ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ЖУРНАЛ
2005 г. ЯНВАРЬ—ФЕВРАЛЬ ТОМ 78, № 1

УДК 53.072.001.57

А. Н. Мигун, Е. А. Матвейчик, А. П. Чернухо, С. А. Жданок

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К МОДЕЛИРОВАНИЮ ХИМИЧЕСКОЙ КИНЕТИКИ

Рассматриваются структура и основные элементы реализации разработанной библиотеки классов для моделирования химической кинетики, современный подход к проектированию которой позволяет достичь высокой универсальности, гибкости и скорости вычислений.

Введение. В последние десятилетия в связи с бурным развитием электронных вычислительных машин (ЭВМ) численное моделирование стало мощным исследовательским инструментом, логично дополняющим экспериментальные исследования. Более того, современные экспериментальные исследования не возможны без поддержки комплексным вычислительным аппаратом, что также обусловило развитие соответствующего программного обеспечения (ПО). В некоторых случаях моделирование является единственно возможным способом вникнуть в детали какого-либо процесса. Все это привело к образованию отдельной отрасли, целью которой является разработка и внедрение на рынок программного обеспечения различного рода исследовательских программных продуктов. Образовавшийся рынок ПО можно условно разделить на две основные группы: прикладное ПО и специализированные библиотеки подпрограмм. Прикладное ПО представляет собой одну или несколько законченных математических моделей, имеющих пользовательский интерфейс, требующих для своей работы только ввода исходных данных и предназначенных для моделирования конкретных устройств или процессов. Примерами прикладного ПО являются FLUENT [1], StarCD [2], CHEMKIN [3] и другие. Специализированные библиотеки подпрограмм также являются логически законченными программными модулями, но в отличие от прикладного ПО предназначены для узкоспециализированных вычислений и имеют только программный интерфейс для вызова из программ пользователя. Примерами специализированных библиотек подпрограмм являются NAG [4], Numerical Recipes [5] и старые версии CHEMKIN.

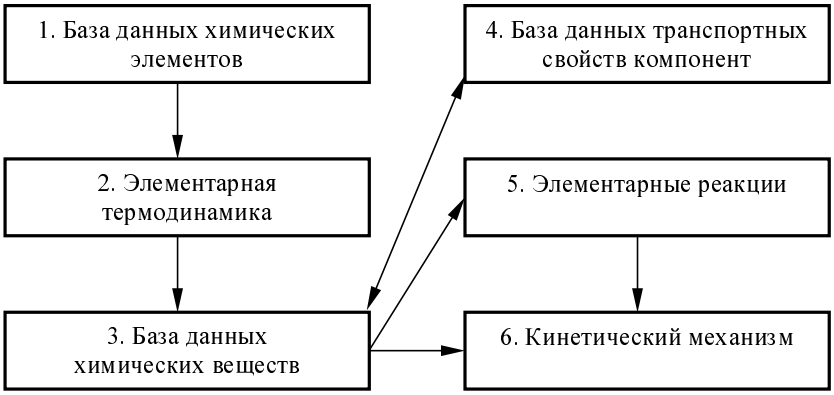
В большинстве своем научно-исследовательское ПО и библиотеки подпрограмм написаны на различных версиях языка программирования Fortran. Являясь линейным языком программирования, Fortran не удовлетворяет современным требованиям и не позволяет применить новые подходы к проектированию комплексного программного обеспечения. Кроме того, распространяемое на коммерческой основе ПО обычно является "черным ящиком" и не позволяет пользователю вносить какие-либо дополнения и усовершенствования.

Как правило, основные усилия программистов были направлены на увеличение скорости расчетов. Однако чрезвычайно быстрый рост производительности ЭВМ за последнее десятилетие несколько понизил приоритет скорости и сместил его в сторону универсальности, позволив уделить больше внимания структуре, функциональности и удобству использования программного обеспечения.

Известно, что объектно-ориентированное программирование является одним из наиболее мощных инструментов для получения универсального, легко расширяемого и оптимизированного программного обеспечения. В данной работе рассматривается новая библиотека классов для моделирования химической кинетики в различного рода теплофизических задачах, например, распространение пламени в свободном пространстве или пористой засыпке, газовый разряд в химически реагирующих системах и т. д. Разработанная с использованием современных технологий объектно-ориентированного программирования библиотека обладает высокими универсальностью и скоростью вычислений. Одной из примечательных ее особенностей является то, что универсальность и гибкость были достигнуты не в ущерб скорости работы, а за счет оригинального алгоритма взаимодействия блоков библиотеки и тщательной проработки программного интерфейса пользователя.

