УДК 004.54

А.А. Немеров, И.В. Бойченко

Применение технологии UniTESK для тестирования сервис-ориентированных систем

Статья содержит краткое описание технологии UniTESK и применение этой технологии для тестирования сервис-ориентированных систем. Представлено построение тестовой системы для тестирования реализации стандарта IEEE 802.16 (WiMAX).

Ключевые слова: сервис-ориентированная система, UniTESK, CTESK, WiMAX, тестирование.

Введение в технологию UniTESK. UniTESK [1] – унифицированное решение для промышленного тестирования и обеспечения качества программного обеспечения. Технология UniTESK пронизывает весь жизненный цикл разработки программного обеспечения от сбора и анализа требований до сопровождения. Основанная на опыте реальных промышленных проектов в компаниях со сложившейся культурой разработки, технология UniTESK не требует для внедрения коренной перестройки процессов, она легко сочетается с другими подходами к тестированию и обеспечению качества, обогащая их возможности и обогащаясь при этом сама. Красной нитью сквозь все элементы технологии UniTESK проходит прослеживаемость (англ. traceability) требований от этапа анализа потребностей пользователей до выпуска конечного продукта. От UniTESK наследует основные черты технология для тестирования программного обеспечения, реализованного на языке C – CTESK.

CTESK реализует технологию UniTESK автоматизированного тестирования, основанного на спецификациях. Это разновидность функционального тестирования, целью которого является проверка поведения системы в различных условиях на соответствие требованиям. Требования для тестированию описываются в виде формальных спецификаций на языке SeC (расширение языка C). CTESK отличается от других инструментов для функционального тестирования тем, что напрямую не описывает необходимые тесты. Цепочки тестов генерируются автоматически на основе заданных сценариев и состояний.

Инструменты, подобные CTESK, применяются к программному обеспечению с повышенными требованиями к надежности. Поэтому наиболее перспективными областями применения CTESK являются:

1. Критическое программное обеспечение (встроенные системы, системы управления промышленным производством, медицинские системы мониторинга).
2. Системное программное обеспечение (компоненты, реализующие службы ОС, веб-серверы).
3. Телекоммуникационное программное обеспечение.
4. Любое программное обеспечение, соответствующее стандартам (протоколы, интернет-приложения).

CTESK повышает качество процесса разработки программного обеспечения за счет того, что тесты разрабатываются независимо от реализации на основе спецификаций требований. Опережающая разработка тестов позволяет сократить цикл разработки программного обеспечения и повышает качество тестирования. CTESK является мощным средством для проведения блочного и интеграционного тестирования.

Спецификации CTESK. Требования к функционалу описываются в спецификациях на языке SeC. Объявление спецификационной функции состоит из ключевого слова *specification* и сигнатуры функции. Пример: *specification double sqrt_spec(double x).*

SeC предусматривает три конструкции, предназначенные для включения в тело спецификационной функции: предусловие, постусловие и локальное определение покрытия.

```
specification double sqrt_spec(double x) {
 pre { ... }
 coverage C { ... }
```

```
  post { ... }
}
```

Предусловие и критерии покрытия могут отсутствовать, тогда как постусловие должно присутствовать обязательно. Между описанными блоками может использоваться дополнительный код, который не должен вызывать побочных эффектов.

Пример реализации спецификационной функции для вычисления квадратного корня:

```
specification double sqrt_spec(double x) {
  pre {
    return x >= 0.0;
  }
  post{
    return (abs(sqrt_spec * sqrt_spec – x) <= EPS);
  }
```

sqrt_post в постусловии означает результат, полученный при выполнении самой функции извлечения квадратного корня [2].

Медиаторные функции. Спецификации являются хорошим способом описать требования к системе независимо от реализации самой системы. Чтобы связать спецификации и тестируемую систему, используются медиаторные функции. Определение медиаторной функции выглядит следующим образом:

mediator sqrt_media *for specification* double sqrt_spec(double x).

Ключевое слово *mediator*, за ним имя медиаторной функции, затем *for specification* и сигнатура спецификационной функции, которую нужно связать с реализованной системой. Пример описания медиаторной функции ниже:

```
mediator sqrt_media for specification double sqrt_spec(double x)
{
  call {
    return sqrtmy(x);
  }
}
```

В конструкции *call* происходит привязка к интерфейсной функции реализованной системы.

Сценарии. Тестовый сценарий предоставляет всю информацию, необходимую для автоматического построения теста. Он соответствует переменной специального структурного типа dfsm или ndfsm, помеченной модификатором scenario:

scenario dfsm testScenario.

В качестве имени типа выступает название обходчика, определяющего механизм построения теста:

1) dfsm – Deterministic Finite State Machine (детерминированный конечный автомат);
2) ndfsm – Nondeterministic Finite State Machine (недетерминированный конечный автомат).

