*Математические структуры и моделирование*
1999. Вып. 3, с.65-69.

УДК 530.51

# КОЛЛЕКТИВНАЯ РАЗРАБОТКА КРИТИЧНЫХ ПО БЫСТРОДЕЙСТВИЮ ПРИЛОЖЕНИЙ

**Д.В. Усов**

What is the problems of traditional team application development? The object models based on dinamic linking is of benefit to simplify developers interaction and speed up a work. But the popular complex technologies such as COM consume a lot of CPU and memory resource. In given paper new simple and effective object modes is discribed.

## 1. Проблемы традиционной коллективной разработки приложений

Традиционным или наиболее часто используемым можно назвать следующий способ разработки Windows-приложений:

В качестве языка используются популярные языки высокого уровня, поддерживающие объектно-ориентированный подход: C++, Pascal, Java.

Чаще всего в качестве инструментального средства используются интегральные среды (Integrated Development Environment) разработки - продукты фирм-лидеров производства программного обеспечения: Microsoft и Borland. Microsoft предлагает универсальную среду для всех языков программирования - Developer Studio. Borland поставляет с каждым языком свою оболочку: Borland Delphi, Borland C++ Builder, Borland Java Builder.

Поскольку необходимым требованием при создании коммерческих и не только коммерческих приложений является уменьшение сроков разработки, приложения создаются, как правило, коллективом программистов, иногда довольно большим.

При коллективной работе над приложением с использованием описанных выше стандартных средств возникают следующие проблемы:

1. Разработчики должны формировать и строго придерживаться общей архитектуры приложения. Без этого часть программы, написанная одним разработчиком, становится недоступной для остальных.



E-mail: usov@iitam.omsk.net.ru
Омский филиал Института математики СО РАН

2. Необходимо средство или технология контроля версий программных модулей. Часто для этих целей используют системы типа CVS (Concurrent Version System) компании Signum Support.

3. Каждый разработчик должен иметь на своем рабочем месте экземпляр исходных файлов всего приложения. При изменении одного исходного файла сборка двоичного исполняемого файла (линковка) производится заново. Чем больше приложение, тем больше времени занимает компиляция и линковка, что замедляет разработку.

4. Для избежания конфликтов нужно придерживаться соглашений об именах переменных, классов, функций и т. д. Иногда это ограничение отрицательно влияет на читабельность исходных текстов.

## 2. Возможности систем динамического связывания компонетов

Стандартный линкер таких языков, как C++, Pascal осуществляет статическое связывание программных компонентов. То есть при сборке двоичного исполняемого файла в места использования переменных и функций подставляются их адреса относительно начала файла. Напротив, при динамическом связывании программные модули загружаются в память по мере необходимости и, соответственно, их адреса являются непостоянными и образуются во время исполнения программы. Для получения адресов загружаемых объектов обычно используется механизм динамически загружаемых библиотек DLL (Dynamic Load Library). Существует несколько технологий использования динамического связывания компонентов [1]. Наиболее распространенная из них - модель компонентных объектов COM (Component Object Model) фирмы Microsoft [2]. Другие известные технологии - CORBA, OpenDoc и т. д. аналогичны в том, что касается вопросов коллективной разработки приложений. Поэтому далее ограничимся кратким рассмотрением COM.

Технология COM создана для преодоления указанных выше трудностей коллективной разработки приложений, а также для создания распределенных приложений и внедрения (в визуальном смысле) одних приложений в другие.

В соответствии с COM, приложение состоит из множества компонентов, связывание которых происходит динамически во время исполнения приложения при помощи вызовов специальных процедур реализации COM (в настоящее время на компьютерах архитектуры Intel используется OLE2). Это позволяет подключать и заменять компоненты без перекомпиляции и линковки.

Программные компоненты идентифицируются с помощью 128-битного кода. Специальные программы генерации этого псевдослучайного кода гарантируют его уникальность с очень большой вероятностью. Методы компонента имеют строго определенный двоичный интерфейс. Такая идентификация программных компонентов позволяет внутри каждого компонента не заботиться об уникальности названий классов и методов.

Проблема контроля версий решается в COM следующим образом: каждая

версия компонента имеет свой уникальный идентификатор. То есть формально новая версия компонента это совершенно другой компонент, который не всегда может использоваться вместо старого, и наоборот.

Компоненты COM поставляются и используются в виде двоичного исполняемого кода (обычно в виде DLL). Поэтому разработчики разных компонентов одного приложения могут поставлять друг другу только готовые к работе части приложения, а не комплект исходных файлов, которые нужно компилировать. Причем, если программный интерфейс новой версии компонента изменился, а другие компоненты еще не адаптированы под эти изменения, они могут корректно продолжать использовать старую версию компонента.

