УДК 629.45.015:004

А. Н. ПШИНЬКО, С. В. МЯМЛИН (ДИИТ)

ПРОГРАММА ВЫЧИСЛЕНИЙ ДИНАМИЧЕСКИХ ПОКАЗАТЕЛЕЙ ПАССАЖИРСКОГО ВАГОНА

Наведено опис алгоритму складання програми розрахунку динамічних показників пасажирського вагона з використанням об'єктно-орієнтованого моделювання, яке застосовувалося для виконання теоретичних досліджень пасажирських вагонів з різними характеристиками.

Приводится описание алгоритма составления программы вычислений динамических показателей пассажирского вагона с применением объектно-ориентированного моделирования, которое использовалось для выполнения теоретических исследований пассажирских вагонов с различными характеристиками.

In article the description of algorithm of drawing up of the program of calculations of dynamic indexes of the carriage with use of object-oriented simulation which was used for performance of theoretical researches of carriages with various characteristics is resulted.

Известно, что при теоретических исследованиях динамики рельсовых экипажей рассматриваются не реальные детали и узлы, а идеализированные объекты, которые отражают в той или иной степени основные свойства действительных конструкций. Поскольку деталей и узлов в экипажах много и они взаимосвязаны друг с другом, то в результате имеем достаточно сложную динамическую систему, движение которой описывается системой дифференциальных уравнений высокого порядка. Решение таких сложных систем дифференциальных уравнений – задача непростая, поэтому первые исследования динамики экипажей сводились к изучению колебаний по упрощенным расчетным схемам в виде подпружиненной массы. Еще в конце XIX – начале XX века выдающимися учеными А. Н. Крыловым, С. П. Тимошенко, Н. Е. Жуковским, а впоследствии В. А. Лазаряном и другими в такой постановке были решены задачи о вертикальных колебаниях вагона на рессорах различных типов, движущегося по изолированной неровности пути; о соударении двух вагонов; о продольном ударе колеса о рельс; об определении частот собственных колебаний и области возможных резонансных явлений. Затем колебания экипажей рассматривались по более сложным расчетным схемам, исследовались раздельно колебания в вертикальной и горизонтальной поперечной плоскостях симметрии. С развитием вычислительной техники расчетные схемы усложнялись. Учитывались нелинейности в связях между деталями вагонов, изучались совместные вертикальные, горизонтальные поперечные и горизонтальные продольные колебания, уточнялись и усложнялись модели возмущений, действующих на экипаж со стороны пути. Рядом ученых разработаны математические модели и программы вычислений для исследования пространственных колебаний рельсовых экипажей.

Создание этих программ по времени совпало с возникновением программирования как средства решения задач прикладных наук. В то время существовал и поддерживался средствами алгоритмических языков стиль процедурного программирования. Господствующей была следующая парадигма: определите, какие процедуры вам нужны, используйте лучшие алгоритмы, которые только сможете найти [1].

В языках программирования эта парадигма поддерживалась функциями, обеспечивающими передачу параметров и возврат результата. Основные проблемы такого программирования состояли в том, чтобы определить, каким способом передавать параметры, как различать виды параметров и функций. Для расширения возможностей функций использовались процедуры, подпрограммы и макроопределения. Функции предназначались для того, чтобы упростить понимание и упорядочить процесс реализации алгоритма решения поставленной задачи. Первыми из процедурных языков были Фортран, Алгол. Позже появились Паскаль и С. Эти языки программирования поддерживали тот же процедурный стиль программирования.

Решение любой задачи средствами программирования является не чем иным, как правильным с точки зрения используемого алгоритма управлением данными. Поэтому с течением времени в программировании больше внимания стали уделять не разработкам функций, а организации данных. Это было первым шагом к ограничению доступа к данным, что привело,

с одной стороны, к увеличению размеров программ, а с другой – к значительному снижению количества ошибок при разработке и как следствие к сокращению времени разработки программ. Набор взаимосвязанных процедур и данных, которыми они оперируют, назывался модулем, а этот стиль программирования – соответственно модульным программированием. С изменением стиля изменилась и парадигма программирования: определите, какие модули вам нужны, расчлените программу так, чтобы данные были скрыты в модулях.

Программные модули, разработанные с использованием такого подхода, обладают весьма важным преимуществом: ввиду своей универсальности они предполагают повторное использование разработчиками программ без изменения кода. В этом стиле разработано большинство библиотек прикладных программ, реализующих математические процедуры поиска решений систем уравнений, численного интегрирования и многие другие. Поскольку данные – это только один из элементов, «спрятанных» от пользователя, то со временем идея ограничения доступа к ним обобщается до концепции «скрытия информации», т. е. имена переменных, констант, функций и типы данных также локализуются в пределах модуля. Первым языком, поддерживающим парадигму модульного программирования, был Модула-2. Разработанный на базе языка С, новый язык программирования С++ хотя и не был специально ориентирован на поддержку данного стиля программирования, однако реализованная в рамках этого языка концепция класса поддерживает модульность.

Стиль модульного программирования в конечном итоге приводит к идее сосредоточения контроля над всеми данными одного типа в одном управляющем модуле. Такой подход к управлению данными дал возможность определять новые, существенно отличающиеся от встроенных в язык программирования, имеющие свои внутренние компоненты типы данных. В связи с этим каждый модуль управления данными должен обеспечить: во-первых, отдельный механизм для создания переменных нового типа, во-вторых, наличие процедур, реализующих алгоритм «поведения» переменных данного типа, и в-третьих, механизм доступа к компонентам этих переменных. Такой подход к разработке программ получил название «абстракция данных». С изменением стиля программирования меняется и его парадигма: определите, какие типы данных вам понадобятся, заготовьте полный набор операций для каждого типа.