Во время тестирования dfsm применяет тестовые воздействия, которые могут изменять состояние сценария. dfsm автоматически отслеживает все изменения состояния и строит конечный автомат, соответствующий процессу тестирования. Состояниями автомата являются все достижимые состояния сценария, а переходы автомата помечаются тестовыми воздействиями, которые их инициировали. Тестирование заканчивается только тогда, когда обходчик подаст все определенные пользователем тестовые воздействия во всех состояниях сценария достижимых из начального.

Описание сценария выглядит следующим образом:

```
scenario dfsm testScenario = {
  .init = init,
  .getState = getState,
  .actions = {
    f_scen,
    g_scen,
    NULL
  },
  .finish = finish
};
```

В поле init задается функция инициализации. Это поле может быть опущено, однако обычно функция инициализации присутствует как минимум для установки медиаторов. В поле getState задается функция построения сценарного состояния. Если это поле опущено, то тестирование ведется в одном состоянии. В поле actions задается список сценарных функций, включенных в данный тест,

завершенный значением NULL. Это поле является обязательным. В поле finish задается функция завершения. Это поле может быть опущено. Тестовый сценарий вызывается как функция с идентификатором тестового сценария, принимающая два параметра, аналогичных параметрам main, без возвращаемого значения.

Применение CTESK для тестирования сервис-ориентированной системы на примере QOS для WiMAX. Любую сервис-ориентированную систему можно условно разделить на 2 части: ядро системы и сервисы, работающие в системе. Ядро системы организует всю работу системы, отвечает за размещение и функционирование сервисов, связь между ними и предоставляет API для создания сервисов. Сервисы, как правило, пишут сторонние разработчики, не имеющие доступа к исходному коду системы.

В качестве тестируемой системы представлена сервис-ориентированная система, написанная на C++ (будем называть ее целевой системой). Конечное предназначение целевой системы – реализация протокола IEEE 802.16 (WiMAX) [3]. Исходный код ядра системы и ряда сервисов, недоступен, так как разрабатывается отдельной рабочей группой. Каждый сервис собирается в виде отдельного плагина (so библиотеки), которые можно заменять при запуске системы. Сервисы взаимодействуют посредством передачи друг другу сообщений, в которые могут упаковываться данные. Необходимо организовать в системе функциональное тестирование отдельных сервисов.

Проблема тестирования отдельных сервисов в целевой системе и интеграция с инструментом CTESK была решена согласно схеме (рис. 1).

Рис. 1. Структурная схема тестирования сервис-ориентированной системы

На рисунке представлены отдельные модули целевой и тестирующей системы. Связь между целевой и тестирующей системой организована через сокеты. К целевой системе добавлено 2 дополнительных тестовых сервиса: тестовый сервис-приемник и сервис-отправитель. Цель этих сервисов – обеспечить интерфейс между целевой и тестовой системой. Тестовый сервис-приемник слушает порт в ожидании сообщения от тестовой системы и транслирует это сообщение в термины целевой системы. У тестового сервиса-отправителя обратная функция: он переводит сообщение в термины тестовой системы и пишет его в сокет. Во избежание взаимных блокировок для принятия и отправки сообщений используются разные сокеты. Также взаимодействие через сокеты окажется полезным на следующих стадиях разработки, когда целевая система будет развернута на целевом оборудовании.

Сложность тестирования сервис-ориентированной системы в том, что она: 1) имеет собственный поток (потоки) управления; 2) может не сразу отвечать на тестовые воздействия. Для тестирования таких систем в CTESK имеется механизм отложенных реакций [4]. Тестовая система после посылки некоторых тестовых импульсов ждет некоторое время пришествия реакций на эти импульсы, затем обрабатывает реакции в соответствующих функциях, устанавливая в этих функциях корректность работы целевой системы. В этом случае уже не тестовая система, а целевая является инициатором взаимодействия.

Ниже представлена диаграмма последовательности тестирования сервис-ориентированной системы. Опишем более подробно взаимодействия, представленные на диаграмме (рис. 2).

