Известия Гомельского государственного университета
имени Ф. Скорины, №6(45), 2007

УДК 681.3

Использование паттернов проектирования при разработке имитационных моделей

П. Л. Чечет, В. Д. Левчук

Введение

Проектирование объектно-ориентированных программ является сложным процессом [1]. Не существует универсальных алгоритмов по переводу реальной системы во множество объектов в программе. Разработчику необходимо подобрать подходящие объекты, отнести их к различным классам, соблюдая разумную степень детализации, определить интерфейсы классов и иерархию наследования и установить существенные отношения между классами. Дизайн проекта должен, с одной стороны, соответствовать решаемой задаче, а с другой - быть общим, чтобы удалось учесть все требования, которые могут возникнуть в будущем. Чаще всего в такой ситуации разработчики опираются на собственный опыт и интуицию.

Развитие теории объектно-ориентированного проектирования привело к попытке систематизации знаний в этой области и к появлению паттернов объектно-ориентированного проектирования. Каждый паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна.

Разработка имитационных моделей в системе моделирования MICIC 4 также является объектно-ориентированным проектированием [2]. Несмотря на то, что общая структура модели с использованием MICIC 4 предопределена системой моделирования, в ней также полезно использовать паттерны объектно-ориентированного проектирования.

К сожалению, только при разработке простых моделей разработчик может ограничиться только наследованием базовых классов системы моделирования MICIC 4. При моделировании сложных систем приходится заниматься проектированием дополнительных классов объектов, при котором использование паттернов объектно-ориентированного проектирования может оказаться очень полезным [3].

Паттерны проектирования в имитационной модели технологических процессов производства

Функционирование имитационной модели технологических процессов производства подробно описано в [4]. Взаимодействие между всеми компонентами имитационной модели технологических процессов производства с иерархической структурой через перемещение транзактов всех видов представлено на рисунке 1.

Сплошными стрелками на рисунке 1 показаны возможные перемещения транзакта «Транзакт-заявка». Пунктирными стрелками показано движение информационного транзакта. Двойными стрелками показано движение транзакта набора и возврата ресурсов. Штрихпунктирные 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.

