Опубликовано в Научно-техническом вестнике СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 273-281.

УДК 004.4’233

МЕТОД РАЗРАБОТКИ ТЕСТОВ ДЛЯ ПРОГРАММНЫХ ИНТЕРФЕЙСОВ ПРИЛОЖЕНИЙ НА ОСНОВЕ КОНЕЧНО-АВТОМАТНОЙ МОДЕЛИ ТЕСТИРОВАНИЯ

К.В. Рубинов, В.В. Веденеев, В.Г. Парфенов
(Санкт-Петербургский государственный университет информационных технологий, механики и оптики)

В работе описывается предложенный авторами метод разработки тестов для программных интерфейсов приложений на основе конечно-автоматной модели тестирования. Предлагаемый метод предполагает построение автоматной модели тестирования в графическом виде. После этого по этой модели с помощью обхода графа переходов строится набор тестовых сценариев. Такой подход к тестированию позволяет упростить и формализовать построение тестов не только для отдельных функций, но и для их совокупностей.
Ключевые слова: автоматное программирование, тестирование

Введение

Программные интерфейсы приложений (*Application Programming Interfaces – API*) проектируются на основе стандартов, которые поддерживают производители программного обеспечения. Соответствие стандартам контролируется и базируется на тестировании интерфейсов в соответствии со спецификациями. Все чаще для тестирования программных компонентов и приложений применяется тестирование на основе моделей (*Model-based testing*) [1, 2]. При этом используются различные модели, на базе которых разработаны разнообразные средства генерации тестов [3]. Анализ литературы показал, что модельный подход практически не применяется для тестирования *API*. К известным работам можно отнести, например, работу [4] о применении марковских моделей. Таким образом, рассмотрение других моделей в качестве базы для создания тестов программных интерфейсов приложений является весьма актуальной задачей.

В работе описывается метод разработки тестов для *API* на основе моделей. Учитывая достоинства и недостатки существующих моделей, в предложенном авторами методе для построения модели тестирования выбраны конечные автоматы, которые позволяют воспользоваться преимуществами теории автоматов для новой задачи.

Отличительной особенностью метода является то, что предлагается строить не модель поведения системы, а модель тестирования использования программных интерфейсов. При этом в качестве состояний выделяются тестируемые функции (методы), а в качестве событий (условий переходов) рассматриваются результаты выполнения функций. Для автоматизации задач, возникающих при разработке тестов для *API*, в работе используются существующие подходы преобразования моделей конечных автоматов в *XML*-формат с последующей генерацией из полученного представления тестовых сценариев и исходного кода тестового приложения. Предложенный метод разработки тестов позволяет решить задачу их создания для *API*, а также автоматизировать некоторые из этапов решения этой задачи. Применение метода демонстрируется на примере.

Постановка задачи

В работе рассматривается процесс разработки тестов для функционального тестирования программных интерфейсов приложений (далее интерфейсов). Эти интерфейсы предназначены для предоставления другим программам функциональности программной системы, в которой они реализованы.

Для проверки соответствия интерфейса спецификации необходимо выполнить следующие виды тестирования.

1. Синтаксическое тестирование отдельных функций интерфейса. Проверка отдельных функций на широком спектре входных данных.
2. Проверка корректности предоставления функциональности отдельных функций интерфейса.
3. Проверка корректности предоставления функциональности совокупностью функций интерфейса.
4. Проверка некорректного применения отдельных функций интерфейса.
5. Проверка некорректного использования совокупности функций интерфейса.
6. Проверка взаимодействия при интеграции функций интерфейса с другими интерфейсами или программными компонентами.

Особый интерес из перечисленных выше видов тестирования представляет проверка корректности предоставления функциональности совокупностью функций интерфейса, так как на данный момент для него не существует формализованного метода разработки тестов.

В работе описывается метод разработки тестов для совокупностей функций интерфейсов. Предлагается конечно-автоматная модель тестирования, а также рассматривается вопрос о тестовом покрытии на основании предложенной модели.

Результатом применения описываемого метода является приложение, которое тестирует функции программного интерфейса в соответствии со спецификацией.

Спецификация API

Процесс тестирования *API* начинается с изучения и анализа заданной спецификации тестируемого интерфейса. Обычно спецификации к интерфейсам описывают функциональное назначение отдельных функций (методов) интерфейса, а также ограничения на используемые через интерфейс данные.

Для процесса тестирования, базирующегося на спецификации, критерием тестового покрытия является разработка такого числа тестов, чтобы был создан как минимум один тест для каждого требования из спецификации.

При успешном прохождении определенного числа тестов, которое удовлетворяет заранее заданному значению, можно утверждать, что система прошла аттестационное/сертификационное тестирование (*conformance/certification testing*).

Рассмотрим процесс тестирования *API* на примере.

Пример спецификации API

В качестве примера спецификации *API* рассмотрим часть программного интерфейса операционной системы, предназначенного для работы с файлами [5]. Для тестирования выделим минимальный набор операций, необходимых для чтения/записи файла: открытие файла с необходимым уровнем доступа, закрытие файла, позиционирование по файлу, чтение и запись последовательности байт с определенного места в открытом файле.

Приведем функции, отвечающие в спецификации за эти операции, в упрощенной нотации и кратко опишем их функциональность.

Функция

```
HANDLE CreateFile(…)
```

позволяет открывать или создавать файл и возвращает указатель на него. Этот указатель необходимо передавать во все функции, которые работают с открытым файлом. Если при его открытии произошла ошибка, то вместо корректного указателя на файл

возвращается зарезервированный код ошибки. В качестве параметров функция принимает название файла, желаемый уровень доступа и т.д.

Функция

```
BOOL CloseHandle(HANDLE)
```

позволяет закрывать файл. В случае успеха возвращается значение `TRUE`, иначе – `FALSE`. Единственным параметром является указатель на открытый файл.

Функция

```
DWORD SetFilePointer(HANDLE, …)
```

позволяет установить указатель на место начала чтения или записи в открытом файле. Если операция позиционирования прошла успешно, то возвращается указатель на место в файле, иначе возвращается отрицательное число. В качестве параметров функция принимает указатель на открытый файл и параметры, определяющие новое место в файле.

Функция

```
BOOL ReadFile(HANDLE, …)
```

позволяет читать данные из открытого файла. В случае успеха возвращается значение `TRUE`, иначе – `FALSE`. В качестве параметров функция принимает указатель на открытый файл, параметры, определяющие адрес и размер буфера для записи прочтенных данных, и число байт для чтения.

Функция

```
BOOL WriteFile(HANDLE, …)
```

позволяет записывать данные в открытый файл. В случае успеха возвращается значение `TRUE`, иначе – `FALSE`. В качестве параметров функция принимает указатель на открытый файл, параметры, определяющие адрес и размер буфера для чтения записываемых данных, и число байт для записи.

Анализ спецификации API

После изучения спецификации перейдем к следующему этапу – инженер по тестированию проводит анализ спецификации и предметной области с целью выделения возможных вариантов совместного использования совокупности функций интерфейса. При этом формируются последовательности вызовов функций. Из этих последовательностей в дальнейшем будут формироваться тестовые сценарии. На основании анализа зависимостей в спецификации функции программного интерфейса можно разбить на группы по типу выполняемой функциональности.

Проведем неформальный анализ описанного выше примера спецификации программного интерфейса. Рассмотрим операции открытия и закрытия файла.

Для тестирования функции `CreateFile` в наиболее простом случае необходимо выполнить два теста:

- успешно открыть файл;
- получить ошибку при открытии файла.

Учитывая известные методики тестирования [6], число этих тестов необходимо расширить. Например, можно выделить следующие ситуации, требующие проверки:

- создать файл с названием, содержащим недопустимые символы;
- создать файл с названием в однобайтной или двухбайтной кодировке;
- создать файл по несуществующему пути;
- создать существующий файл;
- открыть уже открытый файл и т. д.

Для тестирования функции `CloseHandle` необходимо провести следующие тесты:

- успешно закрыть файл;

- получить ошибку при закрытии файла.

Составим тестовый сценарий для успешного закрытия файла. Для этого в функцию `CloseHandle` необходимо передать указатель на открытый файл. Однако этот указатель можно получить только с помощью еще непротестированной функции `CreateFile`.

Отложим написание тестового сценария для функции `CloseHandle` и приступим к проверке функции `CreateFile`. При тестировании функции `CreateFile` необходимо иметь в виду, что при завершении теста следует закрыть открытые файлы с помощью функции `CloseHandle` для освобождения ресурсов. При этом функция `CloseHandle` еще не протестирована.

Тестовые сценарии для функций программного интерфейса неявно опираются на то, что остальные его функции работают корректно и в пределах каждого теста происходит дублирование тестов для других функций.

В результате анализа спецификации выявлены возможные варианты совместного использования совокупности функций интерфейса. Далее на основе полученных вариантов осуществляется построение модели тестирования *API*.

Модель тестирования API

Для разработки тестовых сценариев строится модель, описывающая использование функций тестируемого интерфейса. Модель необходима для формализации процесса разработки тестовых сценариев и тестовых приложений. Применение модели позволяет автоматизировать некоторые этапы процесса разработки тестов, а также в дальнейшем, при внесении изменений в спецификацию, поддерживать соответствие между спецификацией тестируемого интерфейса и тестовым приложением.

Построение модели тестирования API

В работе для построения модели выбраны конечные автоматы. Такой выбор обусловлен несколькими факторами:

- наглядное визуальное представление модели в форме конечного автомата;
- удобство отображения связей между возможными вызовами функций интерфейса в конечном автомате;
- существование широкого спектра алгоритмов, методов и средств работы с моделями в форме конечных автоматов.

Конечно-автоматная модель тестирования API

Модель системы, как отмечено выше, представляется в виде конечного автомата, в котором состояниям соответствуют вызовы функций, а условиями перехода являются результаты выполнения функций. Вызовы функций осуществляются при входе в состояния. Наличие действий в выходных воздействиях таких автоматов не является обязательным, но они могут быть. Действия могут вводиться искусственно для вызова промежуточных функций, которые не являются тестируемыми, но необходимы для поддержания тестового приложения в необходимом состоянии (например, вызов функции обновления экрана). Инициализация перед запуском отдельных функций программного интерфейса представляется в виде состояний автомата.

Поясним это на примере (рис. 1).

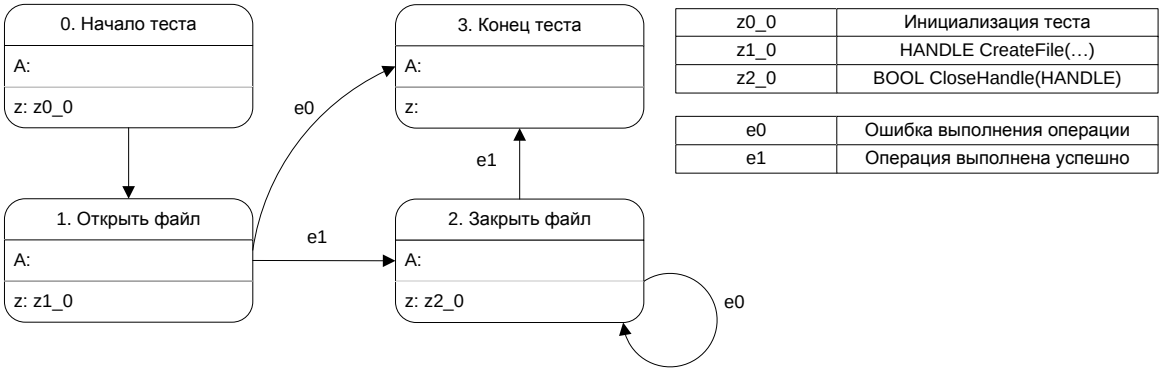

Рис. 1. Модель тестирования функций открытия и закрытия файла

Конечный автомат имеет четыре состояния. Состояния 0 и 3 соответствуют началу и концу теста. В состояниях 1 и 2 вызываются функции `CreateFile` и `CloseHandle` соответственно.

Из начального состояния 0 после проведения необходимой инициализации автомат переходит в состояние 1. В случае неуспешного открытия файла в состоянии 1 происходит окончание теста – переход в состояние 3. При успешном открытии файла автомат переходит в состояние 2. Если файл успешно закрывается в состоянии 2, то происходит окончание теста в состоянии 3, иначе автомат остается в состоянии 2.

Представив возможные зависимости между функциями программного интерфейса в виде конечного автомата, удается решить проблемы, описанные выше. Из рассмотрения графа на рис. 1 следует, что обе функции (`CreateFile` и `CloseHandle`) можно протестировать только в совокупности.

На основании построенного графа переходов можно получить необходимые тестовые сценарии: любой путь из начального состояния в конечное является тестом. Например, путь 0 – 1 – e0 – 3 является описанным выше тестом «Получить ошибку при открытии файла». Это верно и для других путей в графе.

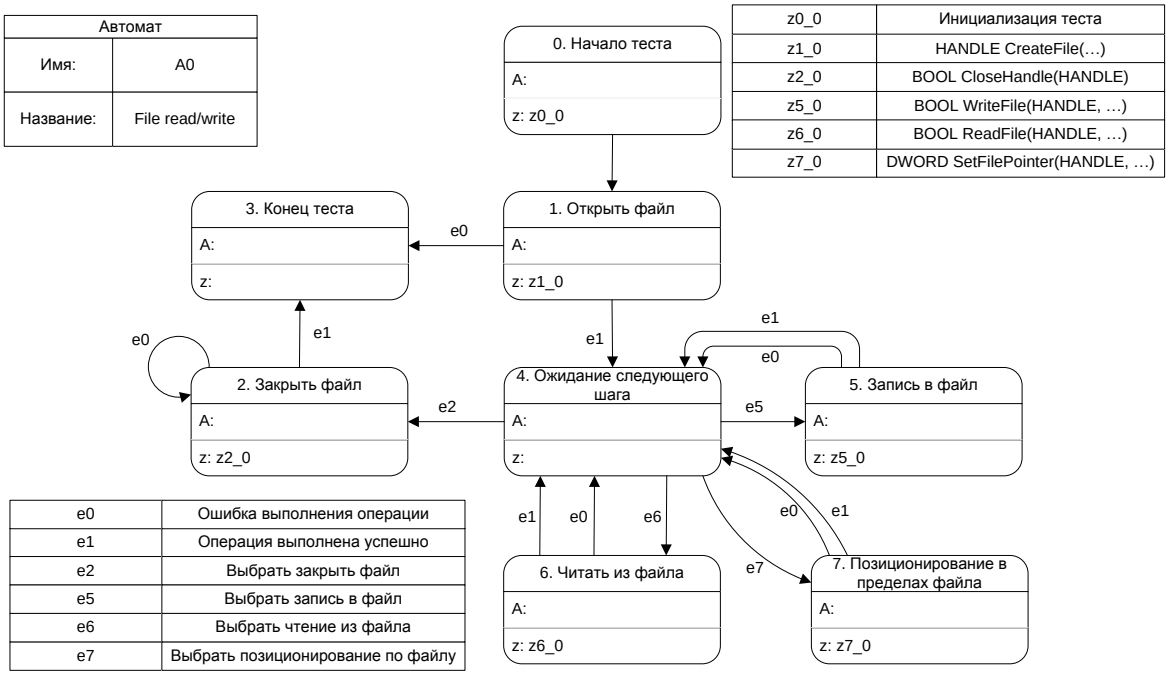

Рис. 2. Моделирование конечным автоматом использования программного интерфейса

Отметим известный факт, что число возможных путей в графе с циклами бесконечно. Поэтому необходимо использовать методы ограничения числа путей для получения приемлемого числа тестов по составленной модели.

Продолжая усложнять созданную модель, включим в нее все описанные в примере функции программного интерфейса (рис. 2).

В данном случае отметим состояние 4. Это «искусственное» состояние, введенное для упрощения модели. В этом состоянии не производится непосредственный вызов тестируемой функции, и его можно рассматривать как ветвление для создания той или иной тестовой последовательности. Из этого состояния под действием искусственных событий e2, e5, e6 и e7 автомат переходит в состояния 2, 5, 6 и 7 соответственно.

Введение искусственного состояния позволило упростить модель, увеличивая при этом число возможных тестовых сценариев.

Преобразование автоматной модели в XML-формат

Граф переходов разработанной конечно-автоматной модели предлагается хранить в формате *XML* [7]. Этот формат является одним из наиболее распространенных форматов хранения динамических данных. Следующие преимущества явились определяющими при выборе этого формата хранения данных:

- простота понимания и доступность;
- простота изменения и дополнения хранимых данных;
- простота преобразования и конвертирования хранимых данных;
- расширяемость.

Для хранения графа переходов предлагается следующая структура *XML*-документа:

- таблица состояний (*States*) – хранит описания всех состояний приложения;
- таблица событий (*Events*) – хранит описания всех возможных событий;
- таблица переходов (*Switches*) – связывает таблицу состояний и таблицу событий.

В табл. 1–3 приведены описания используемых полей.

Таблица 1. Описания полей таблицы *Events*

Поле	Описание
StateId	Уникальный идентификатор
Description	Описание сущности

Таблица 2. Описания полей таблицы *States*

Поле	Описание
StateId	Уникальный идентификатор
Description	Описание сущности

Таблица 3. Описания полей таблицы *Switches*

Поле	Описание
StateFromId	Из какого состояния осуществляется переход
StateToId	В какое состояние осуществляется переход
EventId	Под действием чего осуществляется переход

Предложенный формат является базовым. По мере необходимости он может легко расширяться и дополняться. Например, в таблицу событий можно добавить поля, отвечающие за специфические события.

Модель в виде *XML*-файла можно получить из обычного визуального представления. В качестве системы визуального проектирования воспользуемся, например, программой *Microsoft Visio 2003*. Эта программа, как и все продукты *Microsoft Office 2003*, позволяет сохранять разработанные документы в *XML*-формате. Для представления автоматов воспользуемся инструментальным средством *Visio2Switch* [8].

С помощью *XSL*-трансформации [9] документ, сохраненный программой *Visio*, преобразуется в требуемый формат.

Далее будет показано, как на основе разработанной модели можно автоматизировать последующие этапы процесса разработки тестов для *API*.

Тестовое приложение

На данном этапе происходит формирование тестовых сценариев и создание каркаса тестового приложения. На основе разработанной модели тестирования автоматизируется процесс создания приложения для тестирования *API*.

Формирование тестовых сценариев

Как было отмечено выше, каждый тестовый сценарий является некоторым путем в графе переходов конечного автомата, представляющего модель использования программного интерфейса. Начальная вершина пути – это состояние приложения перед началом теста. Путь по вершинам графа – это последовательность вызовов функций во время проведения теста. Конечная вершина пути – это ожидаемый результат проделанных действий. Таким образом, можно рассматривать модель как первичную сущность, а элементарный тест – как вторичную. В этом случае изменение спецификации программного интерфейса отразится только на изменении модели, а тестовые сценарии будут обновлены по модели.

Набор тестовых сценариев можно создать автоматически на основе разработанной модели – достаточно произвести обход графа модели. Задача поиска путей в графе описана в ряде источников (например, в работе [10]). При этом для создания тестовых сценариев и преобразования графа переходов, представленного в *XML*-формате, повторно выполняется *XSL*-трансформация. Так как число возможных сценариев в общем случае бесконечно, у разработанной трансформации есть два ограничивающих параметра: максимальное число проходов по циклу и максимальная длина сценария.

Помимо этого, в рамках настоящего метода можно опитимизировать создание тестовых сценариев – задача поиска минимального числа тестовых сценариев, наиболее полно покрывающих функциональность, является задачей, которая может решаться отдельно. Решения этой задачи описаны, например, в работе [6]. Тем самым можно говорить о том, что предлагаемый метод не ограничивает использование вновь появляющихся технологий и алгоритмов.

Каркас тестового приложения

Используя описанную выше модель и тестовые сценарии, создается тестирующая программа. Так же, как и на предыдущих этапах, это выполняется с помощью *XSL*-трансформации. Для этого разрабатываются конфигурационные файлы, с помощью которых осуществляется переход от *XML*-представления тестовых сценариев к представлению на используемом языке программирования – создается каркас тестового приложения.

XSL-трансформацию можно легко модифицировать для использования необходимого языка программирования и конкретной среды для запуска тестов. Как и другие этапы автоматизации, этот этап является независимым.

Созданный каркас приложения содержит заглушки функций, которые необходимо реализовать вручную инженеру по тестированию. Каждая функция проверяет генера-

цию конкретного события в определенном состоянии. Эти функции могут быть реализованы один раз и впоследствии использоваться при изменении модели.

Последним шагом подготовки тестового приложения является насыщение его каркаса данными для тестирования. Они выбираются по спецификации в соответствии с используемыми техниками тестирования. Задачи выбора данных для тестирования изложены во многих работах (например, в работе [6]).

Тестовые сценарии запускаются из меню приложения в автоматическом режиме. Результатом выполнения приложения является вердикт: тест прошел – положительный результат (`PASSED`) или тест не прошел – отрицательный результат (`FAILED`).

Заключение

Предложенный метод позволяет разрабатывать тесты для программных интерфейсов приложений на основе спецификаций. При этом увеличивается вероятность обнаружения неисправностей, не только связанных со свойствами отдельных функций, но и с общей функциональностью системы, предоставляемой через интерфейс.

Использование конечно-автоматной модели обеспечивает прозрачность структуры и поведения разрабатываемых тестовых приложений. Кроме того, применение модели позволяет автоматизировать определенные этапы создания тестов.

В рамках настоящего метода имеются возможности для улучшений и будущих исследований. Наиболее интересными представляются следующие направления работ:

- извлечение паттернов использования *API* из существующих приложений и репозиториев кода [11] и применение их в методе создания тестов;
- оптимизация обхода модели тестирования с использованием различных алгоритмов.

Литература

1. El-Far I.K., Whittaker J. Model-Based Software Testing. Encyclopedia on Software Engineering (edited by Marciniak J.). – Wiley. 2001.
2. DACS (Data and Analysis Center for Software) Gold Practices Website. Model-based testing. – Режим доступа: https://www.goldpractices.com/practices/mbt/
3. Hartman A. AGEDIS model based test generation tools. – Режим доступа: http://www.agedis.de/documents/ModelBasedTestGenerationTools.pdf
4. Jorgensen A., Whittaker J. An API Testing Method /Proceedings of the International Conference on Software Testing Analysis & Review (STAREAST 2000). Software Quality Engineering. Orlando. 2000.
5. File Management Functions, MSDN. – Режим доступа: http://msdn2.microsoft.com/en-us/library/aa364232.aspx
6. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб.: Питер. 2004.
7. XML Developer Center. – Режим доступа: http://msdn.microsoft.com/xml
8. Головешин А. Конвертор Visio2Switch. – Режим доступа: http://www.geocities.com/goloveshin/v2s_rus.htm
9. XSL Transformations. – Режим доступа: http://www.w3.org/TR/xslt
10. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. – М.: МЦНМО. 2001.
11. Acharya M., Xie T., Pei J., Xu J. Mining API patterns as partial orders from source code: from usage scenarios to specifications / In Proc. ESEC/FSE 2007. – Режим доступа: http://people.engr.ncsu.edu/txie/publications/esecfse07.pdf