επιστημηの

オブジェクト指向的夜話

C#から見た.NET Framework

第3回 有限状態機械生成機

επιστημη
えぴすてーめー

Technology Tools

- ❑ Visual Basic .NET
- ☑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ❑ ASP.NET
- ❑ Internet Information Services
- ☑ Other:
  - Visual C++ .NET
  - NUnit v2.1

Level

★★★☆☆

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、付録CD-ROMの¥DOTNET¥YAWAディレクトリに収録しています。

¥EPISTEME
今回取り上げたサンプル

おさらい：ゲートのシミュレーション

ちょいと失礼して前回の「書き出し」をそのまま引用させていただきます。

前回の「書き出し」

オブジェクト指向の教材として、少し前に簡単なコードをC++/Javaで書きました。お題は“ゲート”です。近所の大きな病院の駐車場にあるもので、見舞い客の車がこのゲートを通過します。病院の受付の横にコインの自動販売機があって、見舞い客はこのコインを買い、駐車場を出るときにこのコインをスロットに投入すればゲートが開き、車が通過するとゲートが閉じるからくりになっています（実際にはもっと複雑なのでしょうが）。ゲートには2つの状態（state：ステート）、

・Open：開いている
・Close：閉じている

があり、ゲートに対して発生する事象（event：イベント）は、

・Coin：コインが投入される
・Pass：車が通過する

があります。ゲートの動きはこの状態と事象の組み合わせで表現できます。ある状態（s）において事象（e）が発生したとき、状態と事象の組み合わせに対応した動作（action）を行なって新たな状態に遷移します（図1）。

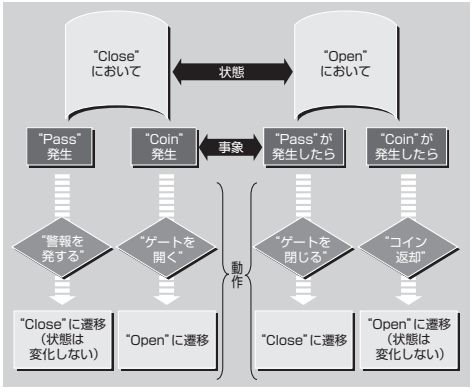

図1：状態と事象と動作の関係

このように、有限の状態数を持っていて、それぞれの状態で発生する事象（event）に応じた動作（action）と新たな状態（state）への遷移を行なうからくりを“Finite StateMachine（有限状態機械）”といいます。ゲートの有限状態機械に「gate_model」と名前を付け、今流行り（？）のXMLで書き表わしたのがリスト1（gate.xml）です。十数行のテキストとなりました。

さて、前回の「ModelとViewを分離する」では、上記の状態遷移表に基づいて動作するアプリケーションのModel部を書きました。少しばかり手を加えたコードをリスト2に示します。コードにして80行程度です。

リスト1：XML化したgate_model

```
<?xml version="1.0" encoding="shift_jis" ?>

<fsm namespace="gate_model">
 <state name=" close ">
  <event name=" coin "
       transition=" open ">ゲートを開く</event>
  <event name=" pass "
       transition=" close ">警報を発する</event>
 </state>
 <state name=" open ">
  <event name=" coin "
       transition=" open ">コインを返却</event>
  <event name=" pass "
       transition=" close ">ゲートを閉じる</event>
 </state>
</fsm>
```

リスト2：Model部に手を加えてみた

```
namespace gate_model {
 public delegate void Action();
 public class Behavior {
  public Behavior(Action action, State transition) {
   action_ = action;
   transition_ = transition;
  }
  public Behavior(State transition)
   : this(new Action(doNothing), transition) {}
  public State transition {
   get { return transition_; }
  }
  public Action action {
   get { return action_; }
  }
  public void setAction(Action action) {
   action_ = (action != null) ? action : new Action(doNothing);
  }
  private Action action_;
  private State transition_;
  private static void doNothing() {}
 }

 public class State {
  public Behavior coin;
  public Behavior pass;
  public State() {
  }
  public State(string name) {
   setName(name);
  }
  public static State execute(Behavior behavior) {
   if ( behavior != null ) {
    behavior.action();
    return behavior.transition;
   }
   return null;
  }
  public string name {
   get { return name_; }
  }
  public void setName(string name) {
   name_ = name;
  }
  private string name_;
 }

 public class FiniteStateMachine {
  public void coin() { setState(State.execute(state_.coin)); }
  public void pass() { setState(State.execute(state_.pass)); }
  public State state {
   get { return state_; }
  }
  public void setState(State state) {
   state_ = state;
  }
  private State state_;
 }

 public class StateTransitionTable {
  public State close = new State("close");
  public State open = new State("open");
  public StateTransitionTable() {
   close.coin = new Behavior(open);
   close.pass = new Behavior(close);
   open.coin = new Behavior(open);
   open.pass = new Behavior(close);
  }
 }
}
```