АКТУАЛЬНЫЕ ШАБЛОНЫ ПРОГРАММИРОВАНИЯ ИМИТАЦИОННЫХ МОДЕЛЕЙ СЛОЖНЫХ СИСТЕМ

В. Д. Левчук, П. Л. Чечет (Гомель)

Концепция объектно-ориентированного программирования в настоящий момент является базисом, на основе которого разрабатываются огромные программные комплексы. В силу глобального распространения высокопроизводительных персональных компьютеров создание таких комплексов потребовало массовой подготовки специалистов в области информационных технологий, прежде всего программистов. При этом многие приложения поставляются вместе со встроенным языком, позволяющим автоматизировать взаимосвязанные этапы работы в приложении, расширяя тем самым его возможности. Причем для программистов предпочтительнее, когда в качестве встроенного языка используется не уникальный, а известный язык программирования. Именно такой подход был применен при разработке программно-технологического инструментария MICIC4 [1], предназначенного для имитационного моделирования дискретных систем, где язык моделирования реализован как библиотека к широко распространенному объектно-ориентированному языку программирования C++.

Независимо от используемого при реализации проекта на имитационное моделирование инструментария эффективность работы программиста является низкой, если он полагается исключительно на собственный опыт объектно-ориентированного программирования в другой проблемной области. Систематизация знаний по преобразованию формальной модели в программу имитационной модели и постановке имитационных экспериментов связана с применением шаблонов объектно-ориентированного проектирования [2]. Каждый шаблон проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого кода.

Разработка имитационных моделей в MICIC4 является объектно-ориентированным проектированием. Общая структура модели с использованием MICIC4 предопределена инструментарием и представляет собой специфический шаблон проектирования. Дополнительно нами выделены известные шаблоны объектно-ориентированного проектирования [2] для скорейшего вовлечения программистов в процесс имитационного моделирования. Рассмотрим на двух примерах объектов моделирования набор активно применяемых шаблонов.

Шаблоны программирования в имитационной модели технологических процессов производства

Функционирование имитационной модели технологических процессов производства подробно описано в [1]. Взаимодействие между всеми компонентами имитационной модели технологических процессов производства через перемещение транзактов всех видов представлено на рис. 1. Сплошными стрелками на рис. 1 показаны возможные перемещения транзакта «Заявка». Пунктирными стрелками показано движение информационного транзакта, двойными стрелками – движение транзакта набора и возврата ресурсов. Штрих-пунктирные стрелки, идущие из узла обслуживания к ресурсу и обратно показывают передачу управляющих сигналов информационным и ресурсным транзактами друг другу в процессе захвата и освобождения ресурсов.

