層化状態遷移表設計手法 Ver.2.0、東銀座出版社(1998)

[4] J. Rumbaugh, I. Jacobson, and G. Booch: The Unified Modeling Language Reference Manual, Addison-Wesley (1999)

(のぐち　けんいちろう)