Для управления иерархическими 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;
};
```

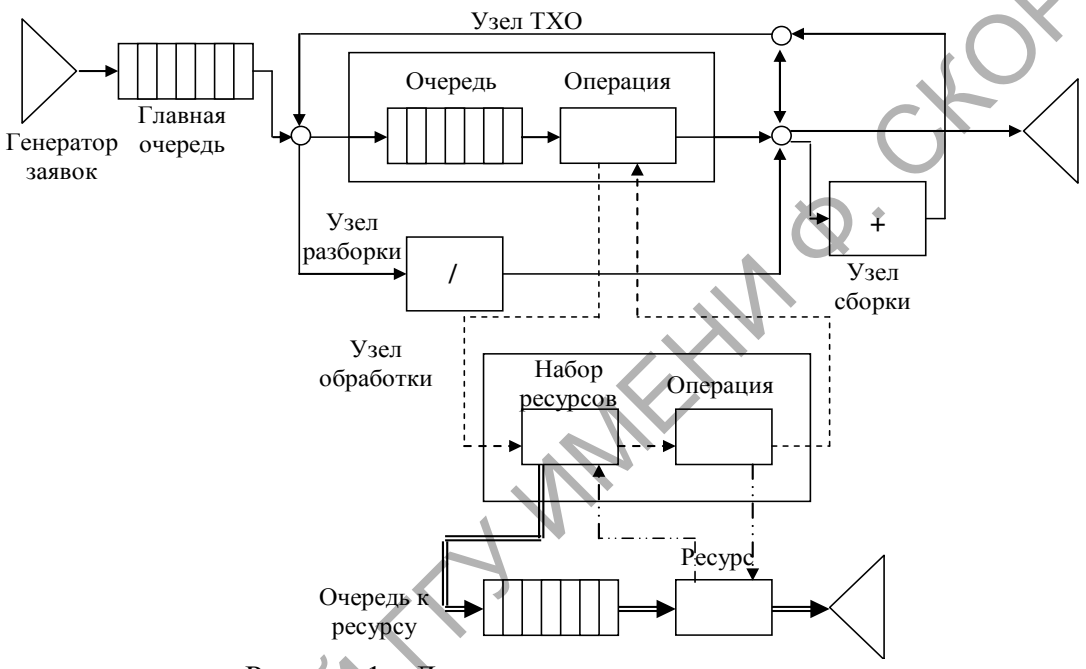

Рисунок 1 – Движение транзактов в модели

При разработке класса DA используется паттерн 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 паттерна Singleton обеспечивает гарантированное создание единственного экземпляра класса DA в тот момент, когда в программе модели в этом возникнет необходимость. Метод Add() 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 «Транзакт-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 с иерархической структурой позволяет исследовать различные стратегии использования ресурсов в технологическом процессе производства. Как видно из рисунка 1, запросы к ресурсам отправляются из устройства «Набор ресурсов», являющегося составной частью устройства «Узел обработки». В программе имитационной модели класс набора ресурсов представлен абстрактным классом GetUO. Класс набора ресурсов реализован с использованием паттерна 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 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, который захватил ресурс. 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 паттерна Strategy (стратегия) выделяет алгоритмы, непосредственно реализующие стратегии набора ресурсов в отдельные классы. В имитационной модели технологических процессов производства с иерархической структурой реализованы две различных 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 для выполнения необходимых 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 успеть выполниться некоторая другая операция, которой требуется некоторая часть ресурсов, запрашиваемых операцией.

Паттерны проектирования в имитационной модели городского пассажирского транспорта

Имитационная модель сети городского пассажирского транспорта предназначена для моделирования перевозки пассажиров городским пассажирским транспортом, следующим по заданным графикам движения и маршрутам [3]. Возможные перемещения транзактов между устройствами этой имитационной модели показаны на рисунке 2.

Сплошными линиями на рисунке 2 показано возможное перемещение транзактов троллейбусов, пунктирными – транзактов пассажиров. Как видно из рисунка 2, маршрут движения транзактов троллейбусов в имитационной модели зависит от заданного маршрута, последовательности расположения на нем светофоров и остановок. Движение транзактов пассажиров в свою очередь определяется видом пассажира («рабочий», «служащий», «учащийся», «прочий») и его маршрутной картой. Некоторые пассажиры после приезда к нужной остановке сразу удаляются из модели, некоторые оправляются 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.

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

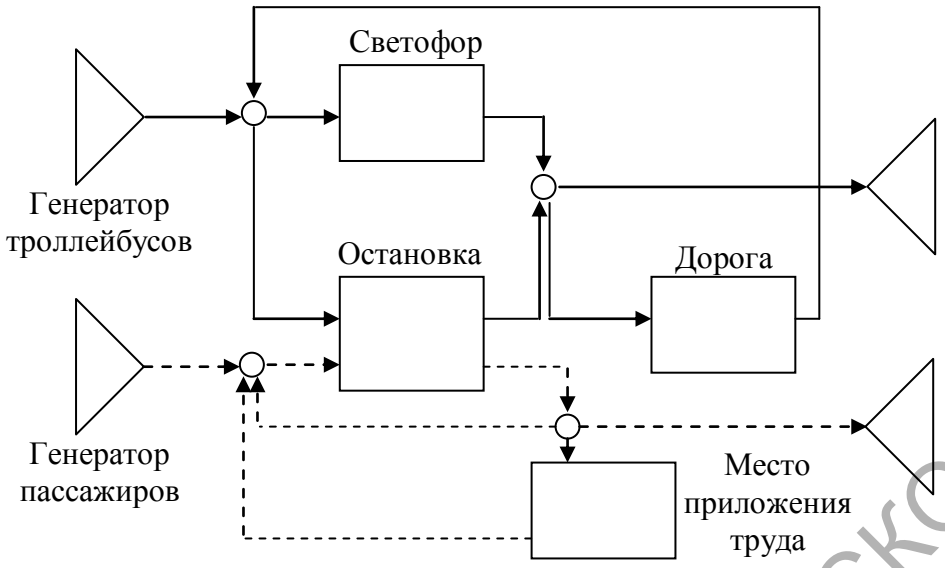

Рисунок 2 – Движение транзактов в модели

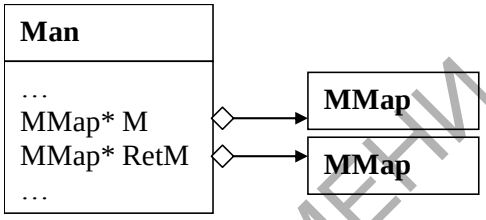

Рисунок 3– Отношение между транзактом пассажиром и маршрутной картой

Алгоритмы имитации движения пассажира в своей работе руководствуются данными из прямой маршрутной карты. Когда начинается фаза движения пассажира обратно от места приложения труда к месту проживания, метод класса ShiftMMap производит замену класса прямой маршрутной карты на обратную. Состояние пассажира меняется, и начинается моделирование его возврата. Использование паттерна State (состояние) позволяет отказаться от использования большого количества условных операторов в алгоритмах имитации движения пассажира по транспортной сети, а также при необходимости 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 системы моделирования MICIC 4 между парными классами остановок (с именем BusStop) и классами множества маршрутных карт устанавливается отношение осведомленности. На созданные в процессе загрузки данных классы множества маршрутный 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 использовать 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 паттерном Bridge (мост).

Заключение

Использование технологии паттернов объектно-ориентированного проектирования позволят быстро и качественно строить сложные имитационные модели в системе моделирования MICIC 4. Вместо интуитивного придумывания своей иерархии классов разработчик может использовать технологии разработки сложных моделей с использованием паттернов объектно-ориентированного проектирования. Так как MICIC 4 является открытой системой моделирования и представляет собой надстройку над языком C/C++, то не накладывается никаких дополнительных ограничений в использовании паттернов проектирования для разработки имитационных моделей сложных систем. Рассмотренные выше примеры демонстрируют возможности использования паттернов объектно-ориентированного проектирования в конкретных имитационных моделях, в которых использование интуитивного подхода, как правило, приводит к усложнению понимания, отладки и дальнейшего развития модели.

Abstract. The design patterns for simulation models are described in the paper. The examples of using the patterns are considered.

Литература

1. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования: серия «Библиотека программиста» / Э. Гамма [и др.]; пер. А. Слинкин – СПб: Питер, 2001. — 368 с.

2. Левчук, В. Д. Базовая схема формализации системы моделирования MICIC4 / В. Д. Левчук // Проблеми програмування, 2005. – № 1. – С. 85–96.

3. Чечет, П. Л. Реализация имитационной модели сети городского пассажирского транспорта / П. Л. Чечет // Известия Гомельского государственного университета имени Ф. Скорины, 2006. – № 4(37). – С. 102–104

4. Чечет, П. Л. Инструментальные средства реализации имитационных моделей технологических процессов производства с иерархической структурой / А. С. Помаз, Е. О. Попова, А. М. Потащенко, В. В. Старченко // Информационные системы и технологии. Материалы II Международной конференции. Минск, 8–10 ноября 2004 г. – Академия управления при Президенте Республики Беларусь, 2004. – С. 228–233.

Гомельский государственный
университет имени Ф. Скорины

Поступило 15.05.07