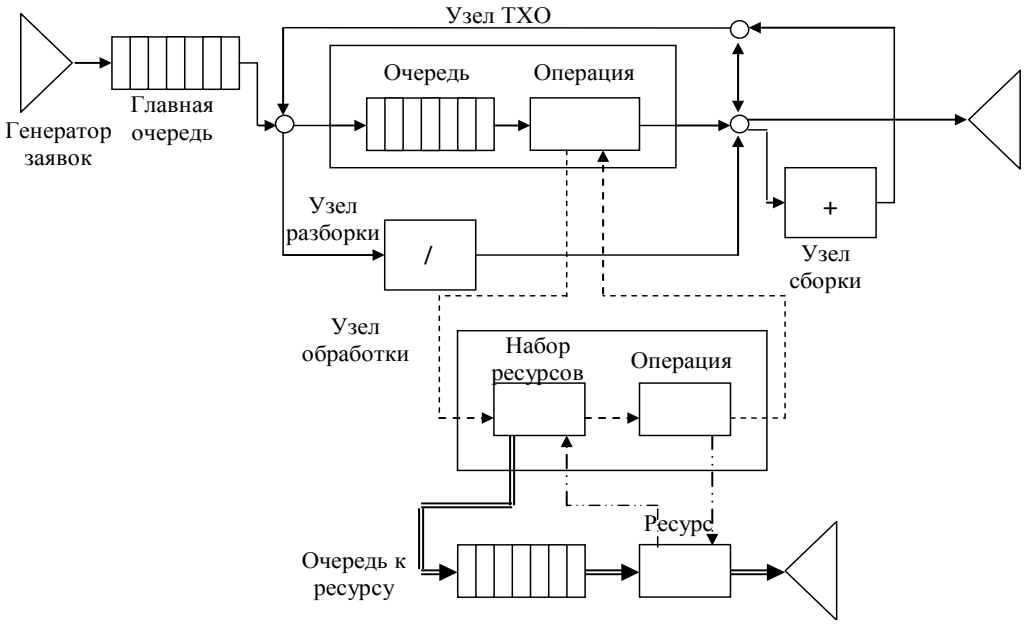

Рис. 1. Движение транзактов в модели

Для управления технологическими процессами в программу имитационной модели введен дополнительный класс для управления точками сборки и разборки и для хранения информации о взаимосвязях транзактов, обрабатывающихся в связанных технологических процессах. В программе имитационной модели этот класс описан следующим образом:

```
class DA{//класс для управления точками сборки-разборки
public:
 void add(char* name,int f,int s);
 int toAssembly(int tnsNum);
 void cameIn(int tnsNum);
 void clearTSS(void);
 static DA* instance();
private:
 static DA* daSelf;
 DA(int maxLength);
 ~DA();
 TSStorage* storage;
 int pos;
};
```

При разработке класса DA используется шаблон Singleton (одиночка). В классе DA есть статический метод instance( ), который приводит к созданию объекта только при первом вызове, при последующих вызовах этот метод возвращает указатель на ранее созданный объект. Использование шаблона Singleton обеспечивает гарантированное создание единственного экземпляра класса DA в тот момент, когда в программе модели в этом возникнет необходимость. Метод add( ) на основе шаблона Factory (фабрика) используется для добавления информации о произошедшей операции разборки. Метод toAssembly( ) позволяет определить дальнейшее перемещение родительского транзакта после сборки. Метод cameIn( ) реализует алгоритм ожидания и сборки частей, представленных двумя транзактами типа «Заявка». Метод clearTSS( ) позволяет удалить всю информацию о произошедших операциях разборки, он используется для очистки в начале каждой реплики моделирования.

Имитационная модель технологических процессов производства позволяет исследовать различные стратегии использования ресурсов. Как видно из рис. 1, запросы к ресурсам отправляются из устройства «Набор ресурсов», являющегося составной ча-

стью устройства «Узел обработки». В программе имитационной модели класс набора ресурсов представлен абстрактным классом GetUO и реализован с использованием шаблона Strategy (стратегия). Абстрактный класс наследован от класса «Устройство» и содержит два виртуальных абстрактных метода для реализации алгоритмов стратегий набора ресурсов. В программе имитационной модели этот класс описан следующим образом:

```
class GetUO : public Device{
  public:
   GetUO(int dOwner):
 Device(dOwner,GET_RES,MAXINT,fifo,clGetResStart,clGetResEnd){};
   virtual void checkRes(void)=0;
   virtual void resCaptured(tnum tns)=0;
};
```

В конкретных подклассах-наследниках GetUO нужно определить эти две функции, реализовав алгоритм набора ресурсов согласно выбранной стратегии. Метод checkRes( ) вызывается при запросе на выделение ресурсов или по окончании выполнения операции в узле обработки после освобождения занятых ресурсов. Перед вызовом этого метода при наборе ресурсов информационный транзакт задерживается на устройстве «Набор ресурсов». Алгоритм, реализованный в этом методе, может выбрать из списка задержанных транзактов любой транзакт согласно требуемой стратегии набора ресурсов, например самый ранний, с максимальным значением приоритета или с минимальным объемом запрашиваемых ресурсов. Метод resCaptured( ) вызывается при захвате ресурсным транзактом некоторого ресурса. При этом в метод передается номер этого ресурсного транзакта. Алгоритм, помещенный в этот метод, может анализировать состояние набора ресурсов и при наборе полного состава требуемых ресурсов освободить задержанный ранее информационный транзакт для начала моделирования процесса выполнения операции.