Предложенная библиотека классов предназначена для использования в разрабатываемых пользователем программах, имеет простой и понятный интерфейс и позволяет легко организовать вычисление термодинамических свойств химических веществ и моделирование газофазной химической кинетики.

Институт тепло- и массообмена им. А. В. Лыкова НАН Беларуси. 220072, г. Минск, ул. П. Бровки, 15; э-почта: migoun@itmo.by. Поступила 23.09.2004.



Структура библиотеки

В качестве языка реализации был выбран язык программирования C++.

Основные принципы. При проектировании библиотеки классов преследовались две основные цели — достичь высоких универсальности и скорости вычислений. Последняя была достигнута разбиением библиотеки на множество функционально-законченных блоков с последующей их оптимизацией. Для достижения высокой универсальности был разработан специальный алгоритм, который будет описан ниже.

На схеме показана структура разработанной библиотеки классов. Блок 1 содержит информацию о химических элементах и набор функций для эффективной работы с ними. Блок 2 отвечает за создание объектов, производящих вычисление термодинамических свойств химических компонент. Созданные блоком 2 объекты хранятся в блоке 3 вместе с набором функций для эффективной работы с базой химических компонент. Блок 4 отвечает за создание объектов, производящих вычисление транспортных свойств веществ. Объекты, вычисляющие свойства элементарных химических реакций контролируются блоком 5. Блок 6 служит хранилищем для объектов, созданных блоком 5. Блоки 2, 4 и 5 разделены на несколько подблоков для достижения гибкости системы и возможности более глубокой оптимизации вычислений для каждого конкретного случая.

Новые технические решения. Как было отмечено выше, для достижения универсальности и гибкости библиотеки был разработан специальный алгоритм, который обеспечивает создание экземпляров объектов в зависимости от типа данных во входном потоке. Таким образом, алгоритм представляет собой виртуальный конструктор объектов и выглядит следующим образом:

1) регистрация всех типов объектов;

2) анализ входного потока каждым типом объектов с возвращением весового коэффициента;

3) выбор типа объекта с наибольшим весовым коэффициентом;

4) создание конкретного экземпляра объекта и его инициализация данными, считанными из потока;

5) повторение п. 2)—4) пока не достигнут конец входного потока.

Ядром разработанного алгоритма является класс `CFactory`, который обеспечивает создание объектов из входного текстового потока. Интерфейс класса выглядит следующим образом:

```
class CFactory {
public:
virtual ~CFactory();
static bool Register(LPESTIMATE estimate, LPCREATE create, LPID id);
protected:
CFactory() {};
static CFactory* Create(istream& in, int& nError, ostream *pLog);
};
```

Каждый тип данных или, другими словами, формат входных данных, определяется тремя обязательными сервисными функциями, указатели на которые определены следующим образом:

```
typedef int (*LPESTIMATE) (istream& in, ostream* pLog);
typedef CFactory* (*LPCREATE) (istream& in, UINT& nError, ostream* pLog);
typedef const string& (*LPID) ();
```

`LPESTIMATE` представляет собой указатель на функцию, которая анализирует входной поток с текущей позиции и возвращает весовой коэффициент, представляющий собой вероятность присутствия определенного типа данных в анализируемом входном потоке. Как правило, весовой коэффициент отражает количество

признаков, идентифицирующих конкретный формат. После анализа потока функция обязана восстановить его состояние. LPESTIMATE указывает на функцию, которая создает объекты данного типа и инициализирует его считанными из потока значениями. Третий параметр представляет собой указатель на функцию, возвращающую текстовое название данного формата. Этот параметр был введен для обеспечения подробного протоколирования процесса анализа и создания объектов, что облегчает поиск ошибок в формате входных данных.

Отметим, что класс `CFactory` имеет защищенный конструктор и не предназначен для прямого использования.

Рассмотрим далее, как работает описанный алгоритм на примере вычисления термодинамических свойств химических компонент.

Практически все популярные базы данных термодинамических свойств используют форматированное текстовое представление данных для вычисления термодинамических свойств [3, 6, 7]. Таким образом, требуется разработать такой способ организации вычислений термодинамических свойств, который, по возможности, избавит конечного пользователя от необходимости вдаваться в подробности каждого из форматов данных и предоставит единый универсальный программный интерфейс, не зависящий от используемого формата данных. Очевидно, что для этих целей подходит разработанный нами класс `CFactory`. Осталось определить класс, открыто наследующий от него и описывающий общий интерфейс для функционально аналогичных типов объектов, в данном случае различного формата химических компонент. Для описания свойств химических веществ был разработан чисто виртуальный класс `CSpecie` с интерфейсом

```
class CSpecie : public CFactory {
public:
static CSpecie* Create(istream& in, int& nError, ostream *pLog);
virtual ~CSpecie() {};
virtual string Name() const = 0;
virtual double Charge() const = 0;
virtual double Weight() const = 0;
virtual int Phase() const = 0;
virtual double Entropy(double dT) const = 0;
virtual double Enthalpy(double dT) const = 0;
virtual double HeatCapacity(double dT) const = 0;
protected:
CSpecie() {};
CSpecie(const CSpecie&);
};
```

Класс `CSpecie` также не предназначен для непосредственного инстанциирования и служит только для задания единого интерфейса для всех объектов, описывающих свойства химических компонент.