COM имеет еще много возможностей, не относящихся непосредственно к коллективной разработке приложений, поэтому в данной работе они не рассматриваются.

## 3. Трудности применения COM для разработки критичных по быстродействию приложений

COM-технология, являясь мощным средством построения распределенных и внедряемых приложений, исполняется намного медленнее, чем обычное статическое связывание объектов методом компиляции и линковки. Для многих приложений, не требовательным к ресурсам, например программы пользовательского интерфейса, это не так важно. Но если приложение производит сложные вычисления, вызывая методы из подключенных компонентов, применение COM может оказаться неприемлемым. Так, если число операций некоторого метода вычисления зависит от n-ой степени числа используемых компонентов, то увеличение времени вызова одного метода компонента в 2 раза увеличивает общее время вычисления алгоритма в $2^n$ раз. Особенно долго выполняются операции создания и удаления компонентов.

Кроме вычислительных ресурсов, COM-компоненты отнимают у операционной системы гораздо больше оперативной памяти по сравнению со статическим связыванием. В условиях недостатка физической памяти система использует виртуальную, что сильно замедляет работу приложения.

Другие сложности применения COM: написание и отладка компонентов COM сильно отличается от написания обычных классов объектно-ориентированного языка и требует от программиста дополнительных знаний и значительных усилий.

## 4. Динамическое связывание без применения громоздких технологий

Трудности применения громоздких COM-подобных заставили искать менее ресурсоемкие пути решения проблем коллективной разработки приложений. Ис-

следования показали, что применение при описании классов языка C++ дополнительных макросов и некоторые служебные функции позволяют подключать объекты, находящиеся в DLL во время исполнения. При этом реализуются и многие свойства COM-технологии, помогающие при коллективной разработке приложений.

1. Разработка компонентов ведется независимо.

2. При модификации одного компонента не требуется линковка всего приложения.

3. Классы C++ идентифицируются 128-битным идентификатором, что гарантирует уникальность.

4. Контроль версий осуществляется способом, аналогичным COM.

5. Возможно получение набора классов объектов по некоторому признаку. Так, разработчик приложения может использовать класс объектов, даже не зная в процессе разработки, что данный класс существует.

Предлагаемая технология вместе с тем сохраняет преимущества статического связывания классов:

1. Высокая скорость исполнения программ. Дополнительные затраты связаны только с загрузкой/выгрузкой DLL и вызовами виртуальных функций вместо обычных.

2. Малые затраты памяти. Дополнительные затраты памяти по сравнению со статическим связыванием практически отсутствуют.

3. Поддерживается обычное для объектно-ориентированных языков наследование классов (технология COM наследование компонентов не поддерживает). Важно, что при изменении версии базового класса автоматически меняются версии всех порожденных.

4. Классы-компоненты описываются в рамках языка C++ обычных образом, а не специальных языком описания компонентов.

Пример описания класса:

```
class TSMBase : public TObjBase
{ DECLARE_CLASSID1(«SM_v1.6»,TObjBase);
public:
TSMBase() { next = NULL;};
virtual ~TSMBase() {};
virtual unsigned long int IsA();
virtual void ResetContactCounter(){};
virtual TBaseContact* GetNextContact();
TBaseContact* GetNextIContact();
TBaseContact* GetNextOContact();
virtual void CreateCtrlWindow(TWindow* parent){};
virtual char* GetTitle(char* rez);
virtual char* GetPrivateMenuItem();
protected:
TSMBase* next;
friend TSMContainer;
```

```
public:
void* EditorInfo;
};
```

Как видно, основное отличие от обычного описания класса на C++ - макрос DECLARE_CLASSID1, которому в качестве параметра передаются данные для получения идентификатора класса и имя базового класса. Другое не столь заметное отличие - все функции объявлены как виртуальные.

Для получения доступа к объектам используется механизм виртуальных таблиц компиляторов C++. Это механизм недокументирован, однако все компиляторы фирм Borland и Microsoft имеют одинаковую схему расположения объекта в памяти, что уже давно используется разработчиками COM-приложений. Данная технология была опробована на компиляторе Borland C++ Builder и доказала свою работоспособность.

## Литература

1. Брюхов Д.О., Задорожный В.И., Калиниченко Л.А., Курошев М.Ю., Шумилов С.С.*Интероперабельные информационные системы: архитектуры и технологии.*// СУБД. – N 4. – 1995.

2. Роджерсон Д. *Основы COM.* – М: Изд. отд. «Русская редакция», 1997.