Использование шаблона Strategy (стратегия) выделяет алгоритмы, непосредственно реализующие стратегии набора ресурсов в отдельные классы. В имитационной модели по одному шаблону реализованы две различные стратегии набора ресурсов, названные «захватить всегда» и «захватить достаточное». По стратегии «захватить всегда» операция последовательно захватывает необходимые ей ресурсы, которые являются свободными. Выполнение операции начинается после захвата всех ресурсов. По стратегии «захватить достаточное» операция проверяет наличие свободных ресурсов, однако захват их выполняет только тогда, когда в системе будут в наличии все требуемые ресурсы. Очевидно, что при второй стратегии «захватить достаточное» за время от захвата первого ресурса до ожидания захвата последнего при стратегии «захватить всегда» может быть выполнена некоторая другая операция, которой требуется некоторая часть ресурсов, запрашиваемых операцией.

Шаблоны проектирования в имитационной модели городского пассажирского транспорта

Имитационная модель сети городского пассажирского транспорта предназначена для моделирования перевозки пассажиров транспортом, следующим по заданным графикам движения и маршрутам [3]. Возможные перемещения транзактов между устройствами этой имитационной модели показаны на рис. 2. Сплошными линиями показано возможное перемещение троллейбусов, пунктирными – пассажиров.

Как видно из рис. 2, маршрут движения троллейбусов в имитационной модели зависит от заданного маршрута, последовательности расположения на нем светофоров и остановок. Движение пассажиров, в свою очередь, определяется видом пассажира («рабочий», «служащий», «учащийся», «прочий») и его маршрутной картой. Некоторые

пассажиры после прибытия к нужной остановке сразу удаляются из модели, некоторые отправляются на остановку пересадки, некоторые – в место приложения труда. Возможность гибко задавать поведение отдельного пассажира позволяет с высоким уровнем детализации моделировать ситуацию в городской транспортной сети.

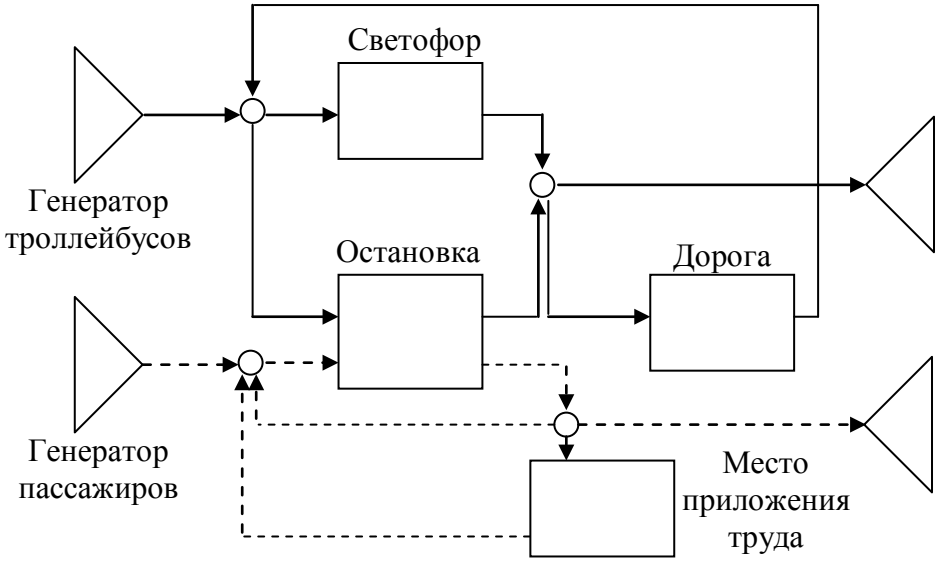

Рис. 2. Движение транзактов в модели

Отношения между классом транзактов «Пассажир» и маршрутной картой задаются шаблоном State (состояние). Класс пассажира Man агрегирует внутри себя прямую и возвратную маршрутные карты (рис. 3).

