Journal of Engineering Physics and Thermophysics, Vol. 78, No. 1, 2005

OBJECT-ORIENTED APPROACH TO SIMULATION OF CHEMICAL REACTION KINETICS

A. N. Migun, E. A. Matveichik, A. P. Chernukho, and S. A. Zhdanok

UDC 53.072.001.57

The structure and main elements of realization of a library of classes for simulation of the kinetics of chemical reactions are considered. Due to a modern approach used in the development of the library, it is characterized by high universality and adaptability and provides a high rate of calculations.

Introduction. As a result of the rapid development of computers in the last few decades, numerical simulation has become a powerful investigation tool that logically supplements experimental investigations. Moreover, modern experimental investigations cannot be carried out without recourse to a complex computational hardware provided with a corresponding software. In some cases, simulation is the only possible method of obtaining some insight into the details of a process. This gave birth to an industry involved in the development and adoption of various investigation program products. The computational programs proposed in the current software market can be conditionally divided into application programs and special subprograms. An application program represents one or several completed mathematical models having a user interface. Such a program begins to operate once the initial data are introduced into it. Application programs are developed for simulation of concrete apparatus and processes. Examples of application programs are the FLUENT [1], StarCD [2], and CHEMKIN [3] programs. Libraries of special subprograms also represent logically completed mathematical models; however, they, unlike the application programs, are developed for special calculations and have only a program interface for making a call from the user programs. Examples of libraries of special subprograms are the NAG [4] and Numerical Recipes [5] programs, as the well as old versions of the CHEMKIN programs.

In the majority of cases, scientific-research programs and libraries of subprograms are written in various versions of the programming language Fortran. However, Fortran is a linear language and thus is limited in application, e.g., it gives no way of constructing complex programs on the basis of modern approaches. Moreover, commercial software, as a rule, represents "a black box" for the user, into which he cannot introduce any additions or improvements.

Until recently, the main goal of programmers was to increase the rate of calculations. However, the appearance of high-speed computers somewhat diminished the calculation-rate priority and heightened the role of the universality of calculations. Therefore, recently most of the attention has been concentrated on the structure and functions of computational programs and their serviceability.

It is known that object-oriented programming is one of the most powerful tools for obtaining a universal software that could be easily widened and optimized. In the present work, we propose a new library of classes for simulation of the kinetics of chemical reactions in various thermophysical processes, e.g., in a flame propagating in a free space or in a porous batch, in a gas discharge in chemically reacting systems, etc. The object-oriented library of programs developed by us with the use of modern methods is highly universal and provides a high rate of calculation. It is remarkable that the universality and adaptability of this library were attained, without sacrifice of the calculation rate, only due to the use of an original algorithm, providing reciprocity between the library units, and a thoroughly developed user interface.

The library of classes proposed can be used in user-developed programs since it has a simple, intelligible interface and allows one to easily organize the calculation of the thermodynamic properties of chemical substances and simulation of the chemical kinetics of reactions in the gas phase.

A. V. Luikov Heat and Mass Transfer Institute, National Academy of Sciences of Belarus, 15 P. Brovka Str., Minsk, 220072, Belarus; email: migoun@itmo.by. Translated from Inzhenerno-Fizicheskii Zhurnal, Vol. 78, No. 1, pp. 153–158, January–February, 2005. Original article submitted September 23, 2004.

 1062-0125/05/7801-0160

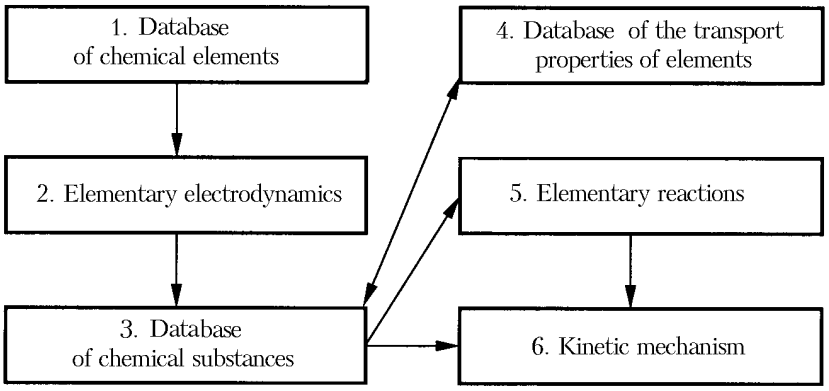

Fig. 1. Structure of the library of classes.

The library considered is written in the programming language C++.

