СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.4’2

РАЗРАБОТКА МЕТОДИКИ МОНИТОРИНГА СОСТОЯНИЯ API ФУНКЦИЙ В НЕУПРАВЛЯЕМЫХ СРЕДАХ

В. А. Рыжков

Воронежский государственный технический университет

Поступила в редакцию 01.04.2015 г.

Аннотация. В данной работе рассматриваются вопросы разработки методики мониторинга состояния API функций в неуправляемых средах, которая позволяет предоставлять информацию о методах, свойствах, процессах и перехваченных событиях..
Ключевые слова: управляемая и неуправляемая среды, программные интерфейсы приложений (API), объектная модель компонента (COM), функция, метод, процесс.
Annotation. This paper deals with the development of a methodology for monitoring the state of API functions in unmanaged environments, which allows you to provide information about the methods, properties, and processes the captured events.
Keywords: managed and unmanaged code, application programming interfaces (API), component object model (COM), function, method, process.

Потребность в создании системы мониторинга API функций любых программных систем связана с ограниченным количеством документации по их API, которая зачастую отстает от текущей версии программного продукта. От этого напрямую зависит скорость развития программных систем, которая достигается за счет доработки этих продуктов, самими пользователями. Это создает предпосылки к разработке методики мониторинга состояния API функций в неуправляемых средах, позволяющей предоставлять информацию о методах, свойствах, процессах и событиях в текущий момент времени. Кроме того, существует возможность построения универсальной системы, подходящей для любой неуправляемой среды, предоставляющей возможность взаимодействия с ней через COM. Рассмотрим основные аспекты данной методики.

Для разработки системы наиболее целесообразным является использование .NETплатформы по двум причинам:

1. Платформа .NET содержит общеязыковую среду выполнения (Common Language Runtime – CLR). В [1] утверждается, что общеязыковая среда выполнения CLR поддерживает управляемое выполнение, которое характеризуется рядом преимуществ. Совместно с общей системой типов, общеязыковая среда выполнения CLR поддерживает возможность взаимодействия языков платформы .NET. Кроме того, платформа .NET предоставляет большую полнофункциональную библиотеку классов .NET Framework, а также, метаданные (Metadata), которые в соответствии с [2], представляют собой информацию о сборках, модулях и типах, составляющих программы в .NET.

Компилятор генерирует метаданные, а CLR и наши программы их используют. Когда загружается сборка и связанные с ней модули и типы, метаданные подгружаются вместе с ними.

2. Взаимодействие неуправляемой среды и высокоуровневых языков программирования, реализовано через OLE/COM модель. Из-за этого при разработке существует необ-

ходимость использования дополнительной «обёртки», обеспечивающей взаимодействие между OLE/COM и .NET. При использовании Visual Studio, разработчик может использовать стандартные средства для «маршалинга», т.е. передачи данных из неуправляемой среды в управляемую и обратно. При использовании языка программирования C# – это пространство имен *System.Runtime.InteropServices*, класс *Marshal*.

Для реализации браузера объектов необходимо установить соединение с запущенным приложением. Согласно SDK, для подключения используется метод *GetActiveObject*, класса *Marshal*.

GetActiveObject предоставляет COM-функцию GetActiveObjectFunction из динамической библиотеки OLEAUT32.DLL; однако последняя ожидает идентификатор класса (CLSID) вместо программного идентификатора (*ProgID*), ожидаемого данным методом. Чтобы получить запущенный экземпляр COM-объекта без зарегистрированного *ProgID*, используется вызов платформы для определения COM-функции GetActiveObjectFunction (1).

При выполнении строки (1) происходит «маршалирование» неуправляемого приложения из OLE/COM среды в объект .Net платформы, после чего появляется возможность получать информацию о нем, и производить над ним какие-либо действия.

После установления соединения, необходимо извлечь информацию об объекте *objApplication*. Для этого используется метод *GetTypeInfo()*, пространства имен *System.Runtime.InteropServices*, который возвращает сведения о типе объекта. Эти сведения затем могут использоваться для получения информации о типе интерфейса (2).

В (2) rootObject – объект objApplication, с которым установлено соединение. Строковые переменные передаются пустые, с типом out.

Следующий этап предполагает получение дочерних объектов, для корневого (или любого другого, имеющего дочерние объекты). Для этого используется структура (3):

Для создания монитора событий, важно определить, что представляет собой событие при работе с неуправляемым приложением. Согласно SDK каждого приложения, существует библиотека или их набор, которые содержат коллекцию объектов, в которой определены все события. Реализация обнаружения каждого из событий и их запись возможна при использовании точек подключения, используя Framework 4, 4.5. Интерфейс *IConnectionPointContainer* пространства имен *System.Runtime.InteropServices.ComTypes* используется следующим образом: точки подключения обеспечивают двустороннее взаимодействие между клиентом и сервером в модели COM. С помощью этого механизма COM-сервер может обратиться к клиенту при возникновении какого-либо события. Сервер может вызвать событие для оповещения соответствующей клиентской программы о каком-либо изменении (например, об изменении заголовка).Чтобы подготовиться к получению входящих уведомлений, клиент создает внутренний COM-объект, называемый приемником событий. Получив уведомление, клиент может выполнить действия, связанные с данным событием. Приемник событий содержит интерфейс для предоставления серверу методов, связанных с событием. Серверы порождают события путем вызова методов, связанных с событиями. Клиент реализует интерфейс приемника событий как обычный COM-интерфейс. Сервер объ-

```
objApplication = (appFramework.Application)Marshal.GetActiveObject(«app.Application»);   (1)

((System.Runtime.InteropServices.IDispatch)rootObject).GetTypeInfo(0, 0).
GetDocumentation(-1, out strName, out strDocString, out dwHelpContext, out strHelpFile);   (2)

_funcDesc = pFuncDesc.ToStructure<System.Runtime.InteropServices.ComTypes.FUNCDESC>();

_comTypeInfo.GetITypeInfo().GetDocumentation(_funcDesc.memid, out _name,
out _description, out _helpContext, out _helpFile);   (3)
```

является интерфейс как исходящий; создатель COM-сервера применяет атрибутsourcек интерфейсу в библиотеке типов COM-сервера. После определения интерфейса приемника событий приемник должен быть подключен (привязан) к источнику. Механизм точек подключения использует для связывания приемника и источника следующий протокол:

Приемник запрашивает у объекта сервера интерфейс *IConnectionPointContainer*. Если объект поддерживает точки подключения, он возвращает указатель.

Получив нужный объект точки подключения, приемник вызывает метод *IConnectionPoint.Advise()* для регистрации указателя интерфейса приемника. Сервер (источник) поддерживает подключение (и порождает через него события) до тех пор, пока клиент не разорвет подключение с помощью метода *IConnectionPoint.Unadvise()*.

Общую работу системы можно описать следующим образом: при вызове подсистемы конструктор класса инициализирует данные подсистемы и обрабатывает их. Появляется главная форма подсистемы, отвечающая за реализацию управляемого интерфейса, которая предоставляет пользователю возможность выбирать нужный ему модуль системы мониторинга. При выборе необходимых параметров, пользователь должен получить информацию об объекте, работа с которым происходит в текущий момент. Заключительным этапом будет являться вывод информации системой мониторинга на экран. В любой момент времени пользователь может получить информацию о модуле, информацию о разработчике, а также ознакомится со справочным руководством. Работу приложения можно представить в виде упрощенной функциональной модели, приведенной на рис. 1.

Система может быть запущена как до запуска неуправляемого кода, так и после. Как только приложение запущено, оно начинает «прослушивать» запущенный экземпляр клиента. При перехвате какого-либо изменения, происходит обновление всей выведенной на экран информации.

При этом система позволяет отслеживать изменения состояний свойств, методов, событий и их параметров в реальном времени, при работе в неуправляемой среде. Это открывает реальные возможности для быстрого расширения любых систем.

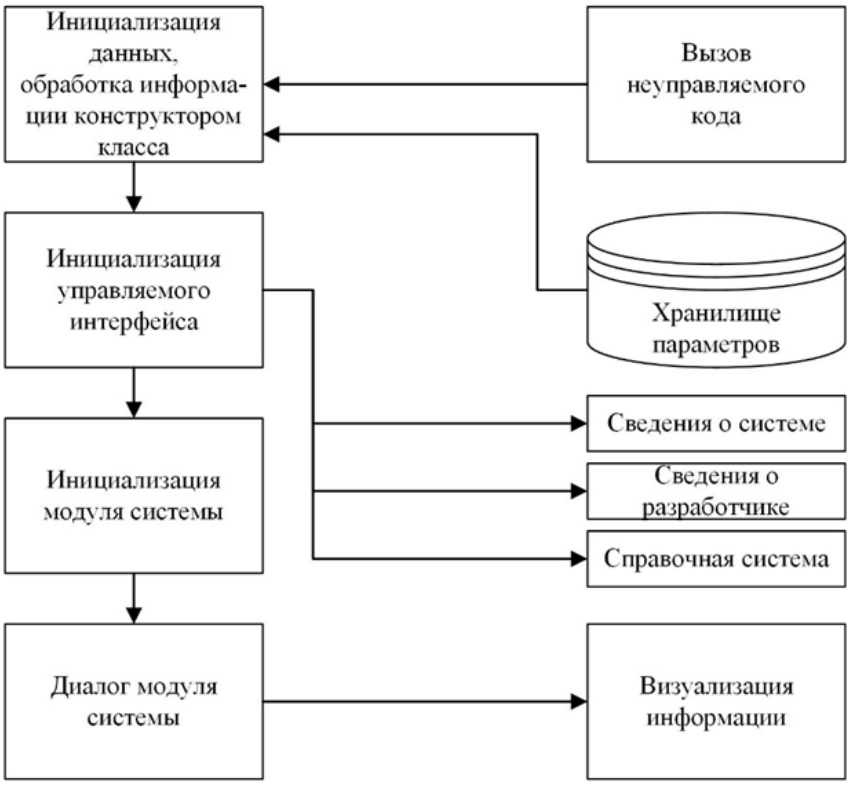

Рис. 1. Функциональная модель системы мониторинга

Таким образом, мною показана возможность применения данной методики на практике и построения на ее основе реальных программных систем мониторинга.

СПИСОК ЛИТЕРАТУРЫ

1. MSDN – Microsoft Software Developer Network. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/8bs2ecf4(v=vs.110).aspx. – Дата доступа: (28.02.2015).

2. *Мартынов А*. Метаданные в среде .Net. RSDN – Russian Software Developer Network. – Режим доступа: http://rsdn.ru/ ?article/dotnet/refl.xml. – Дата доступа: (28.02.2015).

Рыжков В. А. – Воронежский государственный технический университет, ф-т ИТКБ, старший преподаватель кафедры Компьютерных интеллектуальных технологий проектирования.
Тел.: (473) 255-42-48
E-mail: ryzhkov@vorstu.ru

Ryzhkov V. A. – Voronezh State Technical University, Faculty of IT&CS, a senior lecturer in the Computer Program Of Intelligent Design Technologies.
Tel.: (473) 255-42-48
E-mail: ryzhkov@vorstu.ru