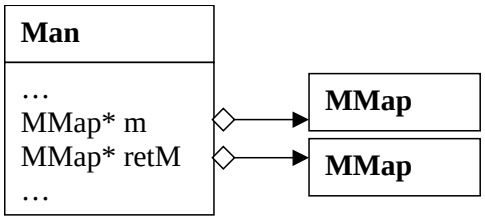

Рис. 3.Отношение между транзактом «Пассажир» и маршрутной картой

Алгоритмы имитации движения пассажира в своей работе определяются данными из прямой маршрутной карты. Когда начинается фаза движения пассажира обратно от места приложения труда к месту проживания, метод класса shiftMMap( ) заменяет класс прямой маршрутной карты на обратную. Состояние пассажира меняется, и начинается моделирование его возврата. Применение шаблона State (состояние) позволяет отказаться от использования большого количества условных операторов в алгоритмах имитации движения пассажира по транспортной сети, а также при необходимости в него легко добавить новые фазы движения пассажира без изменения алгоритмов имитации.

Для обеспечения гибкости в реализации множества маршрутных карт и остановок, структуры данных для хранения множества маршрутных карт и добавления новых маршрутных карт не описаны в классе остановок, а вынесены в отдельный класс MapSet.

В начале имитационного эксперимента в соответствующем методе класса эксперимента между парными классами остановок (с именем BusStop) и классами множества маршрутных карт устанавливается отношение осведомленности. На созданные в процессе загрузки данных классы множества маршрутных карт, каждый экземпляр класса

остановки получает ссылку для доступа к маршрутным картам пассажиров, генерируемых на этой остановке.

В тексте программы имитационной модели класс множества маршрутных карт описан следующим образом:

```
class MapSet{
 public:
  int num[4];//Кол-во МК по видам пассажиров (РСУП)
  MMap* maps[4][50][2];//Список маршрутных карт
  MapSet(void){num[0]=0;num[1]=0;num[2]=0;num[3]=0;};
  void add(char k, mMap* m,MMap* retM=NULL);
  void add(int k, mMap* m,MMap* retM=NULL);
};
```

Как видно из текста программы, в классе MapSet описаны два метода add( ) для добавления маршрутной карты, отличающиеся лишь типом первого параметра. Такая реализация позволяет при добавлении маршрутной карты не только указывать номер типа пассажира (от 0 до 3 для рабочего, служащего, учащегося и прочего), но и первую букву типа пассажира («Р», «С», «У» и «П»), что повышает удобство использования класса в программе модели. При необходимости развития множества маршрутных карт методы добавления можно объявить виртуальными, а вместо массива для хранения маршрутных использовать свойство, которое для доступа к данным также будет применять виртуальные функции. Это позволит строить свою иерархию классов от базового класса MapSet без изменения описания класса остановок. При этом между классами остановок и множествами маршрутных карт будет использоваться отношение осведомленности, задаваемое шаблоном Bridge (мост).

Выводы

Привлечение шаблонов проектирования позволяет эффективно реализовывать сложные имитационные модели как на универсальном языке объектно-ориентированного программирования, так и с помощью специализированного инструментария. Вместо интуитивного придумывания своей иерархии классов разработчик должен использовать соответствующие программные решения, которые апробированы ведущими специалистами в области объектно-ориентированного анализа. Рассмотренные выше примеры демонстрируют типовые возможности использования шаблонов объектно-ориентированного проектирования в конкретных имитационных моделях.

Литература

1. **Левчук В. Д, Максимей И. В**. Программно-технологические комплексы имитации сложных дискретных систем. Гомель: ГГУ им. Ф. Скорины, 2006. 263 с.
2. Приёмы объектно-ориентированного проектирования. Паттерны проектирования: Сер. Библиотека программиста/*Э. Гамма* [и др.]. СПб.: Питер, 2001. 368 с.
3. **Чечет П. Л.** Реализация имитационной модели сети городского пассажирского транспорта//Известия Гомельского государственного университета имени Ф. Скорины. 2006. № 4(37). С. 102–104.