Basic Principles. The library of classes developed is characterized first of all by a high universality and a high rate of calculations. The high calculation rate is attained due to the division of the library into a large number of functionally complete and optimized units. The universality of this library is provided by a special algorithm that will be described below.

Figure 1 shows the structure of the library of classes developed. Unit 1 contains data on chemical elements and functions providing effective use of these data. Unit 2 is responsible for the creation of objects for calculating the thermodynamic properties of chemical elements. The objects created in unit 2 are stored in unit 3 together with a set of functions for effective work with the database on chemical elements. Unit 4 is responsible for the creation of objects for calculating the transport properties of substances. The objects used for calculating the properties of elementary chemical reactions are controlled by unit 5. Unit 6 is a store for objects created in unit 5. Units 2, 4, and 5 are divided into several subunits that provide an adaptibility of the system and a deep optimization of calculations in each concrete case.

New Technical Solutions. As was mentioned above, the universality and adaptibility of the library considered is attained due to the use of a special algorithm that allows one to create objects depending on the type of input data flow. This algorithm comprises a virtual designer of objects and has the following form:

1) display of all types of objects;
2) analysis of the input data flow for each type of object with recovery of the weight factor;
3) selection of the object with the largest weight factor;
4) creation of a concrete object and its initialization with the use of data taken from the initial data flow;
5) repetition of operations 2–4 until the input data flow ends.

The core of the algorithm developed is the `CFactory` class, which provides the creation of objects from the input data flow. The interface of this class is as follows:

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

Each type of data or, in other words, the input data format, is determined by three obligatory service functions having the following designators:

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

LPESTIMATE designates the function that analyzes the input data flow from the current position and recovers the weight factors representing the probability of the presence of a definite type of data in the input data flow analyzed. As a rule, the weight factor points to a number of features identifying a concrete format. After the data flow has been analyzed, the function recovers its state. LPCREATE designates the function that creates objects of a definite type with the use of data from the initial data flow. The third parameter designates the function recovering the text name of the format considered. This parameter was introduced to provide detailed recording of the process of analysis and creation of objects, which makes the search for errors in the input data format easier.

Note that the CFactory class has a protected designer and cannot be used directly.

We will consider the operating of the algorithm proposed with the example of calculating the thermodynamic properties of chemical elements.

In practically all of the popular databases on the thermodynamic properties of chemical elements, the data are presented in the formatted-text form [3, 6, 7]. Thus, it is necessary to develop a method of calculation of the thermodynamic properties of elements that would allow the end user to do away with the details of each of the data formats, i.e., to develop a unique, universal program interface independent of the data format used. The CFactory class developed by us is suitable for this purpose. From this class we may directly determine the class describing the general interface for functionally similar objects, differently formatted chemical elements in our case. For determining the properties of chemical substances, we have developed the purely virtual class CSpecie with the interface

```
class CSpecie: public CFactory {
public:
  static CSpecie* (istream& in, nError, ostream *pLog);
  virtuel ~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&);
  };
```

The CSpecie class also cannot be used for direct instantiation and serves only for prescription of a unified interface for all objects featuring the properties of chemical elements.

The functions providing the use of each type of data are concretized in classes following directly from the CSpecie class. Below, we present, as examples, interfaces of classes working with the CHEMKIN format [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 () {return m_strName; };
  virtual string Formula () const;
  virtual string Comment () {return m_strComment; };
  virtual double Charge () {return m_nCharge; };
  virtual double Weight () {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 CChemkinSpecieLite: public CChemkinSpecieLite
{
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 double Entropy (double dT) const;
    virtual double Enthalpy (double dT) const;
    virtual double HeatCapacity (double dT) const;
protected:
    CChemkinSpecieLite () {};
    CChemkinSpecieLite (const CChemkinSpecieLite&);
    };
```

The main difference between these two classes is that the first of them works with single-interval approximation of thermodynamic properties, and the second class continues the treatment of the first interval and, additionally, treats the second interval.

It is evident that each class contains three static service functions. Before use, each class representing a concrete data format should be registered by calling the function `Register`, for example:

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

In this case, the designators of service functions for each type of data are retained in the internal structures of the library and are used in the algorithm for identifying objects in the input data flow. All the classes introduced are automatically registered when the user program starts.

The `CSpecie` class overdetermines the virtual designer `CFactory: Create`, which makes the interface more convenient, and the `dynamic_cast` operator of the function exercises dynamic control over the types of data in the process of operation, which provides safety of the code and makes it more reliable.

We will consider the operation of the above-described algorithm with the example of a simple user program (some rows of the code are omitted for simplicity):