Конкретная реализация функциональности каждого поддерживаемого типа данных содержится в классах, открыто наследующих от `CSpecie`. Ниже для примера приведены интерфейсы классов, осуществляющих работу с форматом CHEMKIN [3]:

```
class CChemkinSpecieLite : public CSpecie
{
public:
virtual ~CChemkinSpecieLite() {};
static int Estimate(istream& in, ostream* pLog);
static CSpecie* Create(istream& in, UINT& nError, ostream* pLog);
static const string& ID() { return m_strID; };
virtual string Name() const { return m_strName; };
virtual string Formula() const;
virtual string Comment() const { return m_strComment; };
virtual double Charge() const { return m_nCharge; };
virtual double Weight() const { return m_dWeight; };
virtual int Phase() const { return m_nPhase; };
virtual double Entropy(double dT) const;
virtual double Enthalpy(double dT) const;
virtual double HeatCapacity(double dT) const;
```

```
protected:
CChemkinSpecieLite() {};
CChemkinSpecieLite(const CChemkinSpecieLite&);
};
class CChemkinSpecie : public CChemkinSpecieLite
{
public:
virtual ~CChemkinSpecie() {};
static int Estimate(istream& in, ostream* pLog);
static CSpecie* Create(istream& in, UINT& nError, ostream* pLog);
static const string& ID() { return m_strID; };
virtual double Entropy(double dT) const;
virtual double Enthalpy(double dT) const;
virtual double HeatCapacity(double dT) const;
protected:
CChemkinSpecie() {};
CChemkinSpecie(const CChemkinSpecie&);
};
```

Основное отличие этих двух классов в том, что первый работает с одно-интервальными аппроксимациями термодинамических свойств, а второй наследует обработку одного интервала и дополнительно реализует обработку второго.

Видно, что каждый класс содержит по три статические сервисные функции. Перед использованием каждый класс, описывающий определенный формат данных, должен быть зарегистрирован вызовом функции `Register`, например:

```
CFactory::Register(CChemkinSpecie::Estimate, CChemkinSpecie::Create,
CChemkinSpecie::ID);
```

При этом указатели на сервисные функции для каждого типа данных сохраняются во внутренних структурах библиотеки для использования в алгоритме идентификации объектов во входном потоке. Все встроенные классы проходят автоматическую регистрацию при старте пользовательской программы.

Класс `CSpecie` переопределяет виртуальный конструктор `CFactory::Create`, что делает интерфейс более удобным, а оператор `dynamic_cast` внутри функции обеспечивает динамический контроль типов во время исполнения, что обуславливает безопасность кода и делает его более надежным.

Рассмотрим работу описанного алгоритма на примере простейшей пользовательской программы (для упрощения некоторые строки кода опущены):

```
ifstream stream("input.dat");
vector* species;
while (stream.good()) {
if (CSpecie* pSpecie = CSpecie::Create(stream, nError))
species.push_back(pSpecie);
else
break;
}
for (i=0; i.size(); i++) {
cout < "H298(" < species[i]-Name() < ") = ";
cout < species[i]-Enthalpy(298) < endl;
}
```

Предположим, что текстовый файл input.dat содержит описания свойств химических компонент в различных форматах, поддерживаемых разработанной библиотекой. В первом цикле программы вызывается виртуальный конструктор CSpecie::Create, который создает объекты, описывающие свойства компонент. В случае успешного создания объекта указатель на него помещается в массив species. Выход из цикла осуществляется при ошибке создания или при достижении конца файла. Во втором цикле в стандартный поток выводится имя компоненты и ее безразмерная энтальпия при 298 K. Видно, что благодаря унифицированному интерфейсу работа со всеми типами компонент производится одним и тем же способом. При этом фактическая реализация вычислений для разных форматов данных скрыта от пользователя. Более того, каждый раз, когда пользователь запрашивает свойства какой-либо компоненты, будет вызвана именно та

Способы задания констант скоростей реакций

Вид зависимости	Вычислительные затраты	Среднее количество реакций, %
$k = A$	Простое присваивание	33
$k = AT^{\beta}$	Одно умножение и одно возведение в степень	6
$k = A \exp\left(-\frac{E_a}{RT}\right)$	Три умножения и одно возведение в степень	33
$k = AT^{\beta} \exp\left(-\frac{E_a}{RT}\right)$	Четыре умножения и два возведения в степень	28

функция, которая оптимизирована для работы с тем типом данных, в котором эти свойства были определены во входном файле, чем и достигается высокая скорость вычислений.

