http://www.cetus.com.ua/research/article/article01.html

А.Г. © Реализация модели переходов состояний

А.Г. © Реалізація моделі переходів станів

Одним из видов моделей (программных систем) являются математические модели, а математическое моделирование системы рассматривается как высшая форма исследования, так как позволяет в наиболее общем виде в простой символьной форме определять отношения между элементами системы.

Л. Берталанфи сказал "... объект, в частности система, может быть охарактеризован только через свои связи в широком смысле слова – через взаимодействие составляющих элементов" [1]. Несмотря на это, уровень освещения проблем математического моделирования программных систем в литературе по вопросам проектирования программного обеспечения (в том числе, информационных и информационно-управляющих систем) не отвечает важности этого вопроса. Например, в [2-4] рассматриваются методы проектирования программных систем, но аспекты, связанные с математическим моделированием не освещаются. Проблемой обычно является не отсутствие математических моделей программных систем, а тот факт, что создатели программных систем в большинстве случаев прямо заимствуют описанные в этих работах методологии проектирования. Поэтому использование математических моделей при разработке программного обеспечения значительно чаще бывает связано с предметной областью проектируемой системы, чем с программным кодом.

С целью иллюстрации того, как математическое моделирование может влиять на прозрачность программного кода, а, следовательно, и на надежность и производительность системы, рассмотрим следующий пример.

Представим программную систему, которая должна определенным образом и с учетом текущего состояния реагировать на события, которые подаются на ее вход.

Традиционный подход к реализации программного обеспечения в этом случае – для каждого из событий установить собственный обработчик. Однако это приводит к большим трудностям понимания программы по ее тексту.

Если поведение системы задается графом переходов (рис. 1), то ее можно реализовать с помощью конструкции `'switch'` или `'if – else'`. При этом внесение изменений приводит к изменению программы в целом.

В то же время функциональная зависимость между событиями и состояниями может быть представлена в программном коде в виде некоторой стандартной структуры. Для иллюстрации этого ниже приведен текст программы на языке *Python*. При этом в качестве модели системы можно рассматривать массив `'stat',` состоящий из строк, число которых равно числу переходов на графе. Каждая строка включает: "текущее состояние", событие", "триггер блокирования перехода", "функцию, которая обеспечивает переход" и "новое состояние".

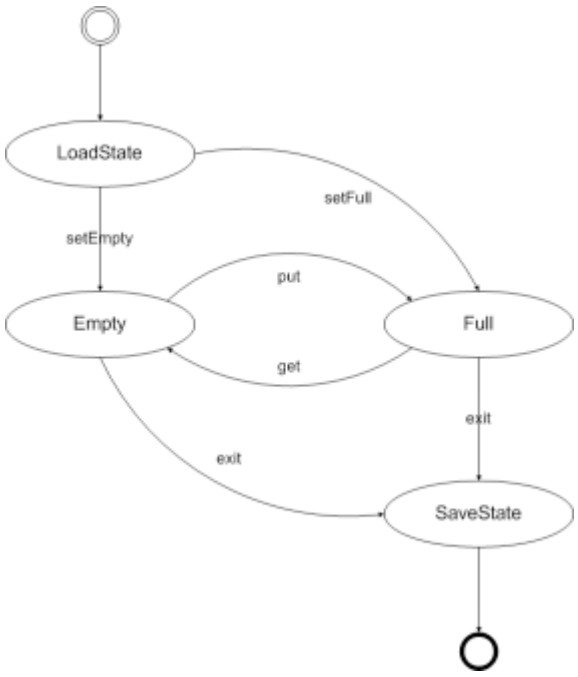

Рис. 1

Класс 'Automat', который включает всего десять строк программного кода, обеспечивает интерпретацию этой модели в соответствии с текущими событиями. Для упрощения модели приняты одинаковые названия для событий, которые подаются на вход системы (поле 2 массива `'stat'`), и функций, которые обеспечивают переход (поле 4 массива `'stat'`).

```
import sys

class Automat:
    def __init__(self,status):
        self.status = status
        self.currentStatus = status[ 0][ 0]
    def Event(self,eventTrigger):
        for i in self.status:
            if(self.currentStatus == i[ 0]  and eventTrigger == i[ 1]  and i[ 2] ):
                i[ 3] ()
                self.currentStatus = i[ 4]
                break

def SetFull():
    print "Set Full"
def SetEmpty():
    print "Set Empty"
def Put():
    print "Put"
def Get():
    print "Get"
def Exit():
    print "Exit"
    sys.exit()

stat = (
    [ "LoadState","SetFull",1,SetFull,"Full"] ,
    [ "LoadState","SetEmpty",1,SetEmpty,"Empty"] ,
    [ "Full","Get",1,Get,"Empty"] ,
    [ "Empty","Put",1,Put,"Full"] ,
    [ "Full","Exit",1,Exit,"SaveState"] ,
    [ "Empty","Exit",1,Exit,"SaveState"] ,
)
```

```
automat = Automat(stat)

automat.Event("SetFull")
automat.Event("Get")
automat.Event("Exit")
```

Использование подобного подхода при моделировании и реализации системы позволяет простыми средствами вносить дальнейшие изменения в математическую модель системы, оставляя без изменений средства его интерпретации.

Литература

1. *Берталанфи Л.* История и статус общей теории систем // Системные исследования. Ежегодник. М., 1973, с. 20-36
2. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы. М.: Символ Плюс, 1999. – 304 с.
3.Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. М.: БИНОМ», СПб.: «Невский Диалект», 1999. – 560 с.
4.Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК, 2000. – 432с.