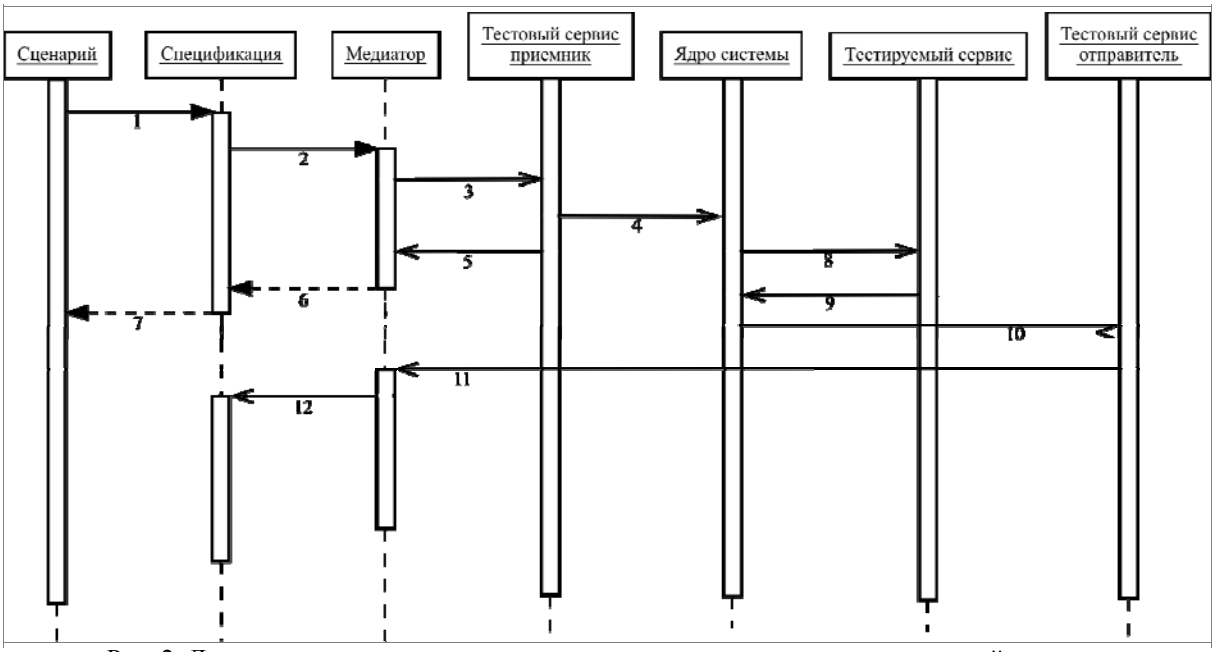

Рис. 2. Диаграмма последовательности тестирования сервис-ориентированной системы

1. Сценарная функция вызывает спецификационную функцию.
2. Спецификационная функция вызывает медиаторную функцию.
3. Медиаторная функция посылает в сокет информацию о тестовом импульсе и необходимые данные.
4. Тестовый сервис-приемник формирует запрос к тестируемому сервису и отправляет его.
5. Тестовый сервис посылает медиатору уведомление, что запрос направлен.
6. Возврат из медиаторной функции.
7. Возврат из спецификационной функции.
8. Посылка сообщения тестируемому сервису.
9. Ответ тестируемого сервиса.
10. Возврат результата запроса тестовому сервису-отправителю.
11. Тестовый сервис-отправитель посылает результат перехватчику отложенных реакций в медиаторе.
12. Перехватчик реакций вызывает функцию обработчика отложенных реакций.

Сообщения 1–7 – это посылка импульса в целевую систему, инициатором является тестовая система. Сообщения 10–12 – это реакции тестового целевой системы, которые фиксируются и обрабатываются тестовой системой, инициатор – тестовая система. Сообщения 8–9 – обработка тестовых импульсов внутри целевой системы.

Тестирование обработки события входа абонентской станции в сеть. Вход абонентской станции (АС) в сеть WiMAX состоит из 9 шагов [5]:

1. АС сканирует исходящий канал.
2. Синхронизация.
3. Ранжирование.
4. Установка параметров передачи.
5. Авторизация и обмен ключами.
6. Регистрация АС в сети.
7. Установка IP-соединения.
8. Установка текущего времени.
9. Передача действующих параметров.

Подсистема QoS участвует в двух шагах: 3 и 6, соответственно их тестирование необходимо. Рассмотрим более подробно тестирование шага 3.

На этапе ранжирования АС посылает базовой станции (БС) сообщение RNG_REQ (ranging request) по специально выделенному для этого каналу. Сообщение включает в себя MAC-адрес АС, по которому производится идентификация станции. БС, получив запрос, должна сформировать ответ –

RNG_RSP (ranging response). В ответе содержится информация о физических характеристиках (мощность АС, временное смещение в канале), MAC-адрес АС и другие поля, среди которых идентификатор основного соединения (basic CID*)* и идентификатор первичного соединения (primary CID) [3]. Задача подсистемы QoS на данном этапе – создать основной и первичный потоки и сообщить эту информацию тому сервису, который формирует RNG_RSP.

Тестировать на данном шаге необходимо следующие условия:

- на каждый запрос должен приходить ответ;
- MAC-адрес в запросе и в ответе на этот запрос должны совпадать;
- в очереди потоков должны добавиться 2 новых потока.

Тестируемые требования были оформлены в виде спецификаций. Спецификация для проверки MAC-адреса в несколько упрощенном виде выглядит следующим образом:

```
reaction List* rngRSPMAC_spec(void)
  updates waitingList, uniqueCIDs, ulflowsRTsize, dlflowsRTsize
{
  post {
    int j;
    for(j = 0; j < size_List(rngRSP_spec); ++j) {
      int i;
      for(i = 0; i < 6; i++) {
        RNG_REQ* req = (RNG_REQ*)get_List(@waitingList, j);
        RNG_RSP* rsp = (RNG_RSP*)get_List(rngRSP_spec, j);
        if(req->mac[i] != rsp->mac[i]) {
          return false;
        }
      }
    }
    return true;
  }
}
```

Проверка MAC-адреса осуществляется в обработчике реакций, так как нам необходимо дождаться ответа от тестовой системы.

Кроме спецификаций, необходимо было добавить в тестовые сервисы возможность отправлять и получать RNG-сообщения и сделать транспортировку этих сообщений в тестовую систему. Здесь примеры расширения тестовых сервисов и медиаторов приводиться не будут в силу их громоздкости и малой информативности.

По окончании своей работы тестовая система формирует трассу прохождения тестов в формате XML. Трасса содержит все действия тестовой системы. Трасса плохо читабельна для человека, поэтому разработан специальный инструмент UniTESK Reports, преобразующий трассу в текстовый формат. UniTESK Reports помещает главную статистическую информацию о количестве пройденных тестов в начало файла. В качестве примера ниже представлена основная (статистическая) часть файла результата при выполнении тестов без ошибок.

```
States in raw trace: 234
Transitions in raw trace: 233
There are no failures in processed trace.
```

Если тесты нашли ошибки, то основная часть файла результата может выглядеть так:

```
States in raw trace: 14
Transitions in raw trace: 13
First failure happened in transition #11: Postcondition Failed.
```

По мере нахождения ошибок нужно их исправлять и повторять запуск тестов до тех пор, пока все тесты не пройдут успешно. Из файлов результатов мы можем получить подробности прохождения теста: переходы между состояниями, стек вызовов сценарных функций, значения переменных. Таким образом, мы можем регулярно проводить регрессионное функциональное тестирование реализации QoS для WiMAX.

Заключение. В работе был предложен метод тестирования систем с сервис-ориентированной архитектурой. Данный метод апробирован в ходе разработки сервиса QoS стандарта IEEE 802.16 – WiMAX (НИР по Государственному контракту № 13.G25.31.0011 от 07 сентября 2010 г.). В настоящее время разрабатываются и добавляются в систему новые сценарии тестирования подсистемы QoS. Также данный метод применен в ходе разработки сервис-ориентированной системы обработки

и анализа изображений, в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» (НИР по Государственному контракту № 14.740.11.0398; шифр заявки 2010-1.1-215-138-022).

Литература

1. UniTESK [Электронный ресурс]. – Режим доступа: http://unitesk.ru/, свободный (дата обращения: 07.09.2011).

2. CTESK 2.2: Руководство пользователя [Электронный ресурс]. – Режим доступа: http://www.unitesk.ru/download/papers/ctesk/CTesK2.2UserGuide.rus.pdf, свободный (дата обращения: 09.09.2011).

3. IEEE Standard for Local and metropolitan area networks. Part 16: Air Interface for Broadband Wireless Access Systems // IEEE Std 802.16-2009 (Revision of IEEE Std 802.16-2004). pp.C1–2004, May 29 2009. doi: 10.1109/IEEESTD.2009.5062485) [Электронный ресурс]. – Режим доступа: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5062485&isnumber=5062484, для зарегистрированных пользователей (дата обращения: 05.09.11).

4. Применение технологии UniTesK для тестирования систем с различной конфигурацией активных потоков управления [Электронный ресурс]. – Режим доступа: http://citforum.ru/SE/testing/unitest_use/, свободный (дата обращения: 07.09.2011).

5. Nuaymi L. WiMAX: Technology for Broadband Wireless Access. – Chichester, UK: John Wiley & Sons, 2007. – 310 p.

Немеров Александр Александрович
Аспирант каф. автоматизированных систем управления ТУСУРа
Тел.: 8-923-401-27-87
Эл. почта: nemerov@asu.tusur.ru

Бойченко Иван Валентинович
Докторант ТУСУРа, канд. техн. наук, доцент каф. автоматизированных систем управления ТУСУРа
Тел.: 8-906-958-24-83
Эл. почта: biv@asu.tusur.ru

Nemerov A.A., Boichenko I.V.

Using of UniTESK technology for service-oriented system testing

This article contains a description of the UniTESK technology and using of this technology to test the service-oriented systems. Building of test system for testing the implementation of the standard IEEE 802.16 (WiMAX) is represented.

Keywords: service-oriented system, UniTESK, CTESK, WiMAX, testing.