```
ilfstream stream (input.dat);
vector <CSpecie*> species;
while (stream.good ()) {
if (CSpecie* pSpecie = CSpecie:: Create (stream, nError))
species.push_back (pSpecie);
else break;
}
for (i = 0; i <species.size (); i++) {
cout (<<"H298 ("<<species [1] → Name () << ") = ";
cout <<species [1] → Enthalpy (298) << endl;
}
```

Let us assume that the `input.dat` text file contains data on the properties of chemical elements in various formats involved in the library developed. In the first cycle of the program, the virtual designer `CSpecie::Create` is called. This designer creates objects featuring the properties of elements. If an object is created successfully, its designator is put into the species file. The cycle is terminated if an object is created with an error or when the file ends.

TABLE 1. Methods of Prescription of the Rate Constants of Reactions

Form of relation	Volume of calculations	Average number of reactions, %
$k = A$	Simple assignment	33
$k = AT^{\beta}$	One multiplication and one exponentiation	6
$k = A \exp\left(-\frac{E_a}{RT}\right)$	Three multiplications and one exponentiation	33
$k = AT^{\beta} \exp\left(-\frac{E_a}{RT}\right)$	Four multiplications and two exponentiations	28

In the second cycle, the name of an element and its dimensionless enthalpy at 298 K are introduced into the standard data flow. It is evident that, due to the unified interface, all the elements are processed by one and the same method. In this case, the virtual calculations realized in different data formats are unnoticed by a user. Moreover, when a user calls the data on the properties of any element, he will obtain a function that is optimized for the work with the data to which these properties are assigned in the input file, which provides a high rate of calculations.

We will show how the algorithm proposed can be used for calculating the rate constants of a reaction. It is known that the rate constants of reactions are most frequencly determined through the coefficients of the modified Arrhenius relation used in four forms presented in Table 1, where k is the rate constant, A is a preexponent, T is temperature, β is an exponent, E_a, is the activation energy, and R is the universal gas constant.

It is seen that the calculation of the rate constants of reactions can be reduced to the simple assignment of values or be as large as four multiplications and two raisings to a power. At the same time, simple statistical analysis of the kinetic mechanisms (see Table 1) shows that the number of reactions occurring with a constant rate is equal to a third of the total number of reactions occurring by a certain mechanism, and the reactions defined by the complete Arrhenius relation account for 28% of all the reactions. It is evident that there is no need for calculating the rate constants by the complete Arrhenius formula if one or two coefficients are equal to zero. This problem can be solved with the use of conditional operators. However, the universality and adaptability of the library decrease in this case because of the necessity of recomplicating the whole project on introduction of changes. Moreover, this approach is in contradiction with the object-oriented principles of programming. In the library developed, this problem is solved with the use of a virtual designer and virtual functions. The virtual designer creates various reaction objects that strictly correspond to each concrete input data block, and the virtual functions used for calculating the rate constants and other properties of reactions unify the interface and, in doing so, make the details of calculations unnoticed for a user. This was demonstrated above by the example of calculating the thermodynamic properties of chemical elements.

The library proposed can be widened if necessary for other data formats. To do this, it will suffice to create a class directly from the CFactory class or its derivatives, to provide this class with the above-described service functions, and to register it prior to the first use of the file functions. It is remarkable that this procedure does not call for the recompilation of the whole library, which is very important for large program systems whose compilation and assembly can take several hours.

Despite the fact that the whole library is written in the C++ programming language, it can also be used in programs written in other programming languages. However, in this case, the need for a special program interface that would provide a connection between the object-oriented and linear parts of the program may arise.

CONCLUSION

A modern approach to the development of scientific-research computational programs has been demonstrated with the example of development of a library of classes for simulation of the kinetics of chemical reactions. The development of the library was begun several years ago and is being continued at the Section of Nonequilibrium Processes of the A. V. Luikov Heat and Mass Transfer Institute of the National Academy of Sciences of Belarus. At present, the seventh version of the library has been developed. This version allows one to calculate the thermodynamic properties of chemical elements, using three main data formats, and the kinetics of several types of elementary chemical reactions.

Since the library of classes proposed can be widely used, we invite all persons who are interested in it or in its further development to discuss any problems concerning this library by e-mail: migoun@itmo.by.

The above-described library of classes was used for the development of a program for determining the thermodynamic properties of chemical substances. A demonstration version of the program as well as additional and contact information can be obtained from the Internet [8].

REFERENCES

1. FLUENT Inc. CFD Flow Modeling Software. http://www.fluent.com.
2. StarCD Software. http:// www.cd-adapco.com/products/starsolver.htm.
3. R. J. Kee, F. M. Rupley, and J. A. Miller, 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. B. J. McBride and G. Sanford, 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.