БИБЛИОТЕКА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СИСТЕМ С ДИСКРЕТНЫМИ СОБЫТИЯМИ C++SIM

Д. В. Щетинин (Тверь)

Введение

В своей деятельности разработчики имитационных моделей сталкиваются с проблемой представления сложных систем и процессов в как можно более простой (но достаточно адекватной) форме. Важную роль при этом играет выбор языка программирования. Современные средства и языки имитационного моделирования можно разделить на несколько категорий [1]. В их число входят языки имитационного моделирования, основанные на универсальных языках программирования.

Современные универсальные языки программирования являются мощным, гибким, удобным и эффективным средством решения широкого круга задач. Возможности универсального языка можно расширить с помощью библиотек, по сути, создавая на его базе специализированный язык программирования. На сегодняшний день существует достаточно много имитационных систем, построенных таким образом на базе какого-либо универсального языка программирования. Такой способ построения имитационных систем хорошо зарекомендовал себя при решении целого ряда задач, особенно при разработке сложных имитационных моделей и крупных программных комплексов на их основе. В данной статье рассматривается одна из реализаций данного подхода – библиотека C++SIM [2], [3].

Библиотека C++SIM позволяет использовать язык программирования C++ для разработки имитационных моделей систем с дискретными событиями в стиле Simula, который характеризуется как язык имитационного моделирования, ориентированный на процессы [4]. Язык программирования Simula хорошо известен среди разработчиков имитационных моделей. Используемый в нем подход многими специалистами [4], [5], оценивается как один из наиболее мощных, обладающих высокой степенью общности, и при этом весьма простой. Библиотека C++SIM сохранила эти качества, предоставив возможность использовать при создании имитационных моделей такой мощный инструмент разработки программного обеспечения как язык программирования C++.

Рассмотрение библиотеки C++SIM может быть полезно с различных позиций. С практической точки зрения, данная библиотека является хорошей основой для построения достаточно сложных, промышленных имитационных моделей и программных комплексов. Так, с ее использованием была разработана имитационная модель наземного движения воздушных судов по летному полю для аэропорта Шереметьево [6]. С другой стороны, на примере этой библиотеки можно продемонстрировать устройство и принципы работы современных языков имитационного моделирования. Наконец, библиотека C++SIM может послужить прототипом при разработке собственных систем имитационного моделирования, в том числе и на других языках программирования.

Устройство библиотеки C++SIM

В основе C++Sim (как и Simula) лежит понятие активного объекта – *процесса*. Процессы могут планировать свою деятельность, привязывая ее к модельному времени. Определение процесса библиотеки C++SIM задано в абстрактном классе ***Process*** (см. Листинг 1).

Квазипараллельное функционирование процессов обеспечивается с помощью механизма т.н. легковесных процессов (нитей – threads), соответственно класс ***Process*** является наследником абстрактного класса ***Thread***[1], в котором объявлены (но не реализованы) все операции, необходимые для квазипараллельного функционирования активных объектов. Таким образом, чтобы описать некоторый процесс разработчик должен создать класс-наследник ***Process***.

Правило действий процесса должно быть задано в методе ***Process::Body***. При активизации процесса управление передается этому методу, причем в ту точку, на которой была завершена предыдущая активная фаза.

В C++SIM поддерживаются те же методы управления процессами, которые использовались в Simula.

В библиотеке предусмотрено пять методов (***Activate*()***) для активизации *пассивных* процессов. Эти методы помещают процесс в очередь событий (управляющий список – см. ниже) на позицию, соответствующую времени активизации процесса. Время активации указывается либо явно (для ***Activate()*** это текущий момент времени, для ***ActivateDelay()*** – через указанный промежуток времени, для ***ActivateAt()*** – конкретный момент времени), либо относительно некоторого процесса (***ActivateBefore()***, ***ActivateAfter()***).

Еще пять аналогичных методов (***ReActivate*()***) предназначены для работы не только с пассивными, но и с активными и приостановленными (присутствующими в очереди событий) процессами. Если процесс активен, он будет приостановлен. Ссылка на него будет изъята из управляющего списка, если она там присутствует, а затем будет выполнен соответствующий метод ***Activate*()***.