Рассмотрим, как данный алгоритм применяется к расчету констант скорости реакций. Известно, что наиболее распространенным способом задания констант скоростей реакции является задание коэффициентов модифицированного выражения Аррениуса. При этом возможны четыре случая, показанные в таблице, где k — константа скорости; A — предэкспонента; T — температура; β — показатель степени; E_a — энергия активации; R — универсальная газовая постоянная.

Видно, что вычислительные затраты на расчет константы скорости колеблются от простого присваивания до четырех умножений и двух возведений в степень. В то же время, простой статистический анализ кинетических механизмов показал (см. таблицу), что количество реакций с постоянной константой скорости составляет треть от общего числа реакций в механизме, а реакции с полным выражением Аррениуса — только 28%. Становится очевидным, что нет необходимости вычислять полное выражение для константы скорости, если один или два из коэффициентов равны нулю. Данную проблему можно решить, используя условные операторы. Однако в этом случае мы теряем в универсальности и гибкости, так как каждый раз при внесении изменений необходимо будет перекомпилировать весь проект. Более того, этот подход противоречит объектно- ориентированным принципам программирования. В разработанной библиотеке эта проблема решена с помощью виртуального конструктора и виртуальных функций. Виртуальный конструктор создает различные объекты реакций, строго соответствующие каждому конкретному блоку входных данных, а виртуальность функций для вычисления констант скорости и других свойств реакций обеспечивает унификацию интерфейса, делая незаметными для пользователя детали реализации, как это было показано на примере вычисления термодинамических свойств химических компонент.

Отметим, что при необходимости библиотека может быть легко расширена для поддержки любых других форматов данных. Для этого достаточно спроектировать класс, открыто наследующий от класса `CFactory` или его производных, снабдить этот класс тремя описанными выше сервисными функциями и зарегистрировать его перед первым использованием функций библиотеки. Примечательно то, что для выполнения этой процедуры не нужно перекомпилирования всей библиотеки, что очень важно для больших программных систем, компиляция и сборка которых может длиться несколько часов.

Несмотря на то, что вся библиотека написана на языке программирования C++, ее можно использовать и в программах, написанных на других языках программирования. Однако в этом случае может понадобиться разработка специального программного интерфейса, который будет обеспечивать связь между объектно-ориентированной и линейной частями программы.

Заключение. На примере разработки библиотеки классов для моделирования химической кинетики продемонстрирован современный подход к созданию научно-исследовательского ПО. Разработка библиотеки, начатая несколько лет назад, производится в Отделе неравновесных процессов Института тепло- и массообмена им. А. В. Лыкова НАН Беларуси. К настоящему моменту разработана уже седьмая версия библиотеки, обеспечивающая расчет термодинамических свойств с поддержкой трех основных форматов данных и расчет химической кинетики для нескольких типов элементарных химических реакций.

Предложенный программный продукт предназначен для широкого использования. В связи с этим все заинтересованные в использовании или участии в дальнейшей разработке рассматриваемого продукта приглашаются к обсуждению по электронной почте любых связанных с данной темой вопросов. Контактный адрес электронной почты: migoun@itmo.by.

Описанная библиотека была использована при разработке программы для работы с базами данных термодинамических свойств химических веществ. Демонстрационная версия программы, а также дополнительная и контактная информация доступны через Интернет [8].

Литература

1. **FLUENT** Inc. CFD flow modeling software. http://www.fluent.com/
2. **StarCD** software. http://www.cd-adapco.com/products/starsolver.htm
3. **Kee R. J., Rupley F. M., and Miller J. A.** Chemkin-II: A Fortran Chemical Kinetics Package for the Analysis of Gas-Phase Chemical Kinetics. Sandia Report SAND89-80090. UC-401, Livermore. September 1989.
4. **Numerical** Algorithms Group. http://www.nag.co.uk/
5. **Numerical** Recipes. http://www.nr.com/
6. **NIST** Chemistry Webbook. http://webbook.nist.gov/chemistry/
7. **McBride B. J. and Sanford G.** Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications: II. Users Manual and Program Description. NASA Reference Publication 1311, June 1996.
8. **Department** of nonequilibrium processes. http://www.dnp.itmo.by/projects/ckcl.html