О некоторых вопросах компонентной архитектуры программного приложения

Алексей Игнатенко

Что такое компонента (plug-in)?

- Независимый программный модуль, обычно подключаемый на этапе выполнения программы

Преимущества компонентной архитектуры

Улучшение качества и однородности кода

- Четкая инкапсуляция деталей реализации за интерфейсами
- Инкапсуляция сторонних библиотек

Улучшение модульности проекта

- В каждый момент времени идет работа с небольшим подмножеством файлов
- Лучший способ организации больших проектов

Радикальное уменьшение времени сборки

- Не производится обработка внутренних заголовочных файлов для других компонент
- Не требуется пересборка других компонент

Возможность замены компонентов

- Возможность обновления не всей программы, а только набора компонент
- Возможность выбора из нескольких однотипных компонент (конфигурация)

Возможность использования компонент с различными открытыми/закрытыми лицензиями

- Таким образом можно использовать GPL-компоненты и раскрывать их код, не нарушая лицензии

Компонентная архитектура

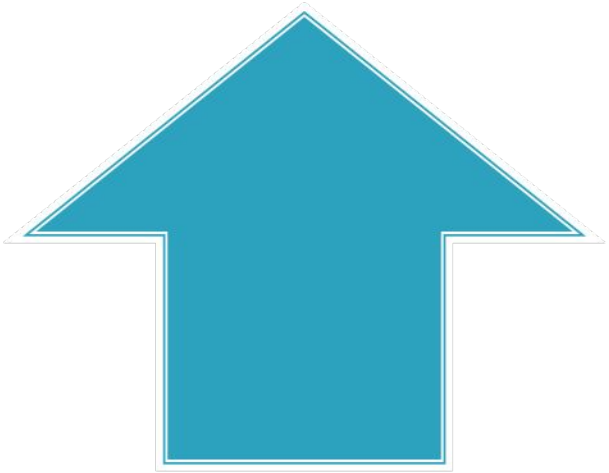
Общий проект разделяется на набор компонент

Компоненты помещаются в отдельные DLL

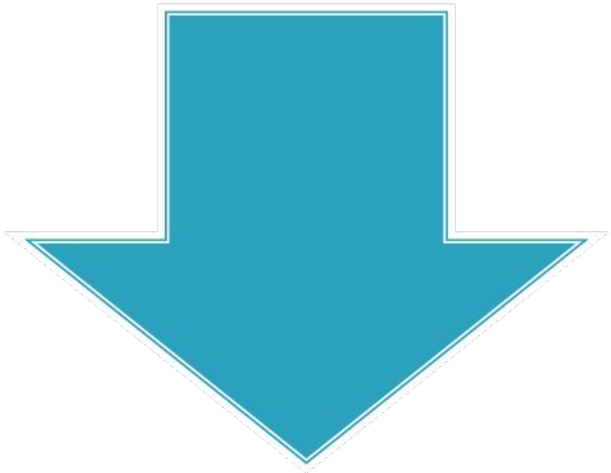
В процессе выделения создаются интерфейсы, которые используется в основном проекте (SDK)

DLL с компонентам подгружаются на этапе выполнения

Отличие обычных DLL от компонент



Обычные DLL связываются на этапе компиляции



Компоненты подгружаются во время выполнения в два этапа:

- Поиск всех компонент (в определенном каталоге)
- Регистрация найденных компонент в определенном месте системы

Простое решение

- DLL:
  - IPlugin *createInstance(const char *);
- Application:
  - IPlugin* pluginInstance = createInstance(“RendererPlugin”);
  - IRenderer* renderer = dynamic_cast<IRenderer>(pluginInstance)

Недостатки простого решения

- Одно из трех:
  - Нарушается безопасность типов
    - static_cast
  - Ограничивается применение плагинов
    - dynamic_cast
  - Необходима разработка сложной и ломкой кастомной RTTI
    - QueryInterface
- Необходимы дополнительные соглашения для поиска однотипных плагинов (по имени, например)

Предлагаемое решение

- Интерфейсы определяется в приложении
- Для интерфейсов применяются соглашения COM
- Плагины регистрируются сами в нужном месте системы

Фабрики для плагинов

```
template<typename Interface>
interface IPluginFactory
{
  virtual STDMETHODCALLTYPE Interface *Create() = 0;
};

interface Renderer
{
virtual void STDMETHODCALLTYPE  BeginScene() = 0;
virtual void STDMETHODCALLTYPE  EndScene() = 0;
};

typedef IPluginFactory<IRenderer> IRendererFactory;
```

Вариант 1, локальный

Application:

```
interface IPluginManager
{
  virtual void STDMETHODCALLTYPE RegisterRenderer(IRendererFactory* in_factory) = 0;

  virtual void STDMETHODCALLTYPE RegisterSceneManager(ISceneManagerFactory* in_factory) = 0;

  // + методы для доступа к плагинам (создание через фабрики)
}
```

DLL:

```
extern "C" void registerPlugins(IPluginManager* in_pluginManager);
```

Вариант 2, глобальный

- Система разрабатывается с нуля для поддержки плагинов

```
interface IKernel: public IInterface
{
  virtual STDMETHODCALLTYPE IImageIOServer* GetImageIOServer() const = 0;
  virtual STDMETHODCALLTYPE IGraphicsServer* GetGraphicsServer() const = 0;
  virtual STDMETHODCALLTYPE ISettingsServer* GetSettingsServer() const = 0;
  ...
}

interface IImageIOServer: public IInterface
{
  virtual STDMETHODCALLTYPE void AddImageReader(IImageReader* in_reader) = 0;
  virtual STDMETHODCALLTYPE HRESULT ReadImage(BSTR in_path);
}
```

```
extern "C" void registerPlugins(IKernel* in_kernel);
```

Проблемы и размышления

- А нужно ли поддерживать возможность работы из разных сред?
- Версии интерфейсов / библиотек
- Суперклассы - да/нет
- Приведение типов
- Подсчет ссылок
- Как искать плагины?
- События

А нужно ли поддерживать возможность работы из разных сред?

- Если нет требований, чтобы плагины и/или основное приложение работали из разных сред, нет смысла поддерживать соглашения COM
- Не нужны STDMETHODCALLTYPE, BSTR и т.п.
- Можно выделять и удалять память в разных DLL (это стоит проверить)
- Более того: можно использовать набор базовых неабстрактных классов и подключать общую библиотеку ко всем плагинам
  - Внимание: но нужно очень четко работать с версиями в этом случае!

Версии интерфейсов / библиотек

- Проверять версии
  - 1) У библиотеки (DLL)
  - 2) У плагина (интерфейса)
- Как проверять?
- Как поддерживать совместимость?
  - Старые плагины должны работать с новыми интерфейсами?
  - Новые плагины должны работать со старыми интерфейсами?

Суперклассы

- Функции:
  - запрос на информацию без создания экземпляра
  - = статические функции
  - Создание объекта = фабрика
- Нужны ли?
  - Накладные расохды на создание/поддержку

Приведение типов

- Dynamic_cast
- QueryInterface

Подсчет ссылок

- AddRef/Release – единственный вариант.
- Есть ли другие возможности? Если нет, почему?

Как искать плагины?

- Перебор файлов в папках
- Конфиг-файл (XML – рекомендуется MS)
- В реестре (пишется инсталлятором)