Метод ***Cancel()*** переводит активный или запланированный процесс в пассивное состояние. ***Hold()*** позволяет приостановить на заданное время работу активного в настоящий момент процесса. ***Passivate()*** переводит процесс в пассивное состояние, но работает только для активного процесса (именно поэтому методы ***Passivate()*** и ***Hold()*** объявлены как защищенные).

Чтобы завершить работу процесса (исключить его как активный объект из дальнейшей работы системы), следует использовать метод ***terminate()***. В принципе, тот же результат должен получаться при завершении (возврате из) ***Body()***. Но из-за ограничений некоторых реализаций классов нитей, рекомендуется завершать процессы только с помощью метода ***terminate()***.

Для организации повторного прогона модели в рамках одного приложения предлагается следующий механизм. Для сброса модели в начальное состояние следует послать планировщику сообщение ***reset()***. В ответ на него планировщик удаляет все запланированные процессы из очереди событий и посылает сообщение ***reset()*** каждому процессу. Предполагается, что при получении этого сообщения объект должен выполнить специфичные для класса этого объекта действия, обеспечивающие повторную инициализацию процесса перед повторным выполнением модели. По умолчанию, этот метод не выполняет никаких действий (за исключением выдачи отладочных сообщений).

[1] Предполагается, что при разрботке приложения в проект подключается некоторый класс-наследник ***Thread***, полностью реализующий функциональность нитей для конкретной платформы (операционной системы). В библиотеку включено несколько таких реализаций. Этот механизм (хотя и не идеально реализованный в C++Sim) позволяет разработчику имитационной модели работать с абстракцией активного объекта, не обращая внимания на специфичные для конкретной платформы детали реализации.

```
class Process : public Thread
  {
  public:
    virtual ~Process();

    static double CurrentTime();  // C++SIM version
    double Time() const;          // SIMULA version

    static const Process * current();
    const Process* next_ev() const;

    Boolean passivated() const;
    Boolean terminated() const;
    Boolean idle() const;
    double evtime() const;


    void Activate();
    void ActivateBefore(Process &);
    void ActivateAfter(Process &);
    void ActivateAt(double atTime= CurrentTime(), Boolean prior = FALSE);
    void ActivateDelay(double delay= CurrentTime(), Boolean prior= FALSE);

    void ReActivate();
    void ReActivateBefore(Process &);
    void ReActivateAfter(Process &);
    void ReActivateAt(double atTime= CurrentTime(), Boolean prior= FALSE);
    void ReActivateDelay(double atTime = CurrentTime(), Boolean prior = FALSE);

    void Cancel();

    virtual void terminate();

    virtual void Body() = 0;

    virtual void reset();

   protected:
     Process();
     Process(unsigned long stackSize);

     void Hold(double t);
     void Passivate();
  };
```

Листинг 1. Класс Process в библиотеке C++SIM

Класс ***Process*** содержит несколько методов для получения информации о внутреннем состоянии процесса. С помощью ***passivated()*** можно проверить, находится ли процесс в пассивном состоянии. Вызов ***terminated()*** показывает, является ли процесс завершенным. Метод ***idle()*** возвращает значение «ложь» (***FALSE***), если процесс активен в настоящий момент или запланирована его следующая активная фаза; в противном случае возвращается значение «истина» (***TRUE***). Функция ***evtime()*** возвращает время, на которое запланирована активизация процесса (если для процесса активная фаза не запланирована, будет возвращено значение, меньшее текущего времени).

В класс Process включено также несколько методов, возвращающих информацию о состоянии всей среды моделирования. К этой категории можно отнести методы ***current()***, возвращающий указатель на активный процесс (***NULL***, если очередь планировщика пуста), ***next_ev()***, возвращающий указатель на процесс, который запланирован для выполнения следующим, а также ***CurrentTime()*** и ***Time(),*** которые позволяют получить текущего значение модельного времени.

