ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ МОДЕЛЬ ПОСТРОЕНИЯ АЛГОРИТМОВ ОБРАБОТКИ СИГНАЛА

Слепов А.В.
Алтайский Государственный Университет
sav@uic.asu.ru, barn_czn@ab.ru

Работа над созданием или тестированием нового алгоритма обработки сигнала всегда связана с рутиной работой постоянного изменения кода, добавления новых переменных, изменения структуры самого алгоритма. При работе с целым множеством разных алгоритмов задача отладки проектируемого алгоритма становиться практически невыполнимой. В этой ситуации имеет смысл отказаться от традиционного представления алгоритма обработки сигнала в виде процедуры с входными и выходными параметрами. Традиционная схема некоторого алгоритма обработки сигнала показана на Рис. 1: входными данными множества процедур вычисляющих соответствующие параметры является входной сигнал и множество глобальных переменных. Результатом работы является при этом одна или несколько переменных принадлежащих все тому же множеству глобальных переменных. Столь неудобная с точки зрения отладки структура кода и данных является наиболее эффективной в вычислительном плане: 1) процедуры, использующие одни и те же параметры, могут обращаться к уже вычисленным переменным, 2) исключаются затраты на помещение входных аргументов в стек.

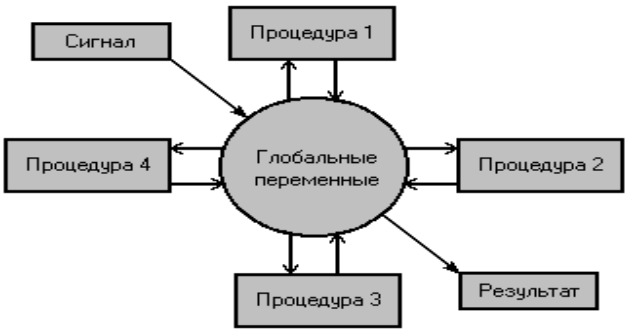

Рис.1 Традиционная схема доступа к данным в процессе обработки сигнала.

В идеальном случае, как известно, максимально эффективным является алгоритм, где каждая переменная является глобальной переменной. С точки зрения надежности сохранения данных алгоритм обработки сигнала предлагается организовывать в виде самостоятельного класса с тремя основными методами:

```
class CDigitalProcAlgoritm
        {     ...
         public:
         //конструктор
     CDigitalProcAlgoritm();
   //обновление данных класса
          Update();
         //деструктор
    ~CDigitalProcAlgoritm();
          }
```

Если наш класс-алгоритм использует при своей работе другие алгоритмы, то их необходимо включить в список членов (полей) данного класса:

```
                class CDigitalProcAlgoritm
                        {
CDigitalProcAlgoritm_1      DigitalProcAlgoritm_1;
CDigitalProcAlgoritm_2      DigitalProcAlgoritm_2;
CDigitalProcAlgoritm_3      DigitalProcAlgoritm_3;
                       ...
```

Так как главный класс использует в своей работе результаты вычислений других алгоритмов т.е. объектов DigitalProcAlgoritm_1, DigitalProcAlgoritm_2 и DigitalProcAlgoritm_3, то он должен позаботиться об обновлении их состояния в своем методе Update():

```
CDigitalProcAlgoritm::Update()
                {
  DigitalProcAlgoritm_1.Update();
  DigitalProcAlgoritm_2.Update();
  DigitalProcAlgoritm_3.Update();
```

Однако обновление состояний объектов может произойти и в другом месте, в другом параллельно работающем алгоритме. Для этого необходим механизм доступа к данным других алгоритмов, который должен обеспечивать эффективность кода и целостность данных. Предположим, что для работы алгоритмов DigitalProcAlgoritm_1-3 необходимы значения входного сигнала находящиеся в буфере. Каждый из этих алгоритмов может иметь свой буфер сигнала и заботится о его обновлении. Однако более эффективным является решение с одним буфером сигнала, но так чтобы каждый метод «думал» что он работает со своей копией буфера сигнала. Такой метод доступа к данным можно организовать с помощью существующего механизма ссылок и указателей, который существует во всех языках программирования высокого уровня [1].

На Рис. 2 показан пример взаимодействия двух алгоритмов. Алгоритм DspAlgoritm_1 использует две переменных, одна из которых полностью принадлежит этому классу, а вторая является ссылкой на член другого класса. Таким образом, все внутренние параметры алгоритмов делятся по уровню доступа на две группы: параметры доступные на чтение-запись и параметры, доступные только на чтение. Естественно при этом появляется необходимость при инициализации алгоритма указывать уровень доступа к некоторым параметрам. При выполнении обновления состояния параметров также нужна проверка уровня доступа. Можно сформулировать основные преимуществами такой идеологии программирования алгоритмов:

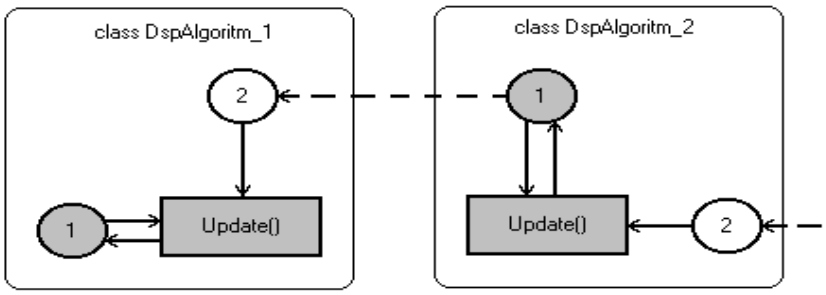

Рис 2. Схема взаимодействия между двумя классами алгоритмов. Пунктирная линия указывает на обращение к данным по ссылке.

- связь между классами алгоритмов и их параметрами полностью определяется на этапе инициализации и остается неизменной;
- вопрос о необходимости обновления того или иного параметра решается проверкой флага уровня доступа к переменной;
- отладка составного алгоритма может быть произведена уже на этапе инициализации.

Перечисленные достоинства определяют также и недостатки:

- увеличение времени проектирования сравнительно простых процедур обработки;
- необходимость следования описанным правилам при построении библиотеки классов алгоритмов.

ЛИТЕРАТУРА

1. Керниган Д. и др. Язык программирования Си. М.: Финансы и статистика, 1992.