Новые разработанные в программе типы данных часто называют пользовательскими типами. Такие языки программирования, как Ада и С++ обеспечивают реализацию данного стиля. В языке программирования С++ пользовательские типы данных называются классами. В рамках класса могут быть определены сколь угодно сложные, имеющие множество компонент данные. Однако при данном подходе не существует иного способа адаптировать тип к повторному употреблению в другом контексте, кроме как заново описать его. Например, класс геометрических фигур может быть описан следующим образом (этот пример часто приводится в литературе, например в работах [1; 2], как наиболее наглядный):

```
  // окружность, треугольник, квадрат
enum Kind { circle,  triangle,   square };
class Shape
{
  int CenterX, CenterY; // координаты центра фигуры
  int Color;          // цвет фигуры
  Kind k;             // тип конкретной фигуры
  // другие параметры
public:
  void Move( int X, int Y ); // переместить фигуру
  void Draw();               // нарисовать фигуру
  void Rotate( int Angle);   // повернуть фигуру
  // другие функции
}.
```

В данном случае поле типа фигуры (переменная k) необходимо для того, чтобы функции Draw() и Rotate() «знали», с какой разновидностью фигуры они имеют дело, т. к. очевидно, что процедура отрисовки или поворота окружности совсем не то же, что аналогичные процедуры для квадрата или треугольника. Получается, что подобные функции должны «знать» обо всех типах фигур, которые только существуют. Следовательно, если ввести в систему новую фигуру, то необходимо переписать часть функций. Поскольку включение новых фигур подразумевает воздействие на код каждой существенной операции над фигурами, это требует большого умения и будет служить потенциальным источником ошибок в коде, относящемуся к ранее созданным фигурам.

В данном случае проблема заключается в том, что отсутствует различие между общими свойствами фигур, такими как цвет и положение их центра, и свойствами, присущими конкретной фигуре (для окружности это радиус, для квадрата – длина стороны, для треугольника – длины трех сторон). Языки программирования, снабженные конструкциями, в которых явно выражено и может быть использовано это

различие, поддерживают парадигму (ООП): определите, какие классы вам нужны, заготовьте полный набор операций для каждого класса, выразите общие свойства явным образом, используя наследование.

Термин «наследование» позволяет решить проблему, о которой шла речь выше. Продолжая пример о геометрических фигурах, опишем класс, определяющий общие свойства фигур [1]:

```
class Shape
{
  int CenterX, CenterY;
  int Color;
  // другие параметры
public:
  void Move( int X, int Y );
  virtual void Draw();
  virtual void Rotate( int Angle);
  // другие функции
}.
```

Функции, у которых известен интерфейс вызова, но реализация не может быть определена в общем случае, а только для конкретных фигур, называются виртуальными. Этот термин означает, что функция может быть переопределена в производном классе. Чтобы определить конкретную фигуру, нужно указать, что она обладает общими свойствами фигур, и дополнить описание ее собственными свойствами:

```
class Circle : public Shape
{
  int Radius;
public:
  void Draw();
  void Rotate( int Angle);
}.
```

Данная запись означает, что кроме свойств, присущих всем фигурам (положение центра и цвет), окружность обладает собственным свойством (радиус). Класс Circle называется производным классом от класса Shape.

Применение объектно-ориентированного программирования для решения разных задач требует нахождения общности между различными типами данных, что может оказаться непростой задачей, которую необходимо тщательно проработать на этапе проектирования системы. Здесь работы необходимо осуществлять по двум направлениям: во-первых, разрабатывать классы как блоки, из которых могут составляться другие типы данных, и во-вторых, поиск вести в классах тех свойств, которые могут быть переданы базовым классам. В целом процесс проектирования программного обеспечения с использованием принципов ООП можно разбить на следующие этапы [1]:

1. Поиск классов и их основных взаимных связей. Выделение основных свойств и отнесение их к базовым классам.
2. Уточнение классов путем задания множества операций над ними.
3. Определение взаимозависимости классов путем установления наследования свойств или подчинения.
4. Определение отношения внутри классов путем разделения функций на приватные, открытые и, при необходимости, защищенные.

Эти этапы определяют не окончательный алгоритм построения системы, а лишь основные шаги итеративного процесса построения системы. Обычно требуется повторить эти шаги несколько раз, прежде чем будет выработан проект, который в достаточной степени подходит для начальной реализации. Полное описание программы вычислений приведено в Свидетельстве о регистрации прав на произведение [3].

Таким образом, описан алгоритм составления программы вычислений динамических показателей пассажирского вагона с использованием объектно-ориентированного моделирования, которое использовалось для выполнения теоретических исследований пассажирских вагонов с различными характеристиками.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Мямлин С. В. Моделирование динамики рельсовых экипажей. – Д.: Новая идеология, 2002. – 240 с.
2. Страуструп Б. Язык программирования С++. – К.: Диасофт, 1993. – 553 с.
3. Свідоцтво про реєстрацію авторського права на твір № 7305. Комп’ютерна програма «Dynamics of Rail Vehicles» («DYNRAIL») / Мямлін С. В.; Зареєстр. 20.03.2003.

Поступила в редколлегию 19.02.2005.