БИБЛИОТЕКА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ТЕЛЕКОММУНИКАЦИОННЫХ СЕТЕЙ TKSYM

А. В. Тимофеев, А. В. Сырцев, А. В. Колотаев (Санкт-Петербург)

В данном докладе представлена библиотека имитационного моделирования (ИМ) телекоммуникационных сетей (ТКС) tksym. Основное отличие этой библиотеки от подобного рода объектно-ориентированных библиотек (например, OMNeT++ [1]) заключается в том, что в качестве основного способа параметризации модулей библиотеки используются шаблоны языка C++. Цель, которая преследовалась при проектировании библиотеки, – предоставить ее пользователю набор компонент для моделирования различных алгоритмов маршрутизации в ТКС, которые было бы легко использовать, настраивать под свои нужды и не платить за это снижением быстродействия имитационной программы.

Представление имитационной модели в виде набора взаимодействующих компонент является общепринятой практикой и имеет большую историю (например, один из первых языков объектно-ориентированного программирования Simula-67 предоставлял средства для модульной декомпозиции моделей еще в конце 60-х годов прошлого века). Преимущества модульного подхода хорошо известны. Во-первых, это возможность отразить в коде имитационной программы структуру моделируемой системы, что, несомненно, способствует пониманию программы и облегчает таким образом ее сопровождение. Во-вторых, такой подход открывает возможности для использования одного модуля в качестве составного элемента для построения разных имитационных моделей, т.е. для повторного использования модулей; это снижает затраты на построение и отладку новых имитационных моделей. Чем сложнее конструируемая модель, тем ярче проявляют себя преимущества модульного подхода.

Разработчики библиотек помимо выполнения главной задачи – предоставления некоторой полезной функциональности – стремятся сделать компоненты библиотеки пригодными для использования при решении как можно большего спектра задач, расширяя таким образом область применимости библиотеки и, следовательно, ее полезность. С пригодностью компоненты к ее повторному использованию связана возможность настраивать пользователю ее поведение для своих нужд, а также способность быть использованой в различных окружениях. Основной прием достижения этих качеств – абстрагирование, которое в упрощенном виде можно описать как вынесение из компоненты всего, что не относится к ее сути, в виде параметров компоненты. Связывая с параметрами компоненты разные значения, можно менять ее поведение. Основной метод подобной параметризации компонент в ООП – использование механизма динамического полиморфизма, поддержка которого во многих ОО языках программирования реализована через механизм виртуальных функций.

К сожалению, подобную параметризацию средствами ООП можно проводить только до определенного предела, когда накладные расходы, связанные с использованием виртуальных функций станут неприемлемо большими. Для разработчика ОО библиотеки это означает необходимость постоянного поиска баланса между настраиваемостью компонент библиотеки и их эффективностью, при котором приходится находить компромисс для различных случаев использования библиотеки, что во многих случаях является крайне нетривиальной задачей.

Основной урон производительности в первую очередь обусловлен неспособностью компилятора проводить оптимизации кода[1] вокруг точки вызова виртуальной

[1] В первую очередь это относится к возможности встраивания тела функции в точку ее вызова.

функции, поскольку он не может определить, какая именно функция будет вызвана. Кроме этого, на некоторых процессорных архитектурах косвенный вызов, к которому сводится вызов виртуальной функции, сбрасывает содержимое конвейера команд, что также наносит удар по производительности.

Кроме урона быстродействию выполняемой программе использование динамического полиморфизма обладает рядом отрицательных эффектов, среди которых можно указать:

a. уменьшение возможностей для статического контроля типов, что отодвигает момент диагностики неправильного использования компоненты на более поздний период: с момента компиляции на момент выполнения программы;

b. необходимость наследования классов, которые предполагается использовать как параметры компоненты, от определенных базовых классов, что снижает адаптируемость компоненты.

Указанные проблемы успешно решаются средствами обобщенного программирования [2, 3, 4], которое поддержано в языке C++ мощным механизмом шаблонов. Библиотеки, выполненные в парадигме обобщенного программирования, сочетают широкие возможности настройки компонент под нужды пользователя, удобство их использования с эффективностью результирующего кода.

При использовании шаблонов языка C++ информация, которая помогает компилятору сгенерировать оптимизированный код, не теряется, и поэтому при использовании хороших оптимизирующих компиляторов получающийся код не уступает значительно по эффективности нешаблонному коду. Безопасность типов также сохраняется, поскольку реальный тип объектов известен в момент компиляции.

За последние 10 – 15 лет были созданы и применяются библиотеки обобщенного программирования на C++ для ряда предметных областей: STL (Standart Template Library) [5] – линейные последовательности (динамический массив, список, стек, очередь, куча, дека, сбалансированное дерево) и алгоритмы над ними; CGAL (Computational Geometry Algorithm Library)[6] – алгоритмы и структуры данных вычислительной геометрии; BGL (Boost Graph Library) [7] – графы и алгоритмы на графах; MTL (Matrix Template Library)[8], GMCL (Generative Matrix Computational Library) [9] – матрицы и матричные вычисления; blitz++[10] – многомерные массивы для численных методов; ITL++ (Iterative Template Library)[11] – итеративные численные методы.

Характерной особенностью дизайна библиотеки tksym является разделение модулей на простые и составные – идея, подчерпнутая из архитектуры системы ИМ OMNeT++ [1]. Простые модули содержат алгоритм поведения некоторого элемента модели. Составные модули агрегируют в себе другие модули (неважно, простые или составные), соединяют их друг с другом, а также связывают с их параметрами определенные значения. Существенным моментом здесь является то, что простые модули берут на себя всю поведенческую составляющую модели (т. е. все поведение модели есть композиция поведения простых модулей), а составные модули – всю организационную составляющую модели (т. е. именно они задают, как модули соединяются и какие значения будут присвоены им параметрам). Моделью является составной модуль верхнего уровня, т.е. такой, который не принадлежит никакому другому модулю.

Обобщенное программирование является подходящей технологией для реализации простых модулей, поскольку позволяет создание сильно параметризованных, обобщенных компонент, которые не уступают в эффективности непараметризованным модулям.

В настоящее время простые модули реализованы как классы, которые имеют один шаблонный параметр – тип наследника. Параметризация базового класса типом

наследника является довольно распространенным в мире C++ программирования приемом и известна под названием curiously recurring template pattern (CRTP) [12].

К сожалению, методов обобщенного программирования недостаточно для разработки всей библиотеки имитационных моделей – кроме высокопараметризованных простых модулей необходимо предоставить средства, помогающие пользователю отобразить замысел о конструируемой модели в набор классов и их параметров, который задуманную модель реализует. В OMNeT++ для описания составных модулей (и интерфейса простых модулей для того, чтобы они могли быть частью составного модуля) используется специальных язык NED (**Ne**twork **D**escription Language). Компилятор с языка NED преобразует описания модулей в исходный код на C++, который после обработки компилятором C++ компонуется вместе с объектными файлами, содержащими алгоритмы модели, формируя таким образом имитационную программу. Именно здесь становится важной разница между простыми и составными модулями: простые модули лучше всего записывать на алгоритмическом языке программирования, а их конфигурирование лучше поручить специальному генератору. В настоящее время при использовании tksym для составления из простых модулей имитационной модели программист должен написать код, явно их конфигурирующий, что весьма неудобно; поэтому очень актуальной является задача автоматической генерации модели по высокоуровневой спецификации. Шаблонное метапрограммирование [13] представляется подходящей технологией для решения этой задачи; перспективность этого подхода исследуется.

На данный момент библиотека моделей содержит модели узлов ТКС, работающих под управлением алгоритмов статической маршрутизации (основанной на построении матрицы кратчайших путей), дистанционно-векторной маршрутизации с несколькими вариантами стратегии обновления таблиц маршрутизации, маршрутизации с учетом состояния каналов, Q-маршрутизации, многоадресных алгоритмов маршрутизации: reverse path multicasting, reverse path broadcasting.

В библиотеке tksym не реализованы многие типичные для систем ИМ компоненты: генераторы случайных чисел с заданными законами распределения, контейнеры, средства статистического анализа и пр. Это решение продиктовано тем, что существуют свободно распространяемые C++ библиотеки высокого качества, которые можно легко использовать для написания имитационных программ. Например, Boost.Random предоставляет генераторы случайных чисел, Boost Graph Library – графы и алгоритмы для работы с ними.

Для удобства пользователя реализован графический интерфейс к библиотеке на языке C#. Среда ИМ позволяет пользователю редактировать графовую модель сети, задавать параметры эксперимента, проводить имитацию в автономном и интерактивном режиме. В интерактивном режиме на графовой модели сети отображается анимация передачи пакетов по сети, а также выводится график загруженности выходных буферов узлов сети. Автономный режим служит для продолжительных по времени экспериментов – результаты эксперимента выдаются пользователю только после его окончания. Реализована возможность экспорта результатов эксперимента в таблицу MS Excel, а также экспорт модели в систему ИМ Arena [17].

Литература

1. OMNeT++ Home page. http://www.omnetpp.org/
2. http://www.cs.rpi.edu/~musser/gp/dagstuhl/gpdag_2.html
3. **Austern, M.** *Generic Programming and the STL: Using and Extending the C++ Standard Template Library.* Addison-Wesley Professional. – 1998. – 576 p.

4. **Vandervoorde D., Josuttis N.** *C++ Templates: The Complete Guide*, Addison-Wesley Professional. – 2002. – 552 p.
5. Standard Template Library Programmer's Guide http://www.sgi.com/tech/stl/
6. Computational Geometry Algorithm Library Home Page. http://www.cgal.org
7. Boost Graph Library Home Page http://www.boost.org/libs/graph/doc/table_of_contents.html
8. Matrix Template Library Home Page http://www.osl.iu.edu/research/mtl/
9. Generative Matrix Computational Library Home Page http://www-ia.tu-ilmenau.de/~czarn/gmcl/
10. Blitz++ Home Page http://www.oonumerics.org/blitz/
11. Iterative Template Library Home Page http://www.osl.iu.edu/research/itl/
12. **D. Abrahams, A. Gurtovoy.** *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond* – Addison Wesley Professional, 2004.
13. Boost Metaprogramming Library Home Page. http://www.boost.org/libs/mpl/doc/index.html
14. **Gamma E., Helm R., Johnson R., Vlissides J.** *Design Pattern*. Addison-Wesley Professional. – 1995. – 416 p.
15. http://www.arenasimulation.com/