Планировщик

Относительно системного времени процессы в системе работают параллельно, но в каждый момент реального времени выполняется всего один процесс. Активизация некоторого процесса обычно сопряжено с изменением в системе и называется *событием*. При этом информация о запланированных, но еще не исполненных событиях помещается в *управляющий список*. Записи в управляющем списке содержат ссылку на процесс и время, когда данный процесс должен быть активизирован.

Работа процессов контролируется *планировщиком* (scheduler). По сути, он занимается обслуживанием управляющего списка и отвечает за корректную обработку помещенных в него уведомлений о событиях. Планировщик, получив управление после того, как текущий процесс завершил свою активную фазу, должен выбрать запись с минимальным временем из управляющего списка и передать управление соответствующему процессу. Если уведомлений о событиях в списке нет, то моделирование считается завершенным.

В C++Sim планировщик реализован в классе ***Scheduler***. В модели должен присутствовать ровно один экземпляр этого класса, поэтому класс реализован с применением паттерна Singleton [7]: конструктор находится в закрытой части класса, а ссылку на единственный экземпляр можно получить с помощью метода ***Scheduler::scheduler()***.

В первых версиях библиотеки планировщик был реализован как отдельная нить, но в целях повышения производительности от этого подхода отказались. Поэтому класс ***Scheduler*** не наследуется от ***Thread***, а действия по активизации очередного процесса перенесены в ***Process::schedule()***. Планировщик же используется только для получения информации о том, запущена ли имитация в данный момент.

Заключение

На сегодняшний день C++ является одним из наиболее популярных объектно-ориентированных языков программирования. Но при этом он не лишен целого ряда недостатков. Среди них в первую очередь следует отметить весьма ощутимую «тяжеловесность» языка и существенные трудности, с которыми сталкиваются разработчики при сопровождении и развитии написанных на C++ программ. Одной из основных причин наличия этих и других недостатков данного языка является статическая типизация, лежащая в его основе. Такие же недостатки присущи практически всем языкам программирования, унаследованым от C++ (Java, C#) и построенным на тех же принципах (Object Pascal, Delphi).

Альтернативой данному направлению является Smalltalk, являющийся чистым объектно-ориентированным языком программирования с динамической типизацией. Этот язык сочетает в себе гибкость и мощь (пожалуй, даже превосходя в этом отношении C++) с предельной простотой лежащих в его основе идей и (как следствие) синтаксиса. Особенно ярко преимущества Smalltalk могут проявиться при его использовании в области имитационного моделирования. Описания имитационных моделей на Smalltalk приведены, например, в [8], [9]. Хотя данный язык в настоящее время нельзя назвать очень популярным (особенно в России), но он заслуживает пристального внимания со стороны разработчиков программного обеспечения, в том числе со стороны сообщества разработчиков имитационных моделей.

Литература

[1] **Stahl, I**. Simulation Prototyping. Proceedings of the 2002 Winter Simulation Conference, 572–579.
[2] **Little, M.C., McCue, D.L.** Construction and Use of a Simulation Package in C++. Department of Computing Science, University of Newcastle upon Tyne.
[3] C++SIM User's Guide. Department of Computing Science, Computing Laboratory, The University, Newcastle upon Tyne.
[4] **Шеннон Р.** Имитационное моделирование систем – искусство и наука. – М.: Мир, 1978.
[5] **Киндлер Е.** Языки моделирования. – М.: Энергоатомиздат, 1985.
[6] **Исаев А.С., Местецкий Л.М., Федоров А.В., Щетинин Д.В.** Имитационное моделирование аэропорта – инструмент обоснования решений//Аэропорт Партнер, №5-6, 2002
[7] **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001.
[8] **Maxwell, D.T**. An Overview of The Joint Warfare System (JWARS). MITRE Corporation, 2002.
[9] **Bright K.P., Sherlock, R.A., Lile J., Wastney M.E.** Development and Γse of a whole farm model for dairying. In: Applied Complexity: From Neural Nets to Managed Landscapes. Halloy S and Williams T (Eds). NZ Institute for Crop and Food Research, Christchurch, NZ